Thomas Johansen

Autonomous Path Planning for Auto Docking of Ferries

Master's thesis in Marine Technology
Supervisor: Dong Trong Nguyen
June 2019

Master's thesis

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

NTNU
Norwegian University of
Science and Technology

Thomas Johansen

Autonomous Path Planning for Auto Docking of Ferries

Master's thesis in Marine Technology
Supervisor: Dong Trong Nguyen
June 2019

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

Preface

This thesis is the final part of my masters degree from NTNU. The topic is a result of my own interest in autonomous ships, and a thesis proposition from Rolls-Royce Marine, now Kongsberg Maritime, to research path planning for autonomous docking of ferries. This combination of personal interest and a possibility to collaborate with the industry provided the perfect motivation for this thesis.

The thesis has been a good way for me to combine most of the different courses I have taken at NTNU and really go deeply into the topic of autonomous ships and control of these. After this thesis, and the work I have done as a part of it, I see a clear development towards more autonomy the marine industry, but also a clear need for more research and development to make the systems safe and efficient. Hopefully parts of this thesis can be used and improved as a small part of this development.

Thomas Johansen

Thomas Johansen

Trondheim, June 11, 2019

Acknowledgment

I would like to thank Dong Trong Nguyen for being my supervisor for this thesis. I have been allowed to follow my ideas and at the same time always got feedback and help with different challenges. You have always been available to answer questions and have regular guidance meetings.

I would like to thank my co-supervisors Øyvind Kåre Kjerstad and Roger Skjetne for help with outlining the topic of my thesis.

I would like to thank Astrid H. Brodtkorp for sharing the models necessary to prepare and conduct testing in MCLab and Torgeir Wahl for help with all the practical problems during testing in MCLab.

I would like to thank Marthe Moengen for very good collaboration in MCLab and Tobias Torben for help with building up the simulation environment for the ferry and other problems related to the thesis.

I would also like to thank Andreas Sanne, Øyvind Blindheim, Embla Larsdotter Holten, and Tobias Torben for a very good office environment. It has been a very good place for both discussions related to the thesis and other topics, providing nice breaks when working long days.

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

MSC THESIS DESCRIPTION SHEET

Name of the candidate:	Thomas Johansen
Field of study:	Marine control engineering
Thesis title (Norwegian):	Autonom baneplanlegging for auto dokking av ferjer
Thesis title (English):	Autonomous path planning for auto docking of ferries

Background
A fully autonomous vessel requires characterization, prediction, control and monitoring of marine environmental and oceanographic parameters and ecosystems. At the same time, the autonomous vessel needs to ensure safe and successful marine structures and operations with increased autonomy during abnormal events or in hostile conditions. There are two main phases in autonomous navigation of ferries, i.e. transit/crossing and docking/berthing. Both operations need smart, robust, and fast control systems to operate safe in interaction with other vessels and obstacles. The dynamic nature of the environmental forces is also a challenge for the autonomous ferries, especially when navigating in tight spaces, such as harbors. This project will consider control systems for handling autonomous docking. More specifically, how to plan a path for safe and efficient navigation and control of the ferry in the docking phase.

Work description
1. Perform a background and literature review to provide information and relevant references on:
   - Autonomous surface ships
   - Path planning methods
   - Sensor integration and modeling
   - Model Predictive Control
   - Nonlinear Optimization
   - Optimal Path Planning

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

2. Adjust the control system so it can function with the path planner

3. Develop an algorithm for autonomous path planning in auto docking of ferries. This should, if possible, consider the following:
   - Avoiding stationary objects in the path
   - Account for environmental forces affecting the vessel and the surroundings around the vessel
   - Minimize time used on path
   - Optimizing control margins for the vessel
   - Optimizing energy efficiency

4. Test the algorithm in simulations on both the ferry model, the model of R/V Gunnerus, and MCSim

5. Test the algorithm in model testing in MCLab

Specifications
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. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts., and it is not expected to be longer than 60-80 A4 pages, from introduction to conclusion, unless otherwise agreed upon. It 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 with objective, background, and scope and delimitations, main body with problem formulations, 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, with the printed copy signed by the candidate, and otherwise according to NTNU procedures. The final revised version of this thesis description must be included. Computer code, pictures, videos, data series, and a PDF version of the report shall be enclosed electronically with all submitted versions.

Start date:	15. January 2019	**Due date:**	11. June 2019

Supervisor:	Dong T. Nguyen
Co-advisor(s):	Øivind Kåre Kjerstad, Roger Skjetne

Trondheim, 6. June 2019

[illegible]

Dong Trong Nguyen
Supervisor

Abstract

The main task in this thesis has been to identify and develop an algorithm for path planning in auto docking of autonomous ferries. The proposed planner treats the planning problem as a nonlinear optimization problem, and uses CasADi in Matlab to solve this optimization problem. The planner models the vessel using a 3DOF kinematic model, with five or six states and two or three inputs, depending on the operation. The planner solves the optimization and produces a reference trajectory for the control system to follow. To test the planner, three different scenarios with varying obstacles, initial, and final conditions are specified. The planner manages to solve the problem and find the optimal solution within the given constraints, but it depends strongly on how the optimization is set up. The trajectories are tested in both simulators and experimental tests, to verify that they can actually be followed. The results here show that the trajectories are possible to follow, but they must be adjusted to the control system. The thesis also includes a literature study about autonomous surface vessels, sensor integration and modeling, path planning, model predictive control, and nonlinear optimization. The results from this study are the background for choosing nonlinear optimization to solve the path planning problem and what information is included in the planner at this stage.

The main takeaway from this thesis is that nonlinear optimization is a good method for path planning for autonomous ferries, both in the docking and transit phase. However, it is still necessary to continue developing the planner to make it more efficient and include more information about both the vessel and disturbances that can affect the vessel's movement.

Sammendrag

Hovedproblemet i denne oppgaven har vært å identifisere og utvikle en algoritme for baneplanlegging for auto dokking av autonome ferjer. Den foreslåtte planleggeren løser problemet som er ulineært optimeringsproblem og bruker CasADi i Matlab for å løse dette. Planleggeren modellerer fartøyet med en 3DOF kinematisk modell med fem eller seks tilstander og to eller tre kontrollinput, avhengig av operasjonen. Planleggeren løser optimeringsproblemet og gir ut en tidsreferanse for banen som kontrollsystemet skal følge. For å teste planleggeren brukes det tre forskjellige scenario med ulike hindringer, start, og slutttilstander. Planleggeren klarer å løse problemet og finne den optimale løsningen innenfor de gitte grensene, men resultatet avhenger veldig av oppsettet på optimeringsproblemet. Banereferansene testes i både simuleringer og modellforsøk for å sjekke at de faktisk er mulig å følge. Resultatene viser at det går bra, men de må justeres for å passe kontroll-sytemene. Oppgaven inneholder også et litteratur studie om autonome overflatefartøy, sensor integrering og modellering, baneplanlegging, modellprediktiv regulering, og ulineær optimering. Resultatene fra dette er bakgrunnen for valget av ulineær optimering for å løse baneplanleggings problemmet og hvilken informasjon som skal inkluderes i planleggeren.

Hovedresultatet fra oppgaven er at ulineær optimering er en god metode for baneplanlegging for autonome ferjer, både for dokking og overfart. Men, det er fortsatt nødvendig å utvikle planleggeren videre for å gjøre den mer effektiv og inkludere mer informasjon om både fartøyet og forstyrrelser som kan påvirke fartøyets bevegelser.

Contents

List of Figures

List of Tables

Nomenclature

A	X Coordinate of Circle Origin. 21
B	Y Coordinate of Circle Origin. 21
K_{d_ψ}	Derivative gain for heading controller. 29
K_{i_U}	Integral gain for velocity controller. 30
K_{i_χ}	Integral gain for relative course controller. 29
K_{p_U}	Proportional gain for velocity controller. 30
K_{p_χ}	Proportional gain for relative course controller. 29
K_{p_ψ}	Proportional gain for heading controller. 29
R	Circle Radius. 21
U	Absolute Velocity. 18, 19, 30
U_d	Reference velocity. 30
χ_p	Path-tangential angle. 29
ω	Angular Velocity. 18, 19, 21, 26, 27
ψ	Heading Angle. 18, 19, 21, 29
τ_N	Force in yaw direction. 29
a	Acceleration. 18, 19, 26, 27
a_u	Surge Acceleration. 21

a_v	Sway Acceleration. 21
e	Cross-track error. 29
r	Angular Acceleration. 19, 21
u	Surge Velocity. 21
v	Sway Velocity. 21
w_{obs}	Cost on obstacle. 27
x	Global X Position. 18, 19, 21, 29
x	X Coordinate in (x,y)-plane. 21, 22
x_d	Reference X Position. 29
x_i	X coordinate of point i. 22
y	Global Y Position. 18, 19, 21, 29
y	Y Coordinate in (x,y)-plane. 21, 22
y_d	Reference Y Position. 29
y_i	Y coordinate of point i. 22

Abbreviations

ACA	The Ariadne's Clew Algorithm. 9
ACC	Adaptive Cruise Control. 4
AUV	Autonomous Underwater Vehicle. 3, 5
CS3	Cybership III. 32, 44, 54, 58
DARPA	Defense Advanced Research Projects Agency. 2
DOF	Degree of Freedom. 27, 57
DP	Dynamic Positioning. 2, 13, 14, 21, 23, 29, 30, 33, 34, 38, 39, 42, 44, 46, 51, 53, 54, 56
EQP	Equality-constrained problem. 11
EST	Expansive-Spaces Tree. 9
GPS	Global Positioning System. 3, 4
IQP	Inequality-constrained problem. 11
LIDAR	Light Detection and Ranging. 2, 4
MCLab	Marine Cybernetics Laboratory. 33, 49, 50
MIT	Massachusetts Institute of Technology. 3

MPC	Model Predictive Control. 12, 14, 28
NLP	Nonlinear Programming. 10, 11, 26
NTNU	Norwegian University of Science and Technology. 1, 2
PID	Proportional–Integral–Derivative. 29, 33, 34, 42, 43
PRM	Probabilistic Road-maps. 9
QP	Quadratic Programming. 10
RGB	RGB color model. 24
RPi	Raspberry Pi Computer. 49
RPP	Randomized Path Planner. 9
RRT	Rapidly-exploring Random Tree. 9
RTK	Real Time Kinematics. 57
SOG	Speed Over Ground. 4
SQP	Sequential Quadratic Programming. 10, 11
STW	Speed Through Water. 4
USV	Un-manned Surface Vehicle. 2, 3

Chapter 1

Introduction

1.1 Background

The subject of autonomous vehicles is one of the most relevant topics in control engineering today. Autonomous cars, drones, underwater vehicles, and surface ships are topics of great interest for both industry and research institutes. One of the most interesting topics of autonomous vehicles is autonomous ferries. Ferries operate on specific routes and they follow the same route for each crossing. This type of repetitive operations is well suited for autonomous vessels. Multiple research and commercial projects have been done on the topic of autonomous ferries of some types. Examples of this are the ferry Miliampere from Norwegian University of Science and Technology (NTNU) (*Teknisk Ukeblad* (2018*a*)), Rolls-Royce Marine (*Teknisk Ukeblad* (2018*b*)), and Wärtsila (*Teknisk ukeblad* (2018*c*)). Rolls-Royce and Wärtsila have tested their systems on operational ferries, and Miliampere has been tested with different levels of autonomy.

One of the key elements in autonomous navigation, is online path planning. For the vessel to navigate safely and economically in an unknown and changing environment, the vessel must have the capability to alter the path in changing conditions. This thesis will discuss and investigate how an autonomous surface vessel can plan a safe path around obstacles, in changing conditions, as it approaches the dock. As the vessel operates near the dock, the margins for safe operation become smaller since obstacles are much closer than in open waters. As a consequence of this, the path planner should put more weight on safety, and maximizing control margins as it approaches the dock. For autonomous ferries with passengers on-board, this safety aspect is even more important.

1.2 Previous Work

Much work has been done in the field of autonomous surface ships, and ferries are one vessel type that many companies and research institutes want to make autonomous. This was investigated in Johansen (2018) as an introduction to this thesis. This project looked into different sensors that could be used on autonomous ferries and how to use the information from these. No conclusions were made in terms of sensors and which sensors should be used, but the sensors needed for autonomous navigation of ferries exist. It briefly investigated some methods that could be used for the path planning problem and different aspects connected with ferry operations. The main product from this project was a simulation environment for testing different path planning methods. This includes a ferry model and a model of NTNU's research vessel R/V Gunnerus with a simple Dynamic Positioning (DP) control system for each vessel.

The topic of autonomous path planning has been researched and tested on multiple platforms, such as cars (Thrun et al. (2006), Dolgov et al. (2010)), drones (Bry & Roy (2011)), surface vessels (Spange (2016)), and underwater vehicles (Hernández et al. (2015)). The two cars in Dolgov et al. (2010) and Thrun et al. (2006) were made to compete in two editions of the DARPA Challenge. In Thrun et al. (2006), they developed an autonomous car that could navigate and operate in unknown off-road terrain. All information they had about the course before start was a set of way-points given with longitude and latitude coordinates, the width of the corridor they should follow between way-points, and speed limits for each part of the course. The car had to get all other necessary information while following the route. The car in Dolgov et al. (2010) was designed to operate in urban environments. More specifically, the challenge was to autonomously navigate in a parking lot. In Bry & Roy (2011), the goal was to make a fixed wing drone capable of navigating in unknown environments that were impossible for humans to control the drone in. This included a parking garage with such limited space that the drone had only a few centimeters clearance on both wings during sharp turns. Spange (2016) tested an Unmanned Surface Vehicle (USV) equipped with a Light Detection and Ranging (LIDAR) and proximity sensors for autonomous navigation in harbor environments. Hernàndez (2017) discussed and tested multiple approaches for autonomous underwater navigation.

1.3 Literature Review

1.3.1 Autonomous Surface Ships

Autonomous Surface Ships, or USVs, have been around for approximately 25 years (Manley (2008)). DP-vessels have been around longer, but the early DP-system was more assisting than controlling the vessel. Modern DP-vessels however operate much more autonomously and should therefore be included in this description. USVs are classified in many different ways, but Schiaretti et al. (2017) provides a nice and easy framework for how they can be classified. To decide the level of autonomy, the system is divided in four parts. Decision making subsystem, actions taking subsystem, exceptions handling subsystem, and cooperative subsystem. Each of these subsystems have a scale based on how much human control is needed. The level of autonomy for the whole system is based on the level for each subsystem. The full scale and description of this can be seen in

Schiaretti et al. (2017).

The first USV developed for autonomous operation was developed at Massachusetts Institute of Technology (MIT) and named ARTEMIS (VANECK et al. (1996)). This was a small monohull vessel with Global Positioning System (GPS), compass, and a simple heading controller. It was used for bathymetry mapping on the Charles River. Its small size meant limited endurance and seakeeping abilities. At MIT they therefore decided to make a new USV with better seakeeping abilities and endurance. They first used a kayak hull, before they decided to use a multihull platform. This new USV was named ACES (Manley (1997)). This performed significantly better than ARTEMIS, and was later upgraded with new technology and renamed AutoCat (Manley et al. (2000)). As computers, sensors, and actuators have become smaller, cheaper, and better, the evolution of USVs has continued. Today both research institutes and commercial companies are testing and using different types of USVs in different operations.

Today, most USVs use electrical motors for propulsion, but this means a limited range and operation time. Both wave and wind powered USVs have been developed and they provide better range and endurance. One example of this is the Saildrone (Meinig et al. (2015)). Saildrone is a 19 ft wind powered USV that can be used for long research projects such as ocean monitoring in the Bering Sea (Cokelet et al. (2015)). Two saildrones were deployed in the Bering Sea and operated for 97 days. The drones were controlled by giving way-points and a corridor width that the drone should stay inside between two way points. It used a wing sail for thrust and solar panels powering two actuators and all sensors onboard the USV.

Another example is CS Saucer (Spange (2016) and Idland (2015)). This USV is a model-scaled vessel used for testing autonomous navigation and docking. The vessel is a buoy with three thrusters and sensors for mapping the area and finding obstacles the USV must avoid. This USV is only for use in indoor basins but it has a high level of autonomy since it can handle unknown elements in the route without human intervention.

Future USVs operate at a higher level of autonomy with less and less need for human intervention and an increasing level of collaboration between both other USVs, Autonomous Underwater Vehicles (AUVs), and AUVs. One example of this is the use of USVs as position references for AUVs, or launching AUVs from USVs, to collect more data than operating only one of these(Zhu & Wen (2019)).

1.3.2 Sensor Integration and Modeling

USVs can be equipped with many different sensors for providing the necessary information. Relevant sensors for autonomous ferries were investigated in Johansen (2018). These sensors and the information must then be combined and integrated in the control system in such a way that the information can be used, but it must also be sorted so it only uses the necessary information. To test these systems, there is also a need for accurate modeling of these sensors. Since the main goal of this thesis is developing a path planner, the main focus in this section will be how to integrate sensor data in the path planner. The sensors relevant in this thesis are GPS, wind sensors, sensors for measuring waves, water speed sensors, and some type of distance measuring that can detect obstacles and calculate the distance to these.

GPS was included in the simulation environment with a simple method for adding measuring noise as a part of Johansen (2018). This will therefore not be investigated further. Wind sensors, or anemometers, are simple sensors that measure the speed of the wind at the place they are mounted. These sensors measure the direction and speed of the wind, but this will not account for the vessel speed if mounted on the vessel. Pedersen et al. (1999) states that cup anemometers are the best sensors for measuring wind speed and direction. The measurements from these can be treated as perfect measurements.

Wave parameters such as wave height and wave period can be measured by buoys or radars. In this thesis, buoys are considered to be the best alternative since they can be deployed around the dock more efficient than radars. One example of such buoys are from Datawell BV(*Datawell BV* (2019)). They operate with 0.5% accuracy on the wave height measurements, and 0.5° on the wave direction measurements. Simulating wave buoys with these values for the accuracy then seems natural. The same buoys can in many cases measure ocean currents. Datawell BV operates with $\pm 2\frac{cm}{s}$ as the accuracy for speed, and 0.5° on the direction measurements. Current speed can also be measured as the difference between Speed Through Water (STW) and Speed Over Ground (SOG). SOG is measured by the GPS and STW can be measured by sensors on the ship hull. Since the speed measurements of the current are supposed used in the path planner as information about the docking area, measuring current using buoys positioned near the dock is chosen as the most useful approach for auto docking.

Johansen (2018) also looked into different ranging sensors that could be used for pinpointing the vessel position relative to the dock. LIDAR, radar, and different types of sonars were proposed used for this. The main difference between these is the accuracy and range, but most systems operate in a similar way with similar performance and measurements. These sensors can be modeled as a sensor sending a signal when the vessel is closer than 100 m to an obstacle. This value is based on a mid range radar from Bosch, normally used in Adaptive Cruise Control (ACC) (*Bosch* (2019)).

By including a model for measuring wind, current, waves, and some distance measuring sensors, the system should contain the necessary sensors to simulate an autonomous ferry. This model can use the perfect measurements affecting the system and add noise corresponding to each sensor. The challenge with this is to then combine the data from these sensors in a map that data can be extracted from. Challenges connected to this are different update cycles for sensors, and how to store the data in one map that can be accessed and updated continuously. One method for this, is dividing a map in different cells and assigning data to each of these. Using loops with different update cycles, information containing data about for example waves can then be added to these cells and accessed in a path planner that uses this information to find a safe and efficient path. Since the docking area of a ferry is the same for each crossing, dividing a map in nodes can be done offline and then the map can just be updated when new data is available. The information in this map can then be added in the planner. An example of this is adding current with a function taking the vessel position as input.

1.3.3 Path Planning Methods

Johansen (2018) identified two possible path planning methods, but also made it clear that other methods exist and might be more efficient. It was therefore necessary to do a more in-depth review of different path planning methods. Hernàndez (2017) investigated methods for path planning for AUVs and also presents a good overview of different techniques that can be used. Hernàndez (2017) describes five main categories for path planning methods. These five are bug-based, search-based, potential fields, road-maps, and sampling-based algorithms. Each of these five, and optimization-based methods, will be described in the following sections.

Bug-based Path Planning

Bug-based path planning is one of the first approaches used for reactive path planning. A robot is moving in a 2D plane with a known goal. The robot has two possible ways of maneuvering: straight towards the goal, and along an object boundary. The main difference between each bug-based approach is how the robot moves around objects. Bug1 (Lumelsky & Stepanov (1987)) is the simplest method where the robot moves straight towards the goal, until it encounters an obstacle at a hit point. It will then move around the whole obstacle until it reaches the hit point again. During this maneuver, the robot will look for the point closest to the goal and mark this as a leave point. After one lap around the obstacle, the robot moves to the leave point and then follows a straight line to the goal position again. Bug2 (Lumelsky & Stepanov (1987)) starts by defining a straight line from start to goal, called the m-line. The robot starts to move along this line, but when it encounters an obstacle, it will move around the obstacle until it crosses the m-line. When this line is reached, the robot follows it to the next obstacle, or to the goal. The last bug method is called the tangent bug method (Kamon et al. (1996)). This method uses a range sensor that can sense obstacles. This enables the robot to find a more optimal path since it can get information about the obstacle prior to actually hitting the object boundary. The robot can then find a more optimal path between obstacles since it can plan in advance.

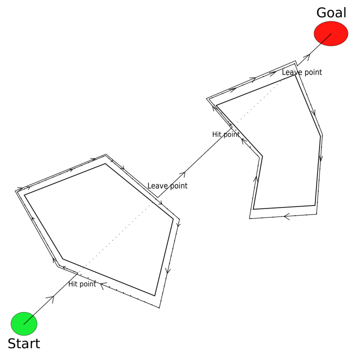


Figure 1.1: Bug1

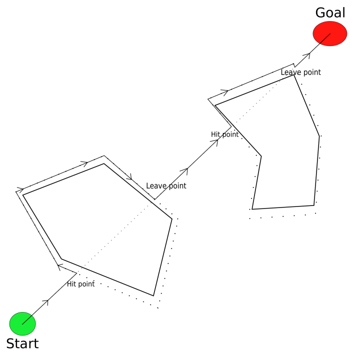


Figure 1.2: Bug2

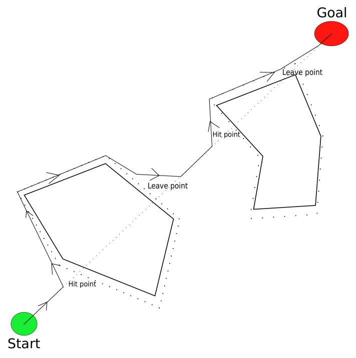


Figure 1.3: Tangent Bug

Search-based Path Planning

This method for path planning uses graph search methods for finding a path through a discrete version of the configuration space. To do this, the configuration space is divided in a square grid. The robot can then move from one square to a connecting square as long as the new square is free of obstacles. To use this method in path planning involves two problems: finding the right resolution for the grid, and finding a path from start to goal in this grid.

Choosing the resolution for the grid will affect how much computation is needed for finding a path, but it will also affect how efficient the path is. A very coarse grid can for example disregard routes between obstacles that the robot can fit in, because the grid is so coarse that all squares have obstacles in them.

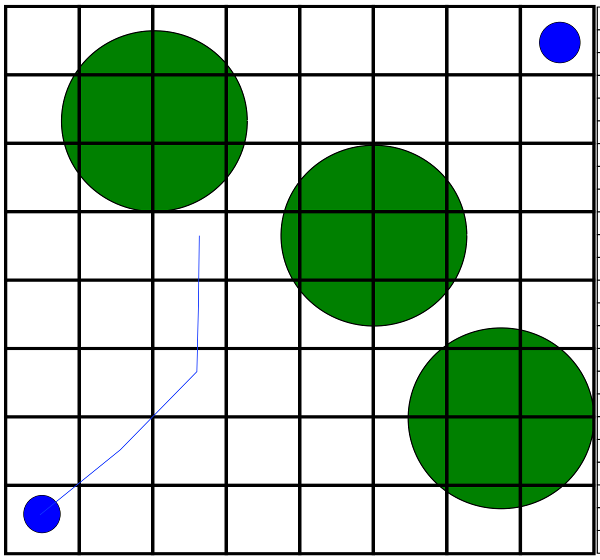


Figure 1.4: Coarse Grid

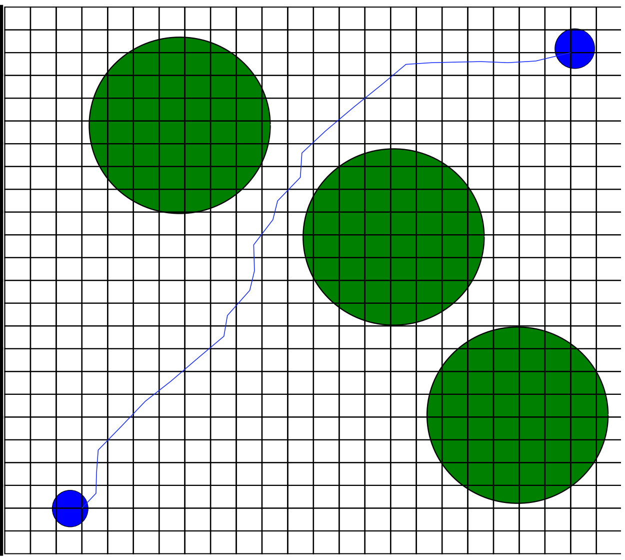


Figure 1.5: Fine Grid

Planning the path is usually done with a search algorithm. One of the simplest algorithms that can be used for this is Dijkstra's Algorithm (Dijkstra (1959)). The method uses a weighted graph to find the optimal path. This is done by calculating the cost of traversing from one square to each of its neighbor squares. The square with the least cost is then chosen as the current square and then the process is repeated until the goal is visited. The shortest path is then found using backtracking along the path with the least cost. This method will find a path, but it means visiting almost every square to determine the shortest path. An alternative algorithm is Astar. This algorithm uses two functions for determining the optimal path. Dijkstra evaluates the cost of moving from the current square to a neighbor, but Astar also takes into account a heuristic cost from a square to the goal square. A simple way of describing the difference is that Dijkstra chooses the closest square from its current position whereas Astar (Hart et al. (1968)) uses both the distance from the initial position and the distance from goal to each square, to decide which square to visit. Both these algorithms are used for planning in a static environment where the path can be planned offline. To account for dynamic environments, Dstar can be used (Stentz (1994) and Stentz (1995)). This method can recalculate the cost of traversing different nodes as new information becomes available. This enables the method to be used for environments where there might be unknown obstacles.

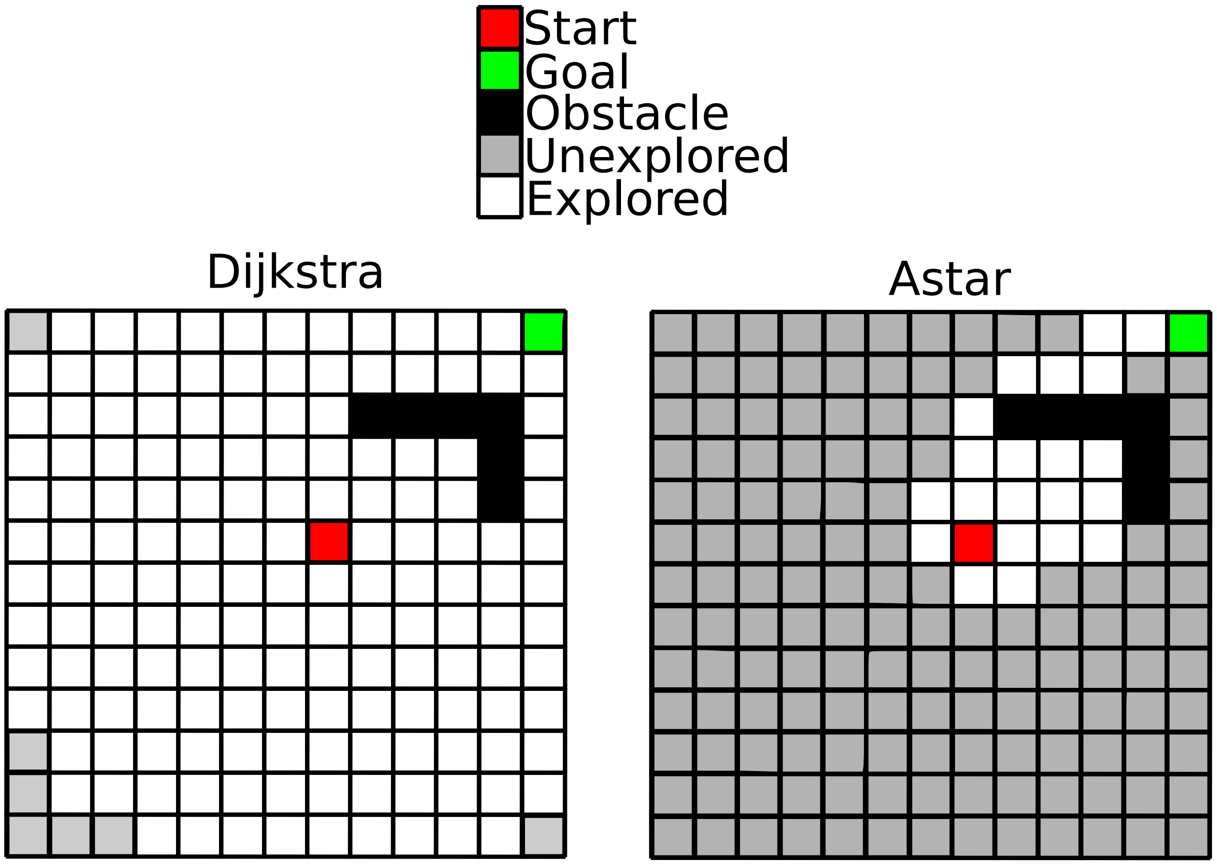


Figure 1.6: Dijkstra Vs Astar

Potential Fields

An alternative approach to search-based algorithms is based on using potential functions (Khatib (1985) and Khatib (1986)). This method can be used without having to lay a grid over the map and therefore saves both time and computer power. The idea behind potential fields in path planning is saying that the goal has a pull force and obstacles push the robot away. This solution is possible to use for both 2D and 3D problems, but it has the problem that it might not converge. This can happen if the robot gets stuck in a local minimum.

Road-maps

Road-map methods are another approach to path planning. Path planners based on the previous mentioned methods, such as bug-based and search-based are well suited for planning a single path from start to finish. Other times it is necessary to find multiple possible routes between the same points. For these applications, maps containing all possible routes are very useful. One type of such maps is called visibility graphs. These maps contain lines from start and goal to the corners on objects, and between corners. This map can then be searched using a graph search method to find a set of lines linking start and finish without going through obstacles. Another method that can be used to divide the configuration space in lines is generalized Voronoi diagrams (Aurenhammer (1991)). Voronoi diagrams draw lines equally spaced between objects, and these lines can then be searched using graph search methods, the same way as visibility graphs, to find a path from start to finish.

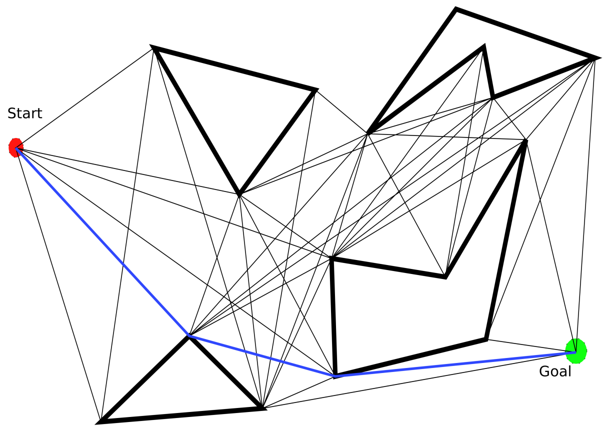


Figure 1.7: Visibility Graph

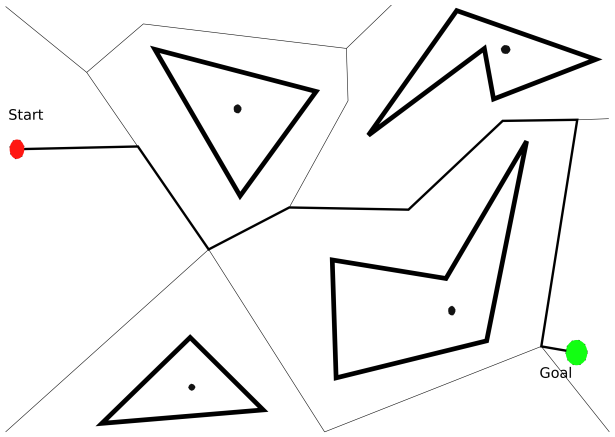


Figure 1.8: Generalized Voronoi Diagrams

Sampling Based Algorithms

The methods mentioned above require explicit representations of the collision free part of the configuration space. For some applications, there exists no explicit representation of a collision free space, or finding this means doing heavy computations and processing. For these applications, an effective alternative is sampling-based algorithms. These methods start by randomly checking different robot configurations and checks if they are collision free. Collision free configurations are then connected to make collision free paths. This way of dividing the method in two gives the possibility to generalize the path planning problem by separating the search algorithm from the geometric representation of the environment. The method does however also have some limitations. Contrary to the previous methods, this method cannot notify if a solution exists. This means that it cannot be guaranteed that a solution exists for the problem. But, since most of these methods use a random sampling, which is dense with probability one, if a solution exists it will probably be found. The problem can be the time used to find this solution.

These methods can, as the previous methods, be divided in two. Some methods are used to find one path for one specific problem, and others are used to make road-maps of all possible paths in a configuration space. One example of the latter is Probabilistic Road-maps (PRM) (Kavraki et al. (1996)). PRM is divided in two phases: one learning phase where a probabilistic road-map is constructed, and a query phase where a given start and goal position are connected to two single nodes in the road-map. The map can then be searched to find a path between these nodes. For solving the other type of path planning problems, finding one path to follow in one specific case, there exists multiple algorithms. These work by either expanding a tree of possible routes from start, or by expanding one from the start and one from the goal position. Examples of such methods are Randomized Path Planner (RPP), The Ariadne's Clew Algorithm (ACA), Expansive-Spaces Tree (EST), and Rapidly-exploring Random Tree (RRT). RPP (Barraquand & Latombe (1990)) is similar to the potential field approach described earlier, but it uses a random walk process to escape local minimums. ACA (Bessiere et al. (1993)) uses a search algorithm to see if a target can be reached easily from a start point. If no simple path can be found, it uses an explore algorithm that places different landmarks as far away from each other as possible, that can be reached from the start point. Then these two algorithms are repeated until the goal is reached. EST (Hsu et al. (1997)) builds one tree from the start position and one from the goal position. These are built by prioritizing less explored regions. When the two trees meet, a path from start to finish is found. RRT (Lavalle (1998), LaValle & Kuffner (1999), and Kuffner & LaValle (2000)) expands a random tree by prioritizing unexplored regions, in a similar way as EST, but it can also handle dynamic constraints. This means the planner will not only check if the new configuration is collision free, but also if the robot is capable of moving to this new position. There exist versions of RRT-planners that expand from only the start position, or from both the start and finish position.

Optimization Methods for Path Planning

The methods for path planning presented above can all be described as way-point methods. They use different methods to identify way-points that, if linked, will make a path from start to finish. Another approach is using optimization to solve the problem. The system can be described as the following

$$\dot{x} = f(x(t), u(t), t), \quad t_0 < t < t_f \tag{1.1}$$

$$x(t_o) = x_0 \tag{1.2}$$

The objective can then be to minimize the time necessary to move from start to finish, the total energy used to move the system, or other terms. Bitar et al. (2018) and Lekkas et al. (2016) uses pseudospectral optimization to plan an optimal path when the vessel is affected by an ocean current. Bitar et al. (2018) optimized the energy used and showed significant improvement on this while using the same time from start to finish. Lekkas et al. (2016) used the same method to account for an unknown ocean current while updating the path while moving. The vessel was then able to avoid obstacles with traditional line of sight guidance without integral term.

1.3.4 Nonlinear Programming Problem

Nonlinear Programming (NLP) is a class of optimization problems where, at least, some of the objective functions or constraints are nonlinear (Nocedal & Wright (2006)). These problems can in general be formulated as the following

$$\min_{x \in R^n} \quad f(x) \quad \textit{subject to} \quad \begin{cases} c_i(x) = 0, & i \in \mathcal{E} \\ c_i(x) \geq 0, & i \in \mathcal{I} \end{cases} \tag{1.3}$$

These problems can be solved using different solving techniques. Examples are Quadratic Programming (QP) problems, penalty and augmented Lagrangian methods, Sequential Quadratic Programming (SQP) methods, and interior-point methods. Each method has some fundamental differences, but all methods are in some way iterative. This means that they must search through different possible solutions.

QP problems have a quadratic objective function and linear constraint functions. In general they can be stated as the following

$$\min_{x} \quad q(x) = \frac{1}{2} x^\top G x + x^\top c \quad \textit{subject to} \quad \begin{cases} a_i x = b_i, & i \in \mathcal{E} \\ a_i x \geq b_i, & i \in \mathcal{I} \end{cases} \tag{1.4}$$

These problems can always be either solved, or shown to be infeasible, in a finite amount of computations, but how many computations will depend strongly on the objective function and the number of constraints.

Penalty and augmented Lagrangian methods use penalty functions to add the constraints to the objective function. There are multiple possible methods for adding these penalty functions to the objective, but the most common are quadratic- or non-smooth penalty functions . Lagrangian methods are similar to the quadratic penalty functions, but they add a Lagrangian multiplier to reduce the possibility of ill conditioning in the cost function. Examples of the three different methods for the problem in equation (1.3) are shown in (1.5), (1.6), and (1.7) respectively.

$$Q(x;\mu) \overset{def}{=} f(x) + \frac{\mu}{2}\sum_{i\in\mathcal{E}} c_i^2(x) + \frac{\mu}{2}\sum_{i\in\mathcal{I}}([c_i(x)]^-)^2; \quad [c_i(x)]^- = max(-c_i(x),0) \tag{1.5}$$

$$\phi_1(x;\mu) \overset{def}{=} f(x) + \mu\sum_{i\in\mathcal{E}} |c_i(x)| + \mu\sum_{i\in\mathcal{I}}[c_i(x)]^-; \quad [c_i(x)]^- = max(-c_i(x),0) \tag{1.6}$$

$$\mathcal{L}_A(x,\lambda;\mu) \overset{def}{=} f(x) - \sum_{i\in\mathcal{E}} \lambda_i c_i(x) + \frac{\mu}{2}\sum_{i\in\mathcal{E}} c_i^2(x) \tag{1.7}$$

μ is the penalty parameter deciding how much violating the constraint should be penalized. λ is the Lagrange multipliers.

SQP generates steps for solving the optimization by solving quadratic sub-problems. This method is especially effective for solving problems with nonlinear constraints. Nocedal & Wright (2006) presents two types of SQP called active-set methods, Inequality-constrained problems (IQPs) and Equality-constrained problems (EQPs). IQP solves a general inequality-constrained quadratic program at each iteration that produces a step and estimates the optimal active-set. EQP first estimate the optimal active-set, and then solve an equality-constrained quadratic program to find a step. For more information about the different algorithms for both IQP and EQP, the reader is referred to chapter 18. in Nocedal & Wright (2006).

The last method for solving NLP-problems that will be mentioned is interior-point methods. These methods use different search methods to search for optimal solutions. The special property of interior-point methods is that all iterations should be inside the region of feasible solutions. For more information about interior-point methods, the reader is referred to chapter 19. in Nocedal & Wright (2006), but one of these will be presented as a part of section 1.3.7 describing IPOPT.

1.3.5 Model Predictive Control

Model Predictive Control (MPC) is formulated as the following by Mayne et al. (2000)

> Model Predictive Control is a form of control in which the current control action is obtained by solving, at each sampling instant, a finite horizon open-loop optimal control problem, using the current state of the model as the initial state; the optimization yields an optimal control sequence and the first control in this sequence is applied to the plant.

The idea of using a model to estimate the system and calculate the optimal control action was first used in industrial processes in the late 1970s (Richalet et al. (1978)). The idea behind MPC was to utilize the development in computer technology to give better control systems. This was especially important for industrial processes. Other applications had better control systems, such as aerospace and spacecrafts, but the solutions used in these systems did not work for industrial processes. Industrial processes have systems with many variables that often change more often than they are measured. How to measure the performance and reliability was also different than for other systems. The idea was therefore to model the system very accurately and estimate the behavior of the system between each sample. The modeled states could then be used to compute the optimal control action between each sample. Constraints on different variables and control inputs can also be handled in these controllers. In cases where these constraints hinder the controller from achieving the wanted results, this can be used to avoid building up errors in the controller. This combination of accurate modeling of states between measurements, and capability of including the constraints in the controller can give very good results. However, the performance will depend on the accuracy of the model. A good model will often give good performance, but a poor model will often give worse performance than for example feedback control.

1.3.6 CasADi

CasADi (Andersson et al. (2018)) is an open-source software for numerical optimization. CasADi is written as stand alone C++ software, but it can also be used with Python or Matlab with only minor differences in the coding. CasADi offers a low-level language for solving optimization problems. Central in CasADi is a symbolic framework that can be used to declare variables for use in solving different problems. These symbolic variables can then be used in different mathematical operations. CasADi is especially suitable for solving optimization problems with differential equations.

CasADi has some tools included, such as integrators and nonlinear solvers, but these can also easily be implemented by the users. This enables the user to develop tools specifically for solving the specific problem. To read more about CasADi, the reader is referred to Andersson et al. (2018).

1.3.7 IPOPT

The standard nonlinear solver used in CasADi for solving the optimization is IPOPT (Wächter (2009)). IPOPT is a primal-dual interior-point algorithm. The method aims to solve problems of the following type:

$$\min_{x\in\mathbb{R}^n} f(x) \tag{1.8}$$

$$s.t \quad c(x) = 0 \tag{1.9}$$

$$x \geq 0 \tag{1.10}$$

The algorithm solves the problem by solving a sequence of the following problem,

$$\min_{x\in\mathbb{R}^n} \varphi_\mu(x) \equiv f(x) - \mu \sum_{i=1}^{n} \ln\left(x^{(i)}\right) \tag{1.11}$$

$$s.t \quad c(x) = 0 \tag{1.12}$$

while μ converges to zero. This method will then search the space where the constraints are satisfied to try and find the optimal minimum of the problem $f(x)$. The barrier objective function, $\varphi_\mu(x)$, will approach infinity along the edge of the feasible region keeping the solution inside the constraints. By computing the minimum of each step and using this as the starting point of the next iteration, while decreasing the barrier term μ, the solution will approach the optimum.

1.4 Improvements from Previous Project Work

This thesis is a continuation of Johansen (2018), where a DP-control system was implemented on a ferry simulator and a simulator based on R/V Gunnerus. This project concluded that it was necessary to improve this control system further to improve performance and function better with a path planner.

The most important problem with the control system before was the amount of noise present after the observer. The plan here was to improve the tuning, but after more research on different sensors, the conclusion was that the noise level was too high before. Lowering this to more realistic values improved the performance significantly. The task of altering the guidance module was, after some research into different solutions, fixed by including this functionality in the planner. The planner would then not provide waypoints to the controller, but complete trajectories. This omitted the need for a guidance module since the reference trajectory could be used directly in the controllers.

1.5 Objective and Scope

The objective of this thesis is to continue the work from Johansen (2018) on a path planning algorithm for auto docking of autonomous ferries. The main objective is to identify and develop an actual path planner for auto docking as well as how the planner should be integrated in the control system. The path planner should consider the following criteria, in prioritized order:

1. Safety
2. Maximizing control margins
3. Minimize the time used
4. Minimize energy used

The thesis will also provide background on autonomous surface ships, different methods for path planning, sensors for autonomous ships, MPC, and nonlinear optimization.

1.6 Thesis Main Contributions

The main contribution from this thesis is an algorithm for optimal path planning using open source software. The main software for this path planner is CasADi, which is an open source optimization tool, compatible with C++, Python, and Matlab with only minor changes needed in the code to switch languages. The most similar work the author has found is Bitar et al. (2018) and Lekkas et al. (2016), which used DIDO for optimal path planning. DIDO is Matlab specific, licensed software, and not available or compatible with open source and free software. The proposed planner produces a reference trajectory that heading/speed- or DP-controllers can follow. Further, the thesis outlines how the planner can be used for offline path planning for autonomous ferries.

As a part of this thesis, tools for extracting obstacle data from image files have also been developed. One code extracts circles and one extracts a point list of the corners of rectangular objects for use in obstacle avoidance.

1.7 Organization of Thesis

Chapter 1 introduces the thesis and presents background on Autonomous Surface Vessels, Sensor Integration, Path Planning methods, nonlinear optimization, MPC, CasADi, and IPOPT

Chapter 2 describes the planner. This chapter includes the model used in the planner, how to handle stationary obstacles, environmental disturbances, map processing, as well as the control system and how the planner can be integrated into this.

Chapter 3 describes the different simulations performed during testing and the results from these

Chapter 4 describes the experimental testing in MCLab. This chapter includes a description of the laboratory setup, what preparations were necessary before testing, what tests were conducted and the results from these.

Chapter 5 concludes the thesis. This chapter sums up the results from tests, both simulations and experimental testing. It also outlines further work necessary for the planner to be used in actual applications.

Chapter 2

Path Planning

The path planner is the main objective in this thesis. The goal was to find a method for planning a path an autonomous ferry could follow during docking. It was also a goal that this same planner could be used for other vessels without needing to change the whole planner. Multiple methods for path planning exist and a number of these were presented in section 1.3.3. Most of these methods are well suited for exploring unknown environments, but the goal for this planner was to plan a path in a well-known environment. It was also necessary that this path could be optimized for different parameters. For autonomous ferries, four main parameters were identified and chosen in the following order:

1. Safety
2. Maximize control margins
3. Time efficiency
4. Power efficiency

The most important part is safety and this will therefore be the main concern when planning the path. Maximizing the control margins is connected with safety, but it will be more connected with planning for different scenarios and always choosing the path that will maximize the ability to navigate safely if unknown things should happen. Time will be the next thing to optimize for, since a delay will have large consequences, both economical and in terms of customer satisfaction. These consequences will most likely be larger than the potential savings from power efficiency and this is therefore more important.

For other ships and operations, the priorities might change and the idea is therefore to find a method where different weights can be set to the different terms. This should make it possible to use the same method for multiple ships and operations without making massive changes. The path planner should therefore take in a map of the surroundings and the position the vessel should dock, information about the ship such as length and width, and output a reference trajectory and speed reference that the vessel should follow. Different approaches were investigated in both section 1.3.3 in this thesis and in Johansen (2018), but methods based on nonlinear optimization were deemed most efficient since this enables the use of different cost functions for different problems, and constraints on different parameters. This path planner was developed in an increased complexity manner to find the balance between computational power and the complexity of the model.

2.1 Basis

2.1.1 Modeling

The first version of a planner is based on a very simple model as shown in figure 2.1 with four states and two control inputs.

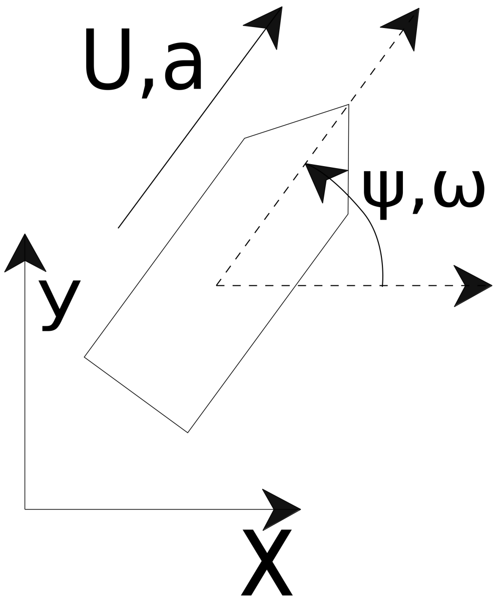


Figure 2.1: Ship Model One

This model has the following model equations:

$$\dot{x} = U \ cos(\psi) \tag{2.1}$$

$$\dot{y} = U \ sin(\psi) \tag{2.2}$$

$$\dot{U} = a \tag{2.3}$$

$$\dot{\psi} = \omega \tag{2.4}$$

where,
x - Global position in X-direction(State)
y - Global position in Y-direction(State)
U - Absolute velocity(State)
ψ - Heading angle(State)
a - Acceleration(Input)
ω - Yaw velocity(Input)

This model can then be used by setting a limit on maximum acceleration and yaw rate, and defining a good cost function to avoid obstacles and ensure convergence towards the goal states. The output from this is a reference trajectory in x- and y-position, heading reference, and a reference in absolute velocity and inputs. This simplified model is not ideal for use with a DP controller since it only gives an absolute velocity and not a reference in surge and sway. The velocity reference in yaw is here a control input and not guaranteed differentiable. This is a challenge since the references used in the controllers must be smooth signals.

The second version tested includes yaw velocity as a state and uses the angular acceleration as input. The model will then include one more state than the previous. This model can be seen in figure 2.2.

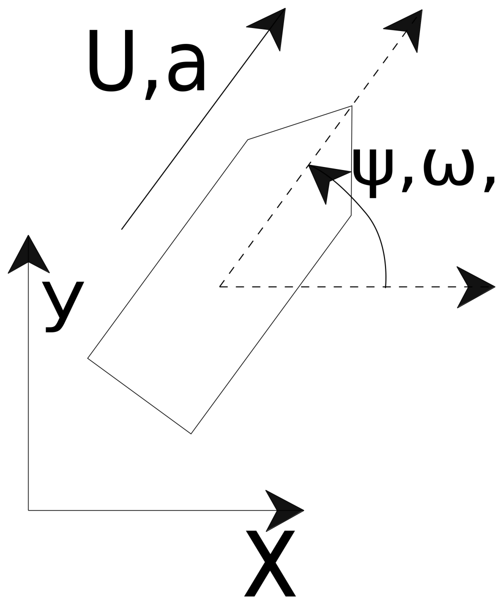


Figure 2.2: Ship Model Two

The model equations are then the following:

$$\dot{x} = U\ cos(\psi) \tag{2.5}$$

$$\dot{y} = U\ sin(\psi) \tag{2.6}$$

$$\dot{U} = a \tag{2.7}$$

$$\dot{\psi} = \omega \tag{2.8}$$

$$\dot{\omega} = r \tag{2.9}$$

where,
x - Global position in X-direction(State)
y - Global position in Y-direction(State)
U - Absolute velocity(State)
ψ - Heading angle(State)
ω - Yaw velocity(State)
a - Acceleration(Input)
r - Yaw acceleration(Input)

This gives better control over the angular speed and makes it much easier to adjust this limit so the control system on the vessel can handle it. This approach still has the challenge that it will only give a reference on absolute velocity, heading, and heading velocity. The references on y and x can be used, but since the planner only gives a reference in absolute velocity these can be challenging to follow.

Another approach is therefore to include six states in the equations and three inputs. The model will then be as shown in figure 2.3.

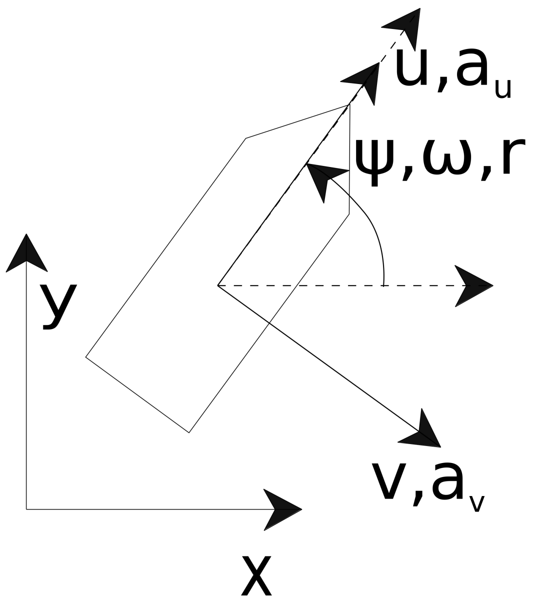


Figure 2.3: Ship Model Three

This model will then have the following equations:

$$\dot{x} = u\ cos(\psi) - v\ sin(\psi) \tag{2.10}$$

$$\dot{y} = u\ sin(\psi) + v\ cos(\psi) \tag{2.11}$$

$$\dot{\psi} = \omega \tag{2.12}$$

$$\dot{u} = a_u \tag{2.13}$$

$$\dot{v} = a_v \tag{2.14}$$

$$\dot{\omega} = r \tag{2.15}$$

where,
x - Global position in X-direction(State)
y - Global position in Y-direction(State)
ψ - Heading angle(State)
u - Surge velocity(State)
v - Sway velocity(State)
ω - Yaw velocity(State)
a_u - Surge acceleration(Input)
a_v - Sway acceleration(Input)
r - Yaw acceleration(Input)

This approach will give a more complicated model, but it will also give reference velocities in the same directions as a DP controller uses.

2.1.2 Obstacle Handling

One of the, if not the, most important aspects of the path planner is the ability to avoid obstacles. Fully autonomous vessels must have systems for handling both dynamic and static obstacles, but in this thesis only static obstacles will be considered. To include this capability in the optimization-based planner, the obstacles are included in the cost function used in the optimization or in constraints. To allow this, it is necessary to formulate mathematical functions for describing the obstacles and the distance from the vessel to these obstacles. This thesis will use two main methods for describing obstacles, circles and line segments. Circles can be described with equation (2.16) and line segments with equation (2.17).

$$R^2 = (x - A)^2 + (y - B)^2 \tag{2.16}$$

where,
R - Circle radius
x - X coordinate of point on the circle edge
y - Y coordinate of point on the circle edge
A - X coordinate of circle origin
B - Y coordinate of circle origin

$$y = (x - x_1)\frac{y_2 - y_1}{x_2 - x_1} + y_1 \tag{2.17}$$

where,

x - X coordinate of point on the line segment
y - Y coordinate of point on the line segment
x_i - X coordinate of point i, $i \in \{1,2\}$
y_i - Y coordinate of point i, $i \in \{1,2\}$

To use these equations in path planners, the important part is to describe the distance from the vessel to the obstacle. For obstacles described as circles, this is easy, since the distance is the same as the radius in equation (2.16). For line segments it is more difficult since you have two cases. One is when the vessel is perpendicular to the line, and one when it is not. To calculate the distance from the vessel to the line will be different for each of these cases. The function used to calculate the distance to obstacles is shown in equation (2.18) for circles and (2.19) for line segments.

$$close(x,y) = \sqrt{(x - x_{obstacle})^2 + (y - y_{obstacle})^2} \tag{2.18}$$

where;
$x_{obstacle}$ - X coordinate of obstacle origin
$y_{obstacle}$ - Y coordinate of obstacle origin

$$close(x,y) = d_{perp}(x,y)\ (test(x,y) = 1) + min(d_1, d_2)\ (test(x,y) \neq 1) \tag{2.19}$$

$$d_{perp}(x,y) = \frac{|(y_2 - y_1)x - (x_2 - x_1)y + x_2 y_1 - y_2 x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}} \tag{2.20}$$

$$d_i = \sqrt{(x - x_i)^2 + (y - y_i)};\quad i \in \{1,2\} \tag{2.21}$$

$$test(x,y) = \Big((p_1 - p_2)\cdot(p - p_2))((p_2 - p_1)\cdot(p - p_1)\Big) \geq 0 \tag{2.22}$$

$$p = \begin{bmatrix} x \\ y \end{bmatrix};\quad p_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix};\quad p_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \tag{2.23}$$

where,
x - X coordinate of the vessel position
y - Y coordinate of the vessel position
x_i - X coordinate of point i, $i \in \{1,2\}$
y_i - Y coordinate of point i, $i \in \{1,2\}$

The output from the function close(x,y) will then be the distance from the vessel to the object. This distance can then be used in both cost functions (2.2.1) and constraints to limit how close the vessel can be to different objects. Figure 2.4 shows how this will look with islands, and figure 2.5 for harbor environments. The green and black areas represent land and the red rectangles show the vessel with the path between them. The pink areas, dependent on how obstacles are treated, represent areas with a high cost for the vessel to maneuver in, or forbidden areas the vessel must avoid.

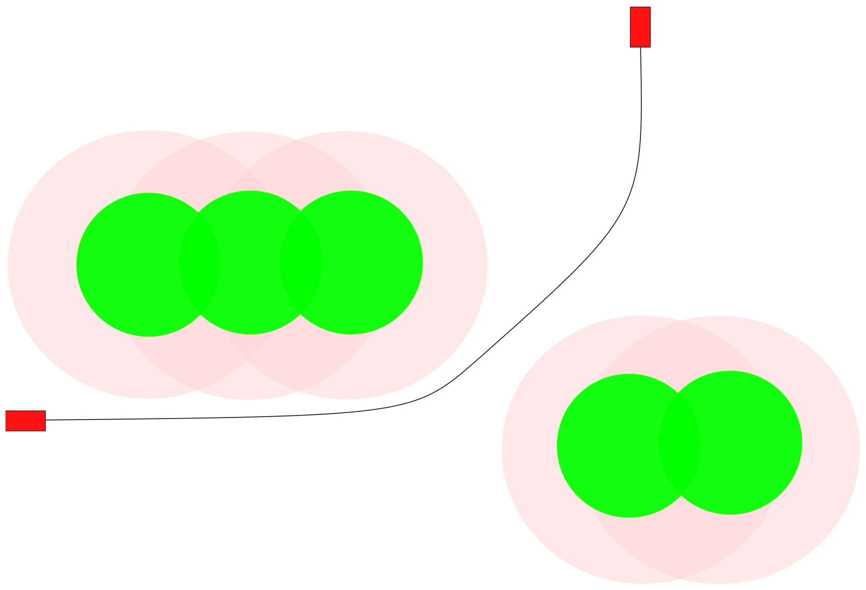


Figure 2.4: Islands

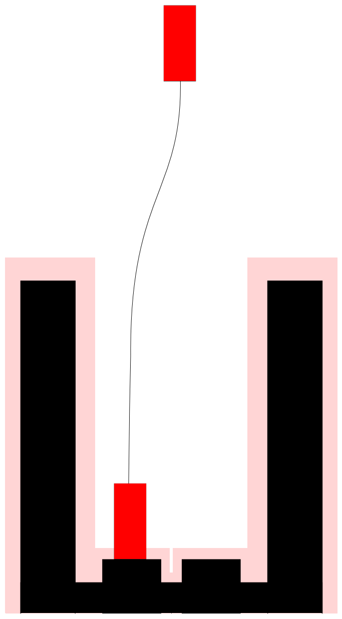


Figure 2.5: Harbor

2.1.3 Environmental Disturbance

One of the key elements to include in the path planner is environmental forces from wind, waves and current. Including these, and especially waves and wind, is one of the most challenging tasks concerning the path planner. Current can be included in the path planner using relative velocity. This is normal for including current in DP modeling of vessels without the need for knowledge of the vessel. Wind and waves are normally modeled as forces on the vessel, where wind is based on empirical tables for calculating the load on the ship. These tables and formulas use some information about the vessel, but the main problem is that the results are given in forces working on the ship. When the models presented in section 2.1.1 only use the kinematic equations with position, speed, and acceleration this is a challenge. Transforming forces to acceleration is normally done by dividing with the mass, but since the vessel mass is unknown in this case and also added mass and damping will affect this relation, this is not straightforward. For waves this is even more difficult since the forces from waves depends on many more factors and include much more complicated formulas. At the current state, only current is included in the path planner since there is no good method for including the others without making the model much more complicated and very specific for each vessel.

2.1.4 Map Processing

The planner in this thesis is designed to plan a path in a known map. It is therefore necessary to take input from a map and use this to describe the environment. This map can include information about obstacles, forbidden areas, weather, and other factors that can affect the vessel. This thesis will take a map from a picture made in Paint to keep it simple, but other projects have worked on how to extract information from marine maps. The maps used in this thesis are as shown in figure 2.6. The white areas represent open water where the vessel can navigate, and the black areas represent land and obstacles. Since the main focus in this thesis is the actual planning of a path between obstacles, and not the specific processing of different obstacles, two types of obstacles are considered, circles and rectangles. Circles are used to represent islands, and rectangles when considering docks and more complex environments. Circles and rectangles are also easy to handle with Matlab's image processing tools.

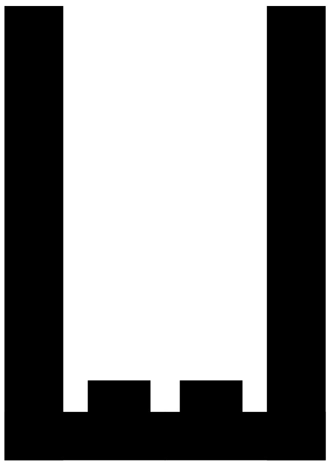


Figure 2.6: Example Map

To process the picture, the picture is imported and converted to a matrix where each entry represents a pixel in the picture. The picture is then converted from RGB color model (RGB) to gray-scale. Converting to gray-scale gives each pixel a value between one and zero. One represents black and zero is white. After this, all values above 0.5 are set to one and all below set to zero. The matrix will now contain only ones and zeros. Further, Matlab has built-in functions for detecting corners, edges and circles in this image.

Harbor environments, represented by rectangles, are processed by identifying the corners in the figure. These corners are then detected using the built in Matlab function detectHarrisFeatures. This function returns the location of each detected corner and the certainty of each of these points being a corner. These points are then sorted to get the top left point first. The list of points is then processed so the starting point is the top left and a function finding the next corner and drawing a line between these two points.

This function assumes that all lines will be either horizontal or vertical. This is not a complete solution but since the main task is the planning, this is decided to be good enough. Another possibility could be to detect the edges and follow these until the line changes direction. This would mean eight search directions instead of four. This would give the program the capability to find edges in all directions and use these to define the obstacles. At this point, the main difficulty finding lines that are not horizontal or vertical is the resolution in the picture. The file formats possible to process will in most cases operate with some rounding of numbers when reading the image and therefore lines will only be horizontal or vertical. All others will be built with small segments of vertical and horizontal lines.

The function detecting circles imports the picture using the same method. Then the built in Matlab function regionprops can be used to find the center of centroids, outputting the center of the circle and the size along the x and y axis. Taking the mean of these and dividing it by two equals the radius of the circle.

How to include both types of obstacles is shown in section 2.1.2.

2.2 Path Planner

The task of planning the path is viewed as a nonlinear optimization problem. There are two main approaches to solve the path planning problem, online planning or offline. Planning online means that the path will be evaluated and optimized while the vessel is moving, and offline means planning the path before the vessel starts. Online planning will in most cases give the best and most efficient path, since this method can handle changing information while the vessel is moving, but it also involves more computation since the planner must run multiple times during the operation. Path planning offline will on the other hand only plan for the information available at a given time, typically right before leaving the dock, but it will mean way less computations since it only runs one time. In this thesis the main focus will be a planner that runs offline. The reason for this decision is that for ferries, the information about the situation will most likely not change significantly during a crossing. For the specific docking phase, there is even less chance of much new information becoming available during the operation. Since the planner is implemented offline, it will be a nonlinear optimization problem.

Implementing an offline planner is done using the software CasADi in Matlab. The planner uses information about the environment, initial, and final conditions as input. With CasADi there are two possible methods for solving this problem. Method one is based on the Multiple Shooting example from CasADi (*CasADi* (2016)) and method two is based on the Opti stack in CasADi (*CasADi* (2017)). The main differences between these two approaches is the objective function, and how constraints on the path can be defined. Each method is tested with all models described in section 2.1.1.

2.2.1 Method One - Multiple shooting

As mentioned earlier, the first method is based on the example of multiple shooting from CasADi(*CasADi* (2016)). This method sets up the problem as an NLP problem. The code is shown in Appendix F, but it will be described here. First a brief review of the setup is presented, before some key parts are described more in detail. The first part is setting up the basics of the problem. This includes defining the time horizon for the planner, defining how many intervals the time should be divided in, and what time-step should be used in the discrete dynamics. The next part of the code is to set the numerical values for the initial conditions, final conditions, and constraints on states and control input. Since this method uses a cost function, it is necessary to define some additional functions that can be used in the cost function to control when different parameters should contribute to the cost. All this is necessary as a basis for setting up the optimal control problem.

The next part is defining the optimal control problem. This means using the framework provided in CasADi. The first part is to define the model variables and model. This includes states, control input, and model equations. The next part is defining the cost function. The cost function is both for obstacle avoidance and defining what terms should be optimized. With the cost function established, the system can be described as a continuous system. The next part is then to formulate the discrete dynamics of the system. This is the last part necessary to setup before the problem can be cast from an optimal control problem to an NLP, but a simple test is first performed to check that the problem is defined correctly.

The rest of the code is now an NLP. First this NLP is empty, before the initial conditions are applied. The next part is to formulate the NLP. This is done by defining an NLP-variable for the control action, integrate to the end of the interval,defining an NLP-variable for the states at the end of each interval, and defining the equality constraints. This is done over the whole time horizon, except the last time step, using a for loop.For the last time step, the end conditions are defined as constraints. This ensures that the vessel will finish at the correct position with the right orientation. The next part is setting up an NLP-solver, before using the solver on the formulated NLP. The last part is to process and display the solution of the NLP.

The cost function is used to limit the control input to the system, and to avoid obstacles. The most simple function is shown in equation (2.24)

$$L = a^2 + \omega^2 \tag{2.24}$$

where,
a - Acceleration
ω - Yaw velocity

ω can be switched with r if yaw acceleration is used.

For avoiding obstacles, the equation shown in equation (2.25) is used

$$L = a^2 + \omega^2 + \frac{w_{obs}}{close(x, y)} \tag{2.25}$$

where,
a - Acceleration
ω - Yaw velocity
w_{obs} - Cost on obstacle

The function close(x,y) is defined in equations (2.18) and (2.19). Using it with this function will then mean that the cost function will increase more when the vessel is close to the obstacle.

2.2.2 Method Two - Opti Stack

The second method is based on the Opti stack in CasADi (*CasADi* (2017)). As for method one, the code can be found in Appendix F, but a description of the code is given here. This method is similar to method one, but it offers better control when defining constraints. The start is similar but not the same as method one. The only value necessary to define in this method is how many intervals should be used. Time and time-steps are variables that the code should optimize. The next part is setting the numerical values for start, finish, and constraints on both states and control.

After this, the optimal control problem can be defined. This method defines the states as one matrix variable, with one row for each state and a column for each interval. The control is defined the same way as the states. Then the time horizon is defined as a variable that can be optimized. Then the term to optimize is set. Since the goal is given as a constraint on the final state later in the code, this optimizing will be to optimize the time used, control action used, and possibly speed in some Degree of Freedoms (DOFs). After defining what to optimize, the model equations are defined and a Runge-Kutta 4 integrator is used to integrate over all control intervals and link each interval to the next. This same loop adds constraints on the change on control input.

After linking the intervals, the constraints are defined using the Opti stack. Defining constraints is done with five types; equal to, less or equal to, larger or equal to, less than, and larger than. The different types of constraints are similar to method one, but with the Opti stack it is easier to include functions in these. This gives better control over obstacles and other constraints that are varying as the vessel moves. The last part is adding an initial guess for the time. This is calculated by assuming constant acceleration equal to the constraint on acceleration and finding the shortest distance from start to finish. Equation (2.26) shows how the initial guess can then be calculated.

$$Init\ Time = \sqrt{\frac{Shortest\ Dist}{a_{max}}} \tag{2.26}$$

2.2.3 Online Path Planner

Both the previous planners are implemented as offline solutions. Another approach is to have an online planner that can alter the path as the vessel moves from one position to the next one. This type of online planner would be more like an MPC since the optimization is done multiple times as the vessel moves. This enables the planner to alter the path as new information becomes available, such as new weather information and dynamic obstacles. Since the optimization now runs multiple times as the vessel moves, the optimization must run much faster to fit with the rest of the control system and not include delays that introduce unnecessary risk in the operation.

2.2.4 Planners

With a total of three models and two methods used in the planners, it is challenging to keep track of what planner is used. For the rest of the thesis, the planners will be numbered from one to six according to table 2.1.

Method	Model	Planner
Multiple Shooting	1	1
Multiple Shooting	2	2
Multiple Shooting	3	3
Opti Stack	1	4
Opti Stack	2	5
Opti Stack	3	6

Table 2.1: Planners

Model one is shown in figure 2.1 and equations (2.1)-(2.4), model two in figure 2.2 and equations (2.5)-(2.9), and model three in figure 2.3 and equations (2.10)-(2.15).

2.3 Control System

To test the planners, it is necessary to have a control system that can use the reference produced by the planners. This control system will vary based on how the vessel should operate. There will be two main types of operations considered in this thesis, transit and maneuvering/DP. Transit operations is when the vessel moves at high velocities and maneuvering/DP at low velocities. What is defined as low and high velocities will vary based on the size of the vessel, but velocities below 1 m/s are normally low and above are high velocities for ship lengths of around 80 meters. For both operations, it is necessary with an observer for state estimation and filtering of the signal. In this thesis, this is a nonlinear passive observer implemented in Johansen (2018). The controller will be different for each type of operation. Transit operations use a heading and speed controller. The heading controller is a PD-controller with PI-control on the heading reference to account for sideslip. The equations used in this controller are shown in equations (2.27)-(2.31).

$$\tau_N = -K_{p_\psi}\tilde{\psi} - K_{d_\psi}\dot{\tilde{\psi}} \tag{2.27}$$

$$\tilde{\psi} = \psi - \psi_d \tag{2.28}$$

$$\psi_d = \chi_p + \chi_r \tag{2.29}$$

$$\chi_r(e) = \arctan\left(- K_{p_\chi} e - K_{i_\chi} \int_0^t e(\tau)d\tau \right) \tag{2.30}$$

$$e(t) = -[x(t) - x_d(t)]\sin(\chi_d) + [y(t) - y_d(t)]\cos(\chi_d) \tag{2.31}$$

where,
τ_N - Force in yaw direction
ψ - Heading angle
K_{p_ψ} - Proportional gain for heading controller
K_{d_ψ} - Derivative gain for heading controller
χ_p - Path-tangential angle
K_{p_χ} - Proportional gain for relative course controller
K_{i_χ} - Integral gain for relative course controller
x(t) - Time varying global position in X-direction
y(t) - Time varying global position in Y-direction
x_d(t) - Time varying reference position in X-direction
y_d(t) - Time varying reference position in Y-direction
e(t) - Time varying cross-track error

The speed is controlled using a PI-controller shown in equations (2.32)

$$\chi_r(e) = K_{p_U}(U_d - U) + K_{i_U}\int (U_d - U)dt \tag{2.32}$$

where,
K_{p_U} - Proportional gain for speed controller
K_{i_U} - Integral gain for speed controller
U_d - Reference speed
U - Velocity

For DP operations, the controller is taken from Johansen (2018). The thrust allocation is also from Johansen (2018), but will vary dependent on the vessel and operation.

For the ferry simulator the difference in the thrust allocation will be on how many thrusters it can use. For DP-operations, both thrusters are used the same way as in Johansen (2018). For transit, only the aft thruster is used, and it can only operate in the range from $-\pi$ to π. For R/V Gunnerus, the tunnel thruster cannot be used in transit and the thrusters in the aft can only operate with the same angles as the ferry. In full scale the thrusters would be able to rotate all the way around, but to simplify the problem, the thrusters are constrained to avoid this. This incapability to reduce the speed using the thrusters, will then also be included in the path planned for this testing. In MCSim, there is no model of the thrusters, and therefore no need for thrust allocation. The force commanded in the controller, both heading/speed and DP, is applied directly to the model with a minor time delay.

2.4 Path Planner for Auto-docking Applications

The plan is to use the proposed planner offline to plan before the vessel departs from the dock. Since information about the weather isn't updated very frequently with existing sensors, there is no real need to update the path during transit. Since most Norwegian ferries operate in fjords where the historic data on weather is good and follows a certain pattern, with some exceptions, planning for some amount of different weather situations is possible. Planning a certain amount of possible paths based on the different normal weather situations for the ferry route that the vessel can choose from is deemed to be an effective method with the available sensors. If the vessel then gets information that the weather is very different from what it thought when leaving the other dock, it can choose from this list of possible paths, and find the path that best corresponds to the current weather.

Chapter 3

Simulations

The path planner in this thesis is developed with the docking phase as the main task, but it should also be able to handle longer distances with only minor changes. To test this different test scenarios are formulated and simulated, before testing in a simplified experimental setup. First only the planner is tested in the different cases, before the output is used as references to the simulators to see if the vessel can actually use the trajectory planned in the planner.

3.1 Test Scenarios

To test the planner, three main scenarios are tested. The first is the ability to plan an efficient path between two points in open water with no obstacles. The goal of this first test is to see if the planner can construct a trajectory that finishes at a desired position with a certain heading. This is the simplest test, but it is necessary to test this before testing more complicated scenarios. The next test is to include two obstacles in the path and see if the planner is able to plan a path around these obstacles and still end at the correct position with the correct heading and speed. The last scenario the planner will be tested in is a more realistic dock environment. The different scenarios are illustrated in figures 3.1-3.3.

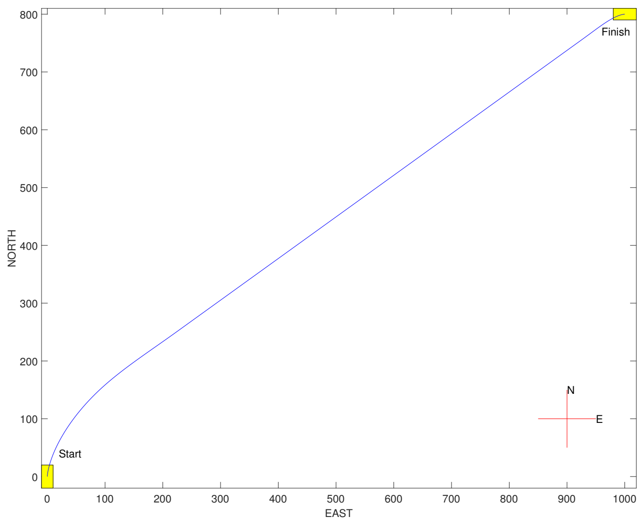


Figure 3.1: Scenario One

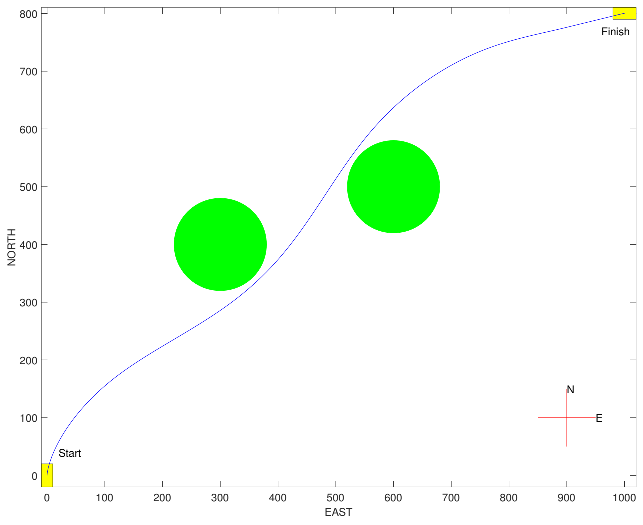


Figure 3.2: Scenario Two

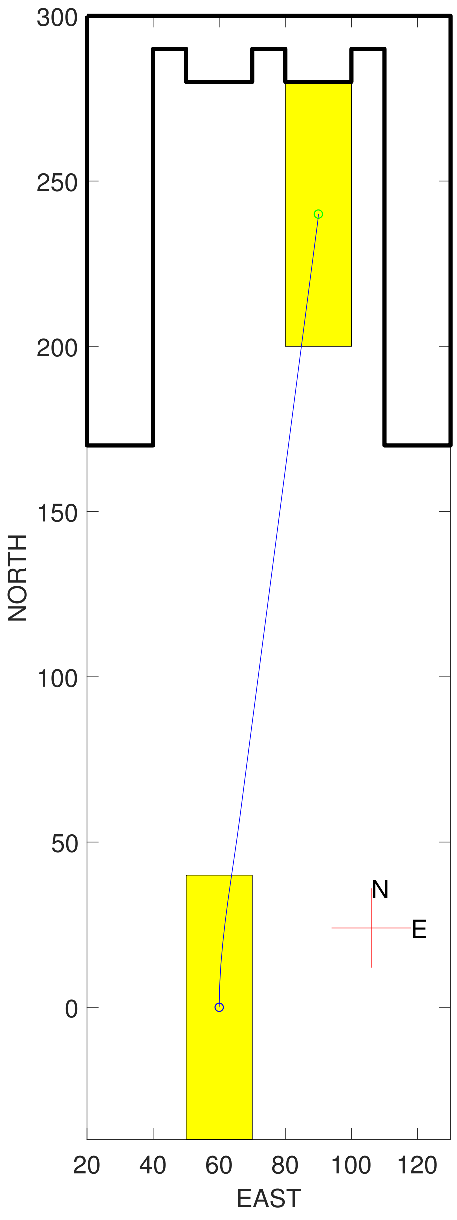


Figure 3.3: Scenario Three

These three scenarios will also be the basis for testing with simulators. However, since the simulators perform differently for each vessel, and include thrust allocation, it might be necessary to define new scenarios later. Since Cybership III (CS3) is so much smaller than both the ferry and R/V Gunnerus, there is not any point of testing this in the same scenario as described above. This is therefore tested in a different case with smaller distances. The harbor scenario is shown in figure 3.4.

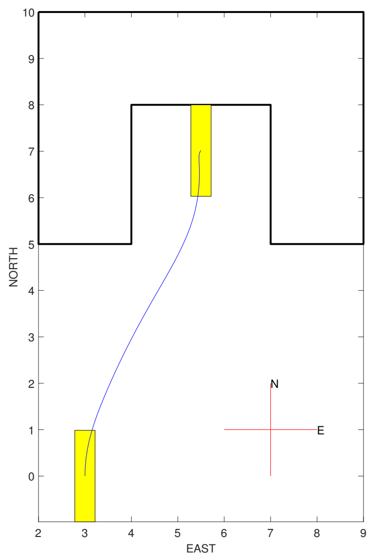


Figure 3.4: CS3 Test Case

3.2 Simulators

To test the planners, it is necessary to check how the references produced can be followed by control systems implemented on vessels. For this purpose, two simulators were developed in Johansen (2018) for testing. Since the experimental testing in this thesis is conducted in the Marine Cybernetics Laboratory (MCLab) at NTNU using Cybership III, MCSim is also used for testing. This is based on the vessel in MCLab, but it only uses models, and no hardware to simulate the vessel. Each simulator will be described in the following sections.

3.2.1 Ferry Simulator

The ferry simulator is based on a model provided by Rolls-Royce Marine. This model is the most relevant for the main problem, autonomous docking of ferries. It was developed in Johansen (2018) with a simple DP control system. The ferry model is a SINTEF Ocean VeSim vessel model. Since this model provides perfect measurements of all states, noise is added to simulate real signals. The ferry is fitted with two azimuth thrusters, one fore and one aft. These are modeled using very simplified models since the included thrusters appeared to have some unknown errors with the produced thrust. The control system implemented consists of a nonlinear passive observer, a Proportional–Integral–Derivative (PID) controller, and thrust allocation. In Johansen (2018), only DP was considered and implemented in the simulations. Since the thruster models are very simplified and assume no loss at high velocities, this simulator will be less accurate at higher speeds and DP was therefore the most relevant to consider.

The simulator is also expanded with a very simple heading and speed controller to allow testing of path following. This controller is implemented with a very simple PID-controller for heading and PI-controller for speed. With this controller, a new thrust allocation is also implemented. This new thrust allocation uses only the aft thruster to control the heading and speed of the ferry. Since this thruster is limited to only producing positive surge forces, slowing down and stopping the vessel will be impossible with this thrust allocation.

3.2.2 R/V Gunnerus

The second simulator developed in Johansen (2018) is based on the research vessel, R/V Gunnerus. The vessel model is implemented using the MSS toolbox (Fossen & Perez (2004)). R/V Gunnerus is equipped with two azimuth thrusters at the back and a tunnel thruster in the bow. The original model had no thrusters, so these were modeled with the same method as for the ferry model. The rest of the simulator is similar to the previous, with noise added on measurements, a nonlinear passive observer, a PID DP-controller,Heading/Speed control, and thrust allocation.

3.2.3 MCSim

MCSim is a simulation environment developed at NTNU for testing of marine control systems. This simulator has been developed at NTNU over multiple years and includes different models that can be simulated. The model used in this thesis is based on Cybership III developed as a part of Brodtkorb et al. (2018*b*). The simulator includes a control system with state estimation, DP controller, and thrust allocation.

A simple heading controller is implemented in the same way as for the ferry simulator.

3.3 Testing in Simulations

As mentioned earlier in this section, the different planners should first be tested by only looking at the results these provide. After this the resulting references should be tested using the different simulators. This section will first describe how each planner was tested and the setup necessary, before describing the setup of the control systems and planners when testing with the simulators.

3.3.1 Testing the Planners

Each of the six planners described in table 2.1 are tested in all three scenarios. This is to verify that the planners can plan a path from start to finish, and to decide what planner should be used further. Scenario one (figure 3.1) and two(figure 3.2) test the ability to plan for transition and scenario three (figure 3.3) is a docking scenario. When testing transition the vessel starts and finishes with zero velocity and in docking it starts with 1.5 m/s and finishes with zero. A full list of the parameters for the planners are shown in appendix A.

A description of testing with each scenario follows, but some results are similar in all three scenarios. The first, and most important result is that all six planners can solve the problem in all three tests, but they solve the problem very differently. The biggest difference is between the two methods. When using multiple shooting, the time the vessel can use must be specified before running the optimization. Since time is one of the parameters that should be optimized, this will practically exclude using multiple shooting further. The planners using multiple shooting also have challenges with obstacles. When using the Opti Stack, it is possible to specify constraint on functions, but in multiple shooting, constraints are only possible on inputs and states. This means that the Opti stack method can guarantee that the planner avoids obstacles, but when using multiple shooting, this is handled in the cost function with no guarantee. The cost function must therefore be tuned to avoid obstacles, but it must also be able to operate as closely as possible to these obstacles. The two methods also have some clear differences in computation time. Multiple shooting is faster for all cases, but since it has some critical faults for this specific thesis, the Opti stack is the preferred method. Both methods will be mentioned for each test case, but only the planners based on the Opti stack are considered for further use.

The rest of the results from this first testing will be discussed for each scenario.

Testing Scenario One

The plots showing the performance of all planners can be found in appendix B, but some of the results will be discussed here. This test is the basic test to verify that all six planners are capable of solving the problem. As mentioned above, all six planners complete the task.

The interesting results from this specific test is the time used to run the planners and how much time the path takes. As mentioned above, only the planners based on the Opti stack can actually optimize the time, but since this method also offers better control of both states and control input, the paths are actually slower compared to multiple shooting in this test. The time used on both the path and the computation is shown in table 3.1.

Planner	Time used on path	Time used for computations
1	368.09 *seconds*	2.58 *seconds*
2	368.09 *seconds*	3.72 *seconds*
3	368.09 *seconds*	6.38 *seconds*
4	412.68 *seconds*	10.53 *seconds*
5	417.05 *seconds*	13.80 *seconds*
6	416.92 *seconds*	23.20 *seconds*

Table 3.1: Results Test One

Of the planners using the Opti stack, planner four is the fastest, both in terms of the actual path time and time used on computations. But since this uses heading velocity as a control input to the model used in the planner, it is more difficult to get a smooth reference signal to the controllers used on the vessel. Between planner five and six, there are very small differences, but for this specific test, planner five performs better. It uses one second longer on the path, but it uses under half the time in computations. Since the vessel in most cases would use a heading controller, it is also logical to avoid velocity references sideways since the controller cannot handle these anyway. The plots from this planner are shown in figure 3.5

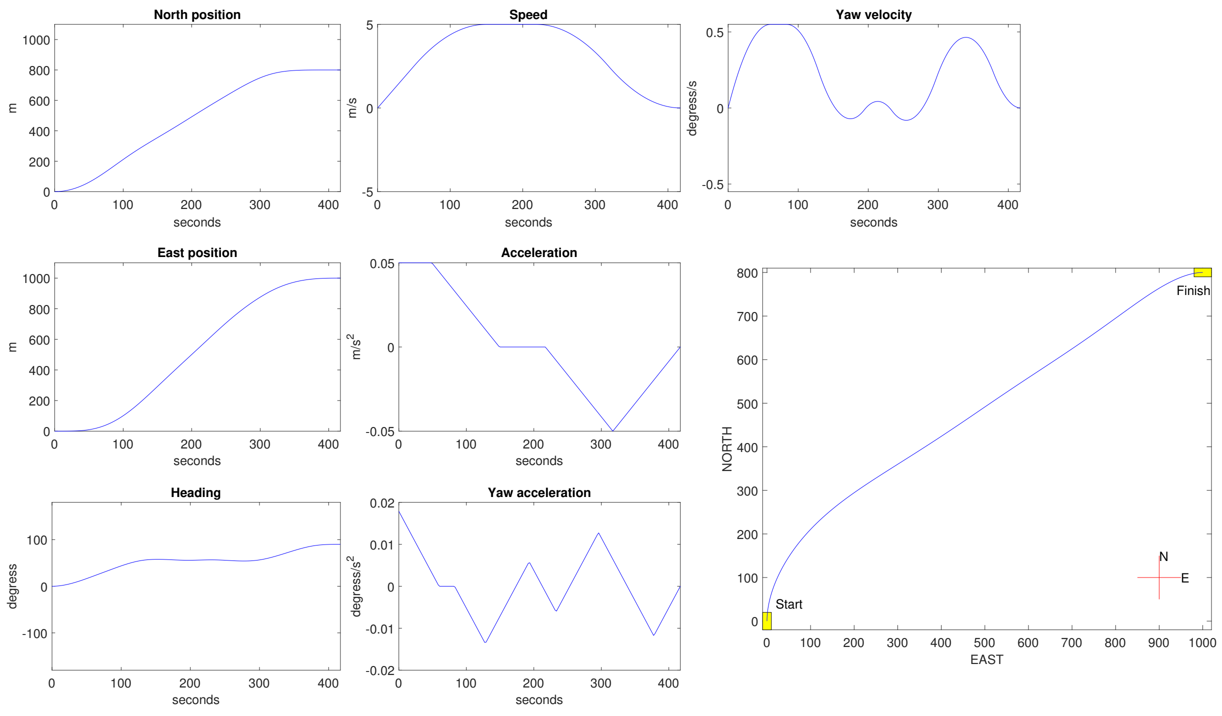


Figure 3.5: Test 1 Planner 5

Testing Scenario Two

This test is similar to test one, but with two islands in the path that the planner must avoid. In this test, the differences between using multiple shooting and the Opti stack are more clear. Using the Opti stack now both plans a faster path, and handles the obstacles best. Multiple shooting is still the fastest method in terms of computation time. The time used for each planner is shown in table 3.2 and the output from each planner is shown in appendix B. As in test one, the best planner is number five. Since the task is a transit operation, this is the best combination of control and computation time. The result from this planner is shown in figure 3.6.

Planner	Time used on path	Time used for computations
1	464.11 *seconds*	3.48 *seconds*
2	464.11 *seconds*	5.59 *seconds*
3	464.11 *seconds*	7.88 *seconds*
4	420.95 *seconds*	55.20 *seconds*
5	460.47 *seconds*	55.09 *seconds*
6	459.78 *seconds*	75.80 *seconds*

Table 3.2: Results Test Two

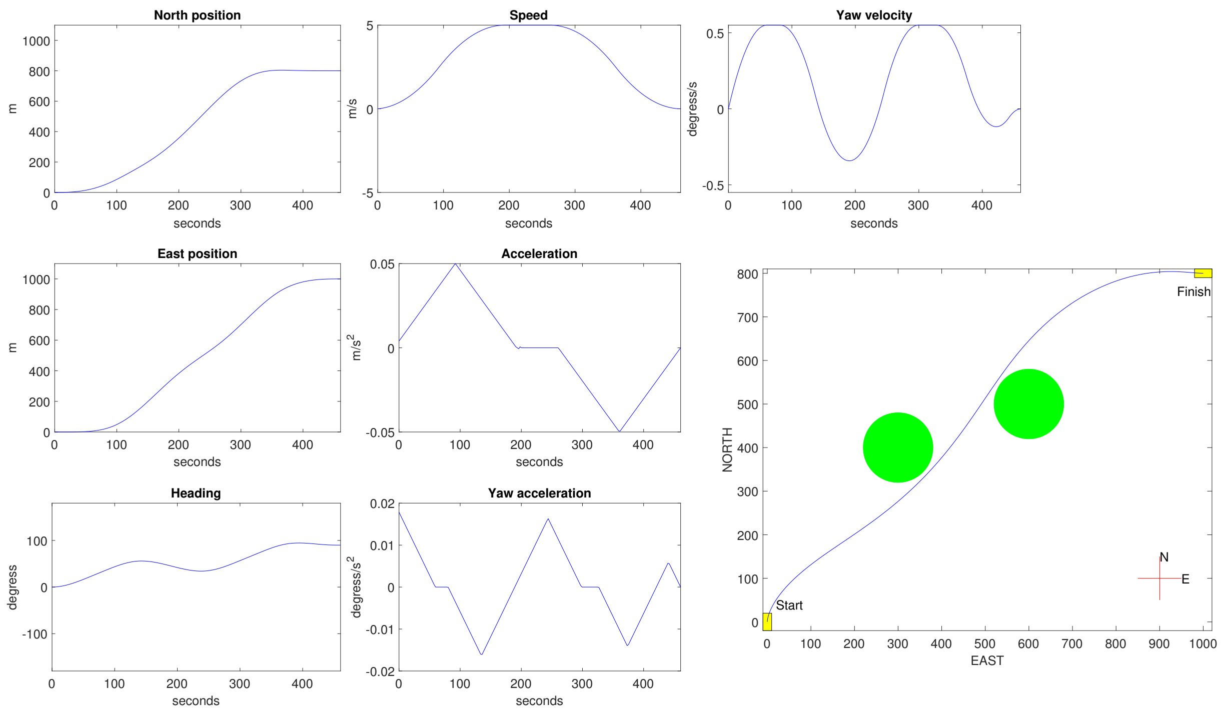


Figure 3.6: Test 2 Planner 5

Testing Scenario Three

Testing scenario three is the most relevant task for docking operations. This test is to verify the capability of planning a path as the vessel is docking. The test is set to the last 200 m before the dock. The constraints are the same as for test one and two, with the exception of velocity. Since the distance of the path now is smaller, the number of intervals used in the planners is now reduced from 200 to 50. This reduces the time necessary to compute the solution, while still checking enough points along the path.

Planner	Time used on path	Time used for computations
1	201.70 *seconds*	21.42 *seconds*
2	201.70 *seconds*	27.27 *seconds*
3	201.70 *seconds*	12.39 *seconds*
4	213.97 *seconds*	21.19 *seconds*
5	214.22 *seconds*	21.45 *seconds*
6	214.19 *seconds*	21.19 *seconds*

Table 3.3: Results Test Three

The tests are shown in appendix B, but the most important results are summarized here. All six planners manage to solve the problem, as before, but this task proved to be more difficult than the other tests. The dock is added as lines that the vessel must keep a minimum distance to, and even this simple geometry with 17 lines adds complexity to the problem. As before, only the planners using the Opti stack can actually solve the task of optimizing the time and guaranteeing that the vessel avoids obstacles.

Since this task is maneuvering at low velocities it is possible to move in all directions. Planner six is the best planner since it is the only planner giving a reference possible to follow for a DP-controller. The results from this planner are shown in figure 3.7

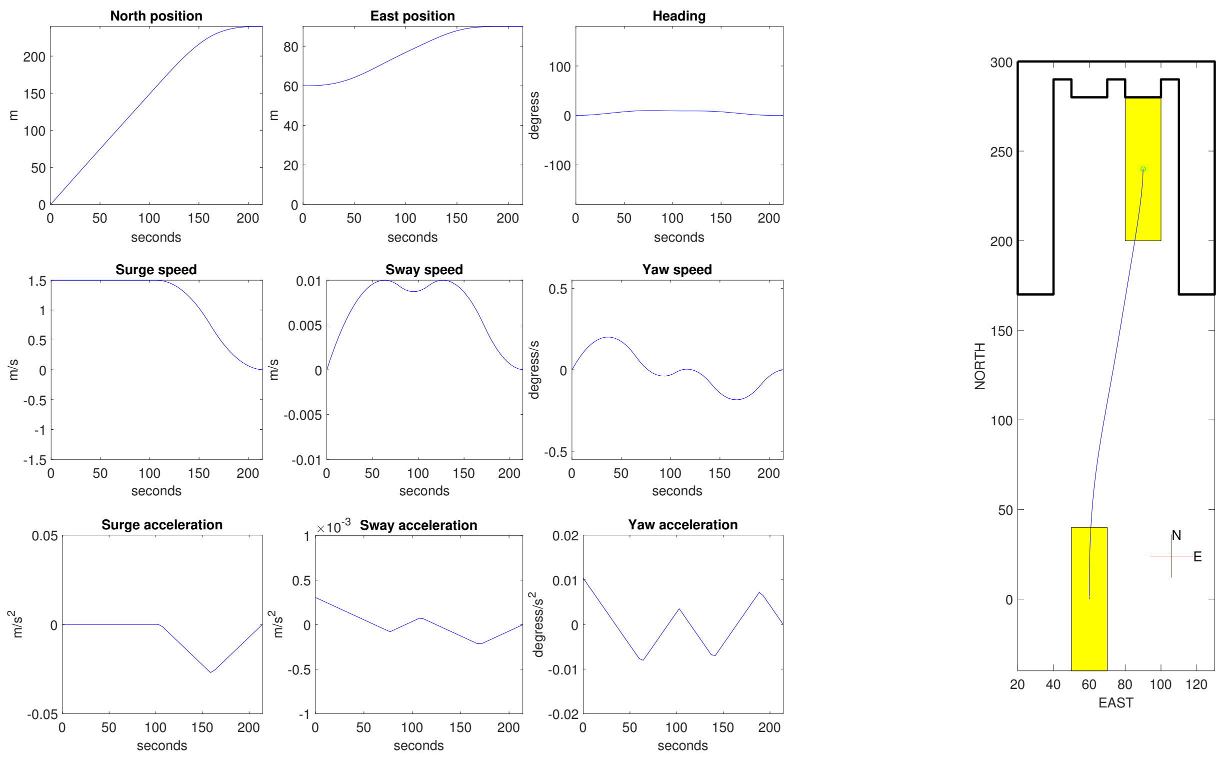


Figure 3.7: Test 3 Planner 6

Summary of Planner Testing

Testing all six planners verify that they can plan a path in all three scenarios, but as mentioned above, only the planners using the Opti stack can optimize the time, and use constraints to avoid obstacles. The two planners that perform best are planner five and six. Planner five is the best planner for transit and planner six is the best for docking. These will therefore be used further in testing for DP and transit respectively.

3.3.2 Testing with Vessel Simulators

After verifying that the planners work and manage to solve the different tasks, the paths must be verified with the simulators presented in section 3.2. To test this, the planners are run offline to produce reference signals that can be used in the control system for each simulator.

DP Test with Ferry Model

This test is to verify that the ferry model from Johansen (2018) is able to follow the reference signal from the planner. The environment is as shown in figure 3.3 and the reference is shown in figure 3.7. The control system is tuned to handle the reference, and the basics of the control system are the same. The specific tuning of the control system is shown in appendix C. This test shows that the planner is capable of planning a path that the DP system on the ferry can follow. The output from the simulations is shown in figure 3.8.

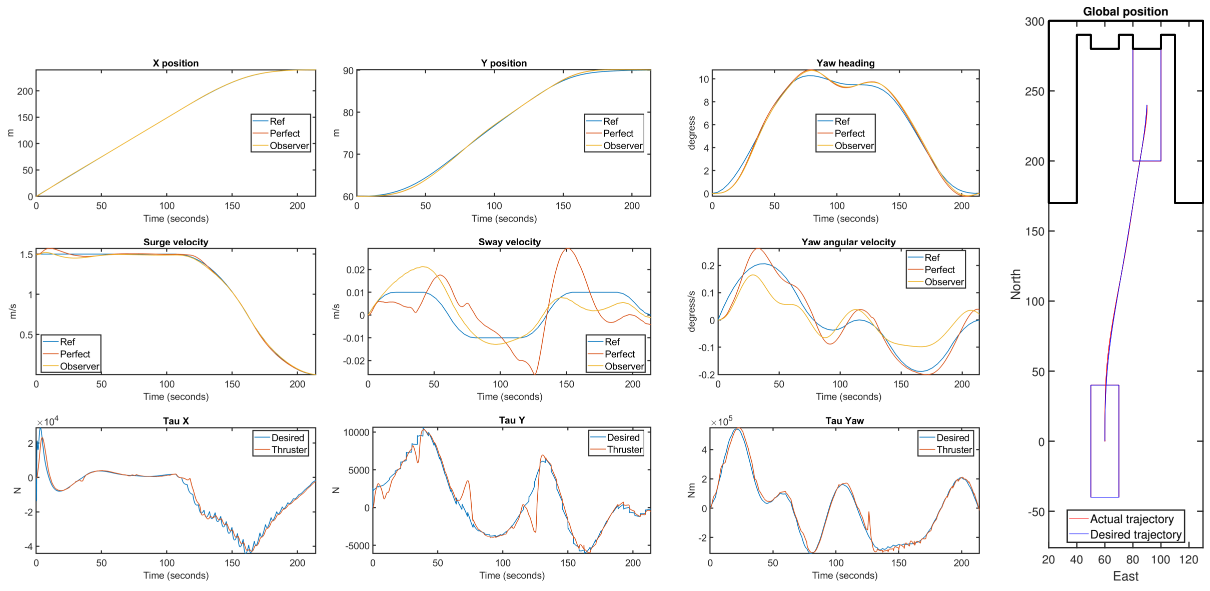


Figure 3.8: Ferry Test One

The plots show that the ferry follows the reference well, but since the control system is not perfectly tuned, it misses the reference with a few centimeters. Since the ferry is assumed to be 80 m long the path traversed in this docking test is 250 m, so this small error is accepted. This test is conducted with no noise on the measurements in the simulator and no environmental disturbances. Since the test is more a validation of the system and to check if the different parts are compatible, this is enough.

DP Test with R/V Gunnerus

This test is to check if the planner is compatible with the simulator based on R/V Gunnerus from Johansen (2018) and to see if the planner can be used with different vessels with minimal changes. The setup is similar to the test with the ferry model, but since the system on R/V Gunnerus has two azimuth thrusters at the back and a tunnel in the bow, it cannot stop as fast as the ferry. This means that the acceleration must be lower for this control system. Since the vessel is much smaller, the velocity must be lower than for the ferry simulator. To avoid very long simulation time, the distance from start to finish is set much lower for this test. The results from this test are shown in figure 3.9.

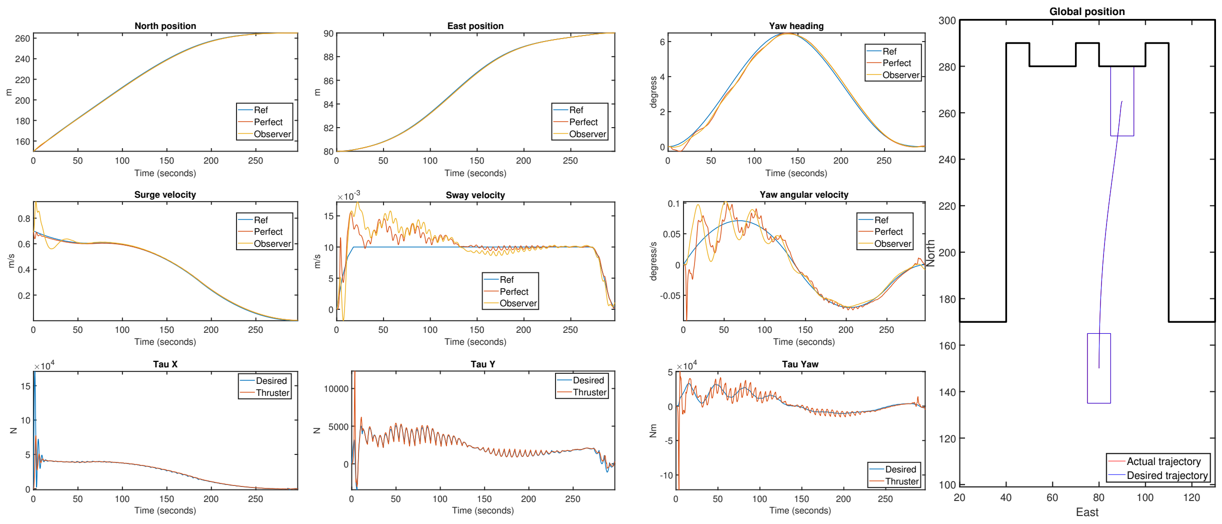


Figure 3.9: R/V Gunnerus Test One

This test shows that it is possible to use the same planner for multiple vessels, and only change the constraints on states and control input for the planner. The vessel follows the reference well with this configuration and tuning. This test also highlighted the need for adjusting the constraints for different vessels. When testing with the same limits as the ferry, the vessel was unable to follow the reference.

DP Test with MCSim

The last simulator to test the DP case with is MCSim, but since the vessel in this simulator is much smaller, the case must be altered. A special map with much smaller distances is therefore used for this test. The map is shown in figure 3.4. This test is both to verify the capability of using the same planner for multiple vessels, but also to test the system before model testing. Since MCSim is based on Cybership III, and this is the model used in the model tests, this is an important test as preparation before laboratory testing. In this test, the constraints on both states and control input were also scaled down to account for the smaller size in the test. The results from this test are shown in figure 3.10.

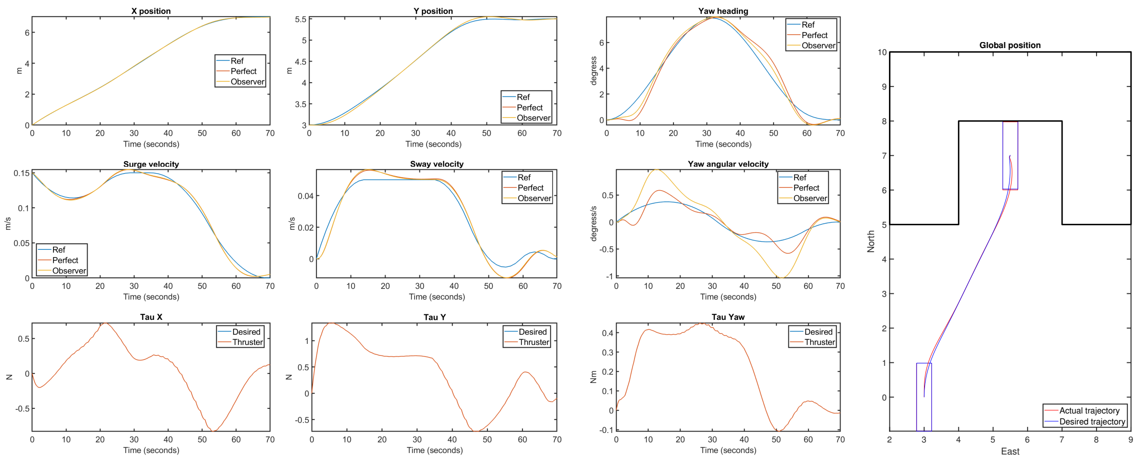


Figure 3.10: MCSim Test One

The control system performs well and is capable of following the reference almost perfectly. The difference is small, even compared to the two previous tests, but since the distance in this test is much smaller, the difference is more visible in this test. The main problem is the small overshoot in north position, since this means the vessel will collide with the dock, but in this case, the overshoot is only a few centimeters and the speed is almost zero.

Total Performance in Simulators for DP Tests

Testing the DP planner in the three different simulators shows that the planner can be used for different vessel types with only small differences. Since the goal of this thesis was to develop a planner with the capability to handle multiple vessels, this is very important. But the testing also shows how important it is to adjust the planner to the corresponding control system and the vessel to get good performance. Especially the test with the model of R/V Gunnerus showed the importance of this. With correct adjustment of the planner to the different vessels, these tests show that the planner is capable of handling multiple vessels and control systems and provide references that the DP-controllers can follow. It also shows that it is easy to alter the planner for different vessels since only the limits on states and control input must be adjusted.

Transit Test with the Ferry Simulator

This test is to verify that the planner can be used for the transit part of a ferry operation and not just the docking phase. To test this, planner five is used. This planner outputs a reference in both heading and speed, but it limits the amount of states to avoid unnecessary computations. This test uses a simple PID-controller for heading control and a PI-controller for speed control. The first output tested is the reference shown in figure 3.5. This reference is not possible to follow since it stops with zero velocity. This means that the vessel must stop using the same amount of time as it used to reach the transit speed. Since only the aft thruster is used, and this only produces a positive surge force, this is not possible. The case must therefore be changed so the vessel will not have to stop. Assuming that the transit phase starts when the ferry has left the dock and reached a velocity of $1.5 m/s$ and ends when it starts to reduce its velocity to prepare for docking fixes this problem. The next challenge is to avoid sideslip. Only using a PD-controller to control the heading introduces much sideslip and the vessel will not follow the trajectory, even when it follows the reference well for both heading and speed. When introducing the PI-controller to counteract sideslip, the ferry struggles with producing enough thrust when following the reference shown in figure 3.6. It was therefore necessary to adjust the planner so the reference is compatible with the control system on the ferry. The main problem was the high turning velocity at high velocities. Limiting this, and after some adjusting on the control, the control system was capable of following the velocity reference, and the trajectory follows the desired trajectory almost perfectly. The results from this test are shown in figure 3.11.

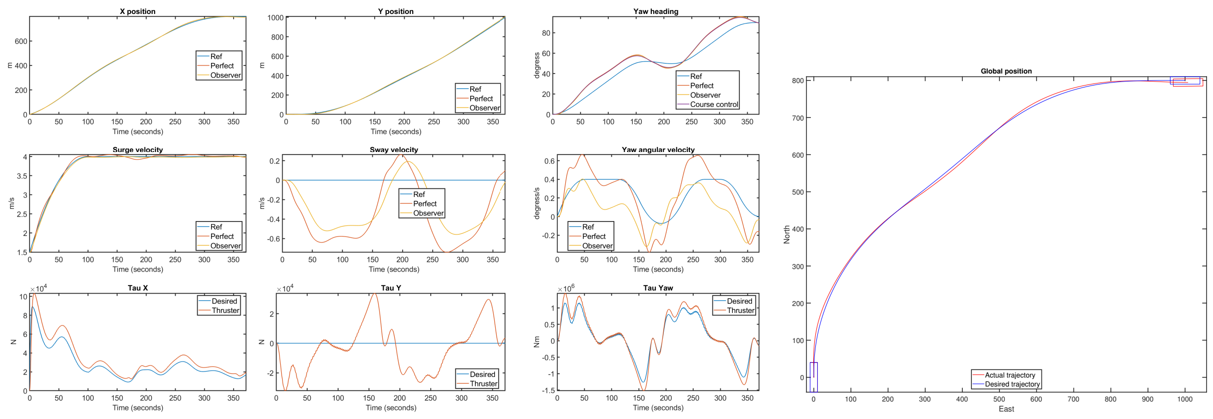


Figure 3.11: Ferry Test Two

The next part is to check if the planner can plan a path between two islands that lay in the natural path, that the control system can follow. The plan was initially to test on case two, but this has the same challenges as described above. The test was therefore altered to be a straight line, but with two islands laying on, or close to, this straight line that the vessel must avoid. The initial velocity was set to 1 m/s. The results from this test are shown in figure 3.12. The ferry follows the reference in speed well, and the actual trajectory follows the desired trajectory very well. The vessel is at no time in risk of colliding with the obstacles which is the most important part of this test.

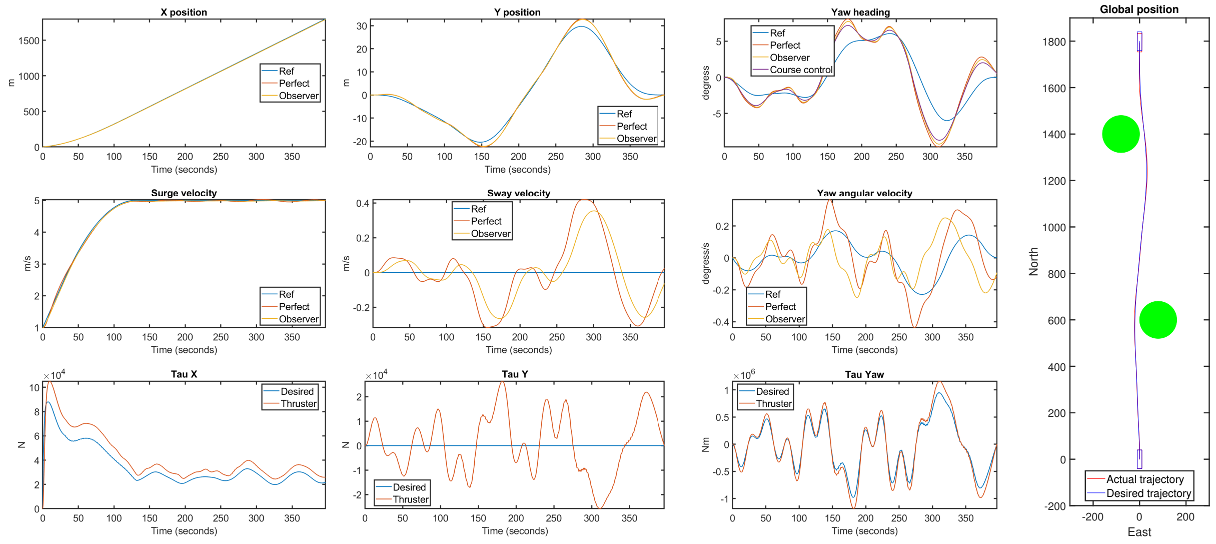


Figure 3.12: Ferry test three

Transit Test with MCSim

To test if the planner can be used for transit on multiple vessels, and since the model testing later is with CS3, it is necessary to test on the model of CS3 in MCSim. Similar to the DP tests, it is necessary to reduce both the distances, velocities, and accelerations for this test since the vessel is so much smaller than the ferry. Another challenge using this simulator is the thruster models. Since this simulator is missing accurate models of the thrusters, it is difficult to say how the vessel will react in model tests, but it is also difficult to decide how high the limits on velocity and acceleration can be. The limits used are therefore taken from the references used in Brodtkorb et al. (2018*b*). Two tests where initially planned. One starting with forward velocity and then taking a sharp turn and finishing with the maximum velocity, shown in figure 3.13, and one moving in almost a straight line shown in figure 3.14.

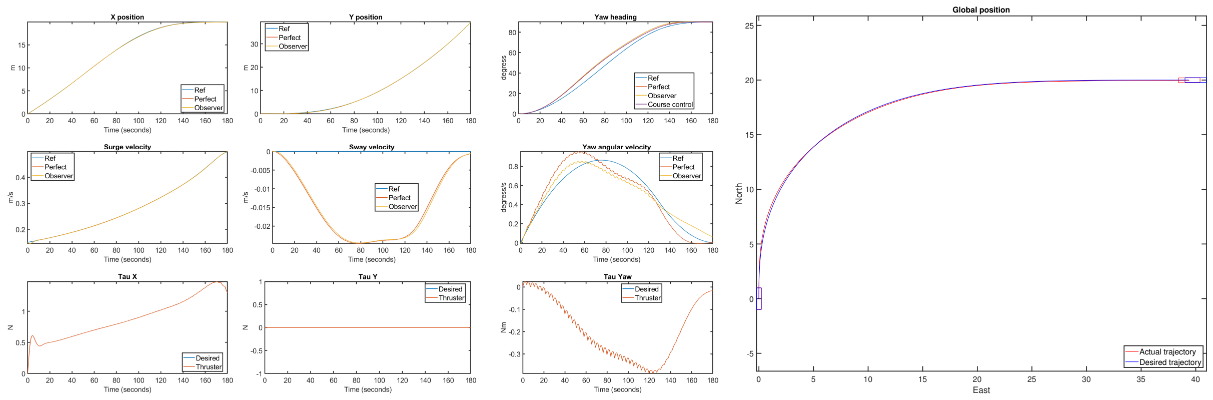


Figure 3.13: MCSim Test Two

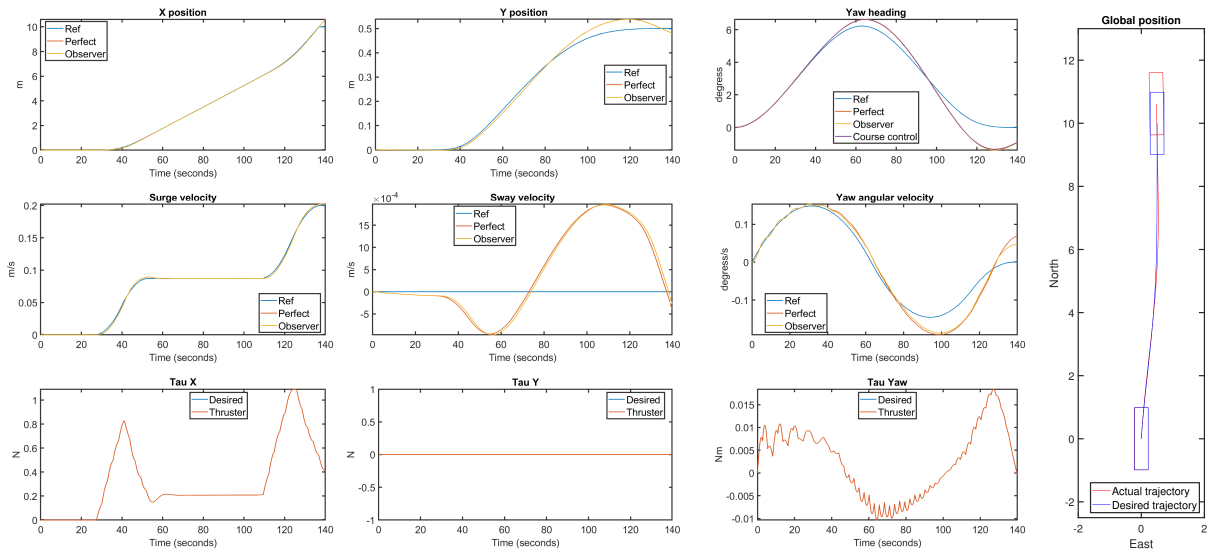


Figure 3.14: MCSim Test Three

These show that the control system is capable of following these references sufficiently, but it is difficult to know if the vessel is capable of following them in model testing. The last test with this simulator is to check if the planner can avoid obstacles and the vessel can follow the reference around these obstacles. The case used is then the same as shown in figure 3.14, but with a circle placed in the preferred path. The results are shown in figure 3.15.

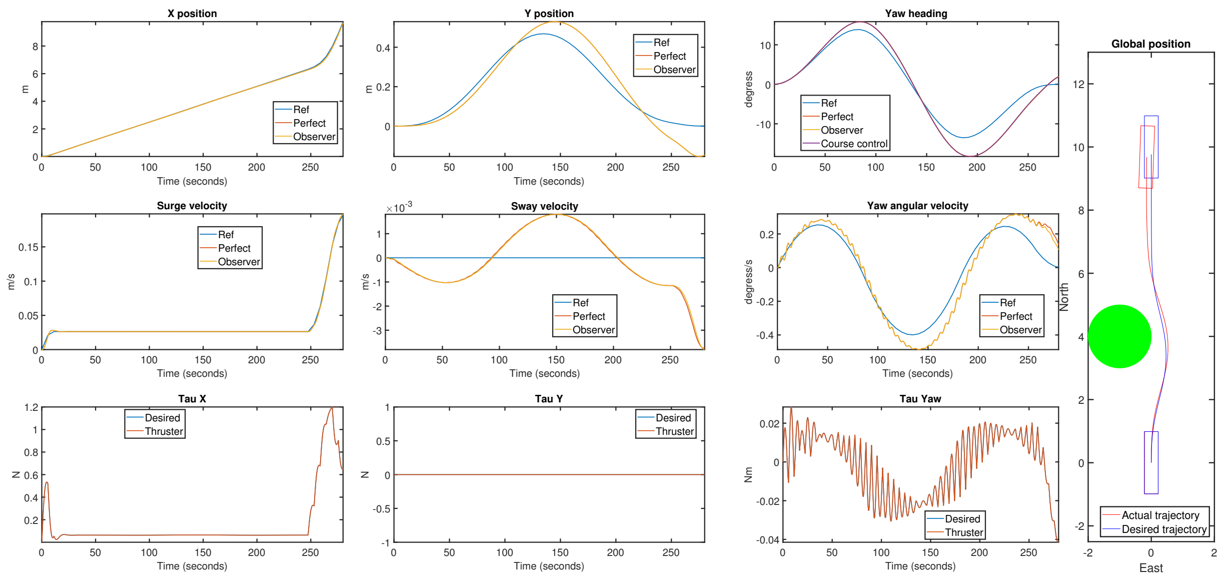


Figure 3.15: MCSim Test Four

The planner handles the obstacle, and the control system follows the reference well enough to avoid the obstacle, but it is not as accurate as the other tests.

Transit Test with R/V Gunnerus

The last simulator to test the planner in transit is based on R/V Gunnerus. This is tested in one test scenario with no obstacles. Ideally it should also be tested with obstacles and more challenging environments, but since the simulator runs very slow and testing with the other simulators show that the planner can handle obstacles, the decision was to skip this. This test is the clearest evidence that the big challenge is to adjust the planner to the control system. With correct tuning, the system is capable of following the references quite well. One important thing in this case is the use of perfect measurements and not observer measurements. The reason for this is the error from the observer, especially in surge velocity. The results from this test are shown in figure 3.16.

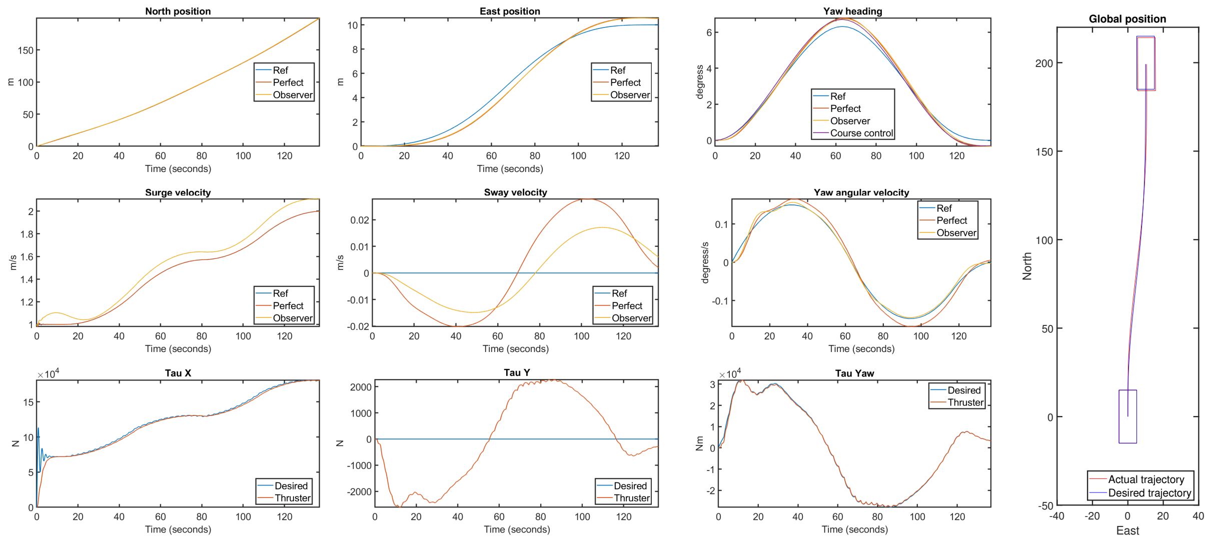


Figure 3.16: R/V Gunnerus Test Two

Total Performance in Transit Testing

Testing the performance of the planners in transit shows mainly two important things. The planners are able to plan routes in transit as well as docking/DP and handle obstacles, but adjusting the planner to the control system and vessel is very important. The main result is therefore that the planners work, but they must be tuned and tested to ensure both convergence in the planner, and compatibility with the control system. In these tests, it is also important to remember that all simulators use models for low velocities, but since the goal here is more a proof of concept than completely accurate results and even high speed models at this level would include some errors, this is not critical.

3.4 Summary from Simulation Testing

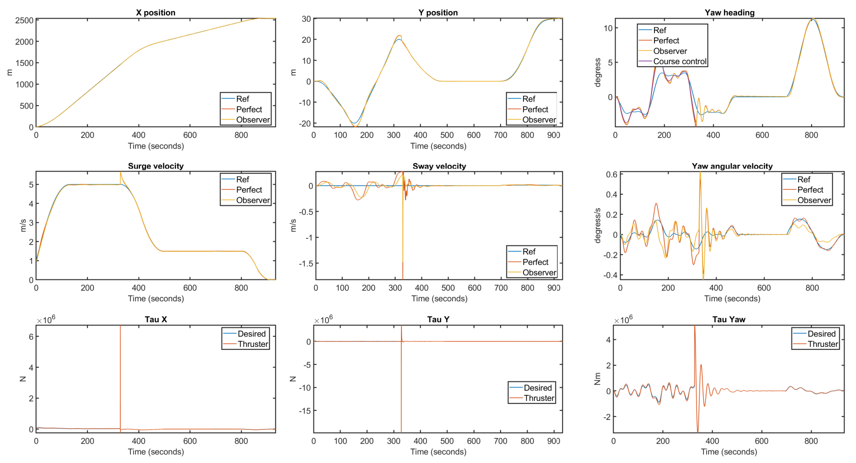
To ensure that the planners work and can work in combination with different vessels, it is important to test in simulations before model testing and, eventually, full-scale testing. In this thesis, testing is divided into two parts. First, only planners are tested to check if they actually can plan a path that reaches the goal and try and decide what planners solve this task best. Then the best planners are used to produce references for control systems implemented in three different simulators to test if the references are possible to follow. This testing in simulators is also necessary as preparation before model scale testing.

All six planners were tested first and all six planners are capable of planning paths in the different scenarios, even though they perform differently. The three planners using the Opti stack are the only three capable of actually optimizing the time within the given constraints and use constraints for obstacle avoidance. Since these are crucial parts of the task outlined in section 1.5, only these three were considered for further use. Of these three, planner five performed best in transit and planner six in docking. These were therefore used further in testing with simulators. This first testing also shows the importance of how the problem is defined. If the problem is poorly defined, it will be difficult to find the optimal solution.

Testing the reference signals with the control systems implemented in the three simulators verifies that the planners can be used with existing control systems, but it also highlights some challenges. How the optimization problem is defined and what values are used for the constraints are now even more important. These tests also show how important it is that the control system works well. The planners can perform well, but if the constraints are too high or the control system is poorly tuned, the vessel will never be capable of following the reference.

3.5 Dock to Dock Path

As stated in section 2.4, the proposed method for path planning in the docking phase is selecting a path from a list of planned paths based on the environmental situation. A situation like this is shown in figure 3.17. The ferry starts from (0,0) with a planned path based on the conditions at that time. Before the ferry reaches the dock, updated measurements on the current are available, and the ferry can then update the path for the last part of the path. In figure 3.17, the ferry starts docking when it reaches the point (2300,0), so from there a number of paths can be planned for different conditions.



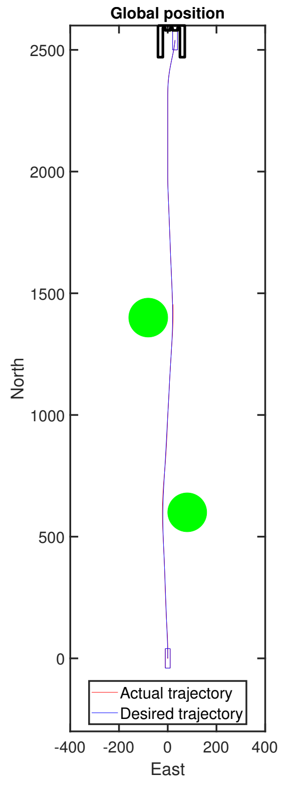


Figure 3.17: Practical Path Planning

The path shown in figure 3.17 consists of one trajectory from start to (2000,0), one from (2000,0) to (2300,0) and one from (2300,0) to finish. The reason for this, and also the explanation for the strange behavior shown in the plots is the switching between the different controllers and corresponding thrust allocation. Since the trajectory is time dependent, the ferry has a very distinct position it's supposed to reach at a given time. When the ferry does not reach this, the controller still switches, and this means the new controller spikes and commands too much thrust for the thrust allocation to provide. Since this figure is only to illustrate the proposed way to use the planner, the thrust allocation is omitted in a switching phase. The force commanded in the controller is then applied directly to the ferry until the ferry catches up with, and stabilizes, at the reference trajectory.

Chapter 4

Experimental Testing

4.1 Experimental Setup

4.1.1 Cybership III

Cybership III is a model ship used for testing at the MCLab at NTNU. The model is a 1:30 scale model of a supply vessel and weighs around 70 kg. The model has three azimuth thrusters, one in the bow and two aft, and one tunnel thruster. Only the three azimuth thrusters will be used in this thesis. The model is controlled using a National Instrument CompactRio. This unit communicates with a host computer using WLAN. The computer is used to log and implement the control system for controlling the vessel. Onboard the vessel, there is also a Raspberry Pi Computer (RPi). The vessel is controlled using the on-board RPi. On this, a C version of the simulink code built on the host computer is used. This RPi is connected to the CompactRio and all this communicates with the host computer and is controlled using Ni VeriStand.

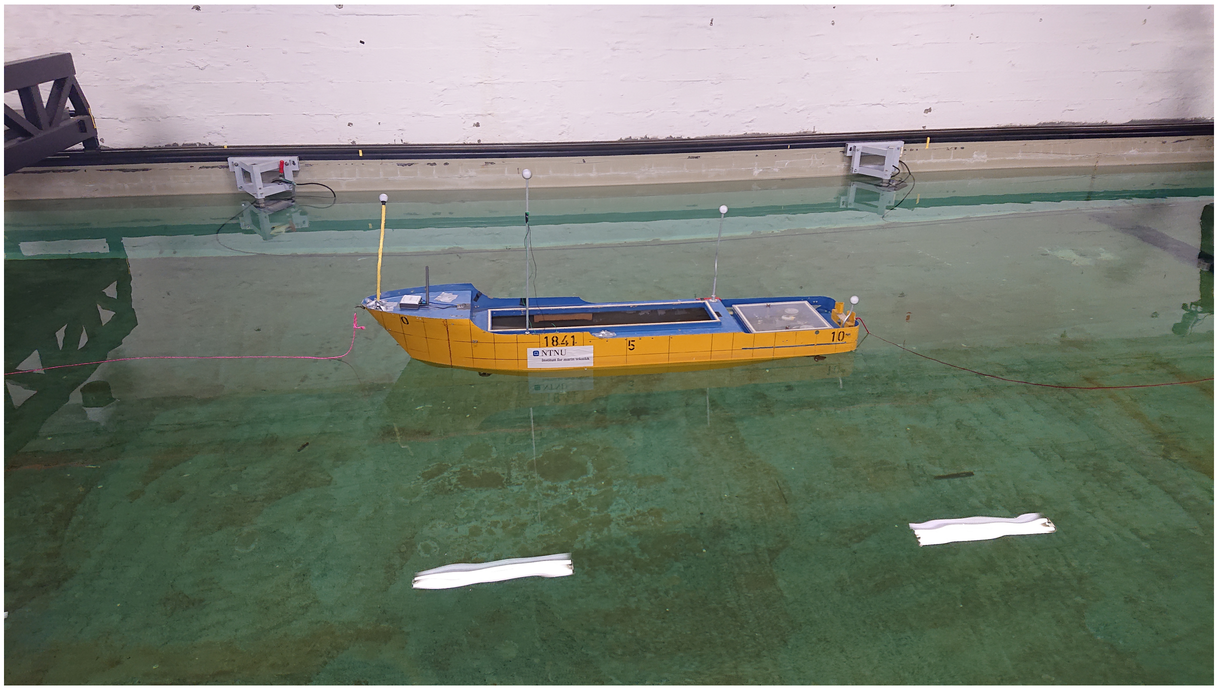


Figure 4.1: CS3

4.1.2 Marine Cybernetics Laboratory

The MCLab is a laboratory at NTNU with a small basin that can be used for testing marine control systems. The basin is 40 m long, 6.45 m wide, and 1.5 m deep. The lab has one towing cart that can be moved almost the whole length of the basin, and a visual positioning system mounted on this cart. The positioning system is a Qualsys Motion Capture System with three Oqus high speed infrared cameras. These cameras have a limited field of view meaning that the actual area possible to use in the lab is much smaller than the total size of the basin. The area where the cameras actually provide measurements is around 6 m by 4 m, but the exact size will depend on the calibration of the cameras. The basin also has a wave generator installed. For more data and information about the laboratory, the reader is referred to *NTNU* (2017).

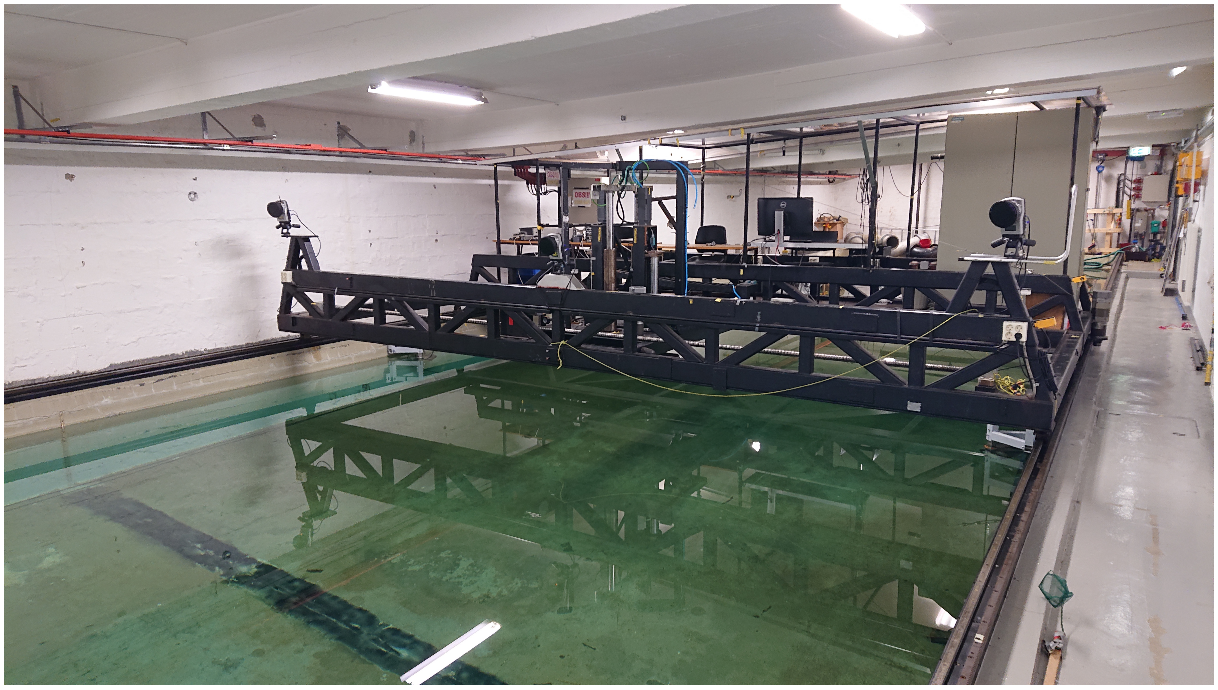


Figure 4.2: Oqus Camera System

4.1.3 Lab Control System

The vessel is controlled using a control system implemented from Brodtkorb et al. (2018*a*) for low velocities, and a heading controller for maneuvering at higher velocities. As a preparation before testing the planner in the laboratory and to get to know the setup the DP- and heading-controller were tested. These were very simple tests to get familiar with the setup and to get a sense of the performance of system before introducing new elements.

The DP-controller was tested by first moving the vessel forward 2 meters and then pushing the vessel forwards, backwards, and sideways. Between each push, the vessel was given time to return to the desired position, and stabilize around this. The plots from this testing are shown in figure 4.3. The vessel is pushed in negative x direction at approximately 190 seconds, positive x direction at 270 seconds, positive y direction at 340 seconds, and negative y direction at 420 seconds. These show that the vessel is capable of maintaining the position but that it will oscillate around this position.

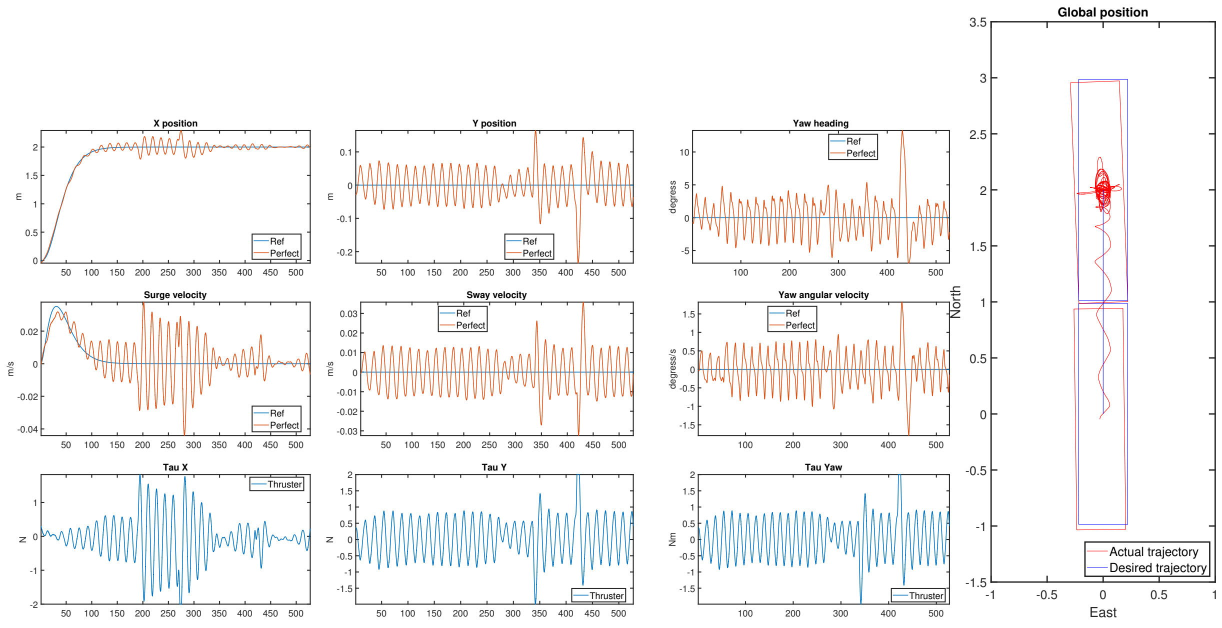


Figure 4.3: MCLab DP Test

To test the heading controller the controller was given a list of five way-points shown in table 4.1

Way-point	Position
1	(0,0)
2	(2,0)
3	(5,1.5)
4	(8,0)
5	(12,-1)

Table 4.1: Way-points for Heading Test

Around these points there is a circle of acceptance of 2 meters so when the vessel is closer than 2 meters, the guidance controller starts turning the vessel towards the next point. This testing is shown in figure 4.4. The vessel follows the reference in heading well so the heading controller is working well, but the guidance controller could be altered so it follows the path more directly between each point.

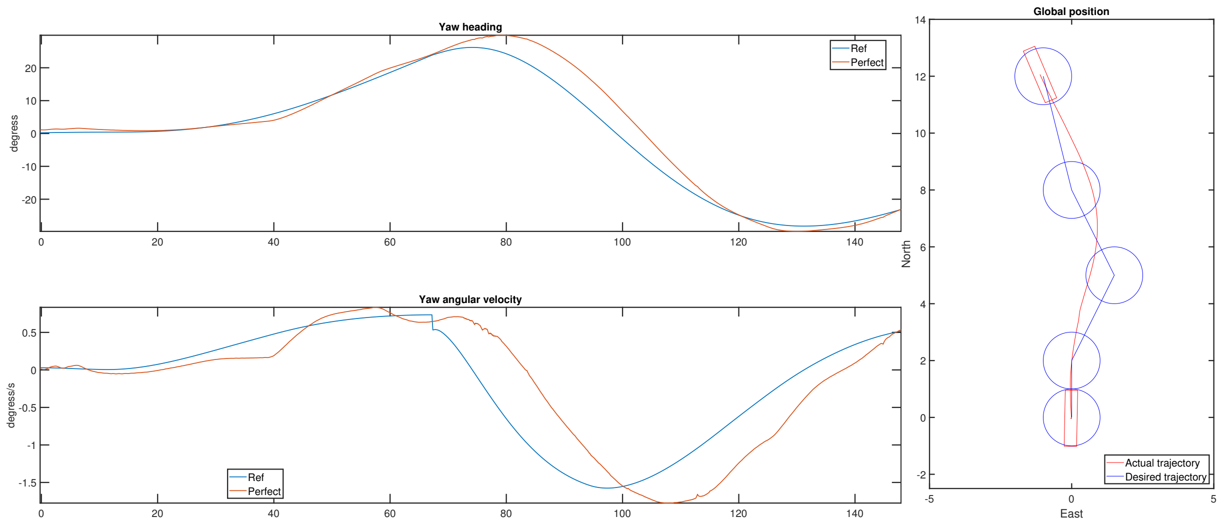


Figure 4.4: MCLab Heading Test

The conclusion after testing both the DP- and heading-controller is that these will work well enough to test the planner, but both controllers could be improved further.

4.1.4 Position Manipulating

The visual positioning system only provides accurate positioning in a limited part of the basin. The precise area will vary with the calibration of the cameras, but approximately 6 m by 4 m seemed to be the normal size. To allow testing longer paths, it was therefore necessary to manipulate the positioning system to expand the area where the cameras could track the vessel. Since the cameras are mounted on a towing cart that can be moved along the basin, the idea was to manipulate the measurements so the cart could be moved while testing. Only moving the cart means shifting the origin's position in the basin, but the vessel will think that the origin is standing still. This is a nice method for introducing current, or other unmeasured disturbances in the system, but not for expanding the area available for testing. To achieve this, the speed of the cart must be added to the differential equations when the cart is moving. To allow multiple starts and stops for each run, both the system in Simulink and VeriStand were altered so the speed of the cart was added when activating this in the workspace console. The speed of the cart could also be varied during the run if the cart was catching the vessel or the vessel was going too fast. These small changes allow testing longer paths, more similar to transit operation. Paths up to 16 m were tested, but it should be possible to run tests longer than this as well with the proper settings and calibration of the cameras.

This virtual manipulation of the speed also allows testing with current. Current can be added, both when the cart is standing still, and moving. When the cart is still, every additional speed will be treated as current. When the cart is moving, adding a different speed than the actual cart speed will be treated as current. Using this method it is therefore possible to test the control system with a speed disturbance, similar to current.

4.2 Lab Testing

To test the planner, four test paths were planned and used as reference to the control system. For the first two, the path started in (0,0) and ended in (4,-1), but with different final heading, lower velocities, and accelerations. The two last paths are with obstacles defined for the planner to handle. Number three is a docking case and four is more of a transit problem, but the velocities are still low enough to use DP-controller for the maneuvering. The complete list of all parameters is shown in appendix D. For the three first paths, multiple tests were conducted, but only the best run is presented here. The results from all tests are shown in appendix E.

4.2.1 Lab Results

The two first paths are tested to assess the tuning of the planner in combination with the control system for CS3. The results are shown in figure 4.5 and 4.6. The difference between path one and two is the constraints on velocity and acceleration, and as a consequence of these differences, final heading angle. The results from the first test show that the model follows the reference well until it almost reaches the final position. At that time, the model moves very quickly away from the desired position. Two possible explanations from these results have been identified. One explanation is bad tuning of the control system and limits for the path planner. If these don't match, the control system will struggle to follow the reference trajectory. Another possible explanation is low voltage on the batteries. This is a known problem when testing in MCLab at the end of long days with testing. With low voltage, the thrusters struggle to produce the thrust commanded by the controller. This can lead to big errors in the controller. For the second path, the model follows the reference very well, and the model also manages to stop at almost the correct place. The problem for this test is the oscillations around the reference. This is a problem on all the states, but it is not so clear on the plot of x position. These oscillations can be a result of the constraints in the planner, but since similar oscillations are shown in the DP test in figure 4.3, it can also be a consequence of the tuning.

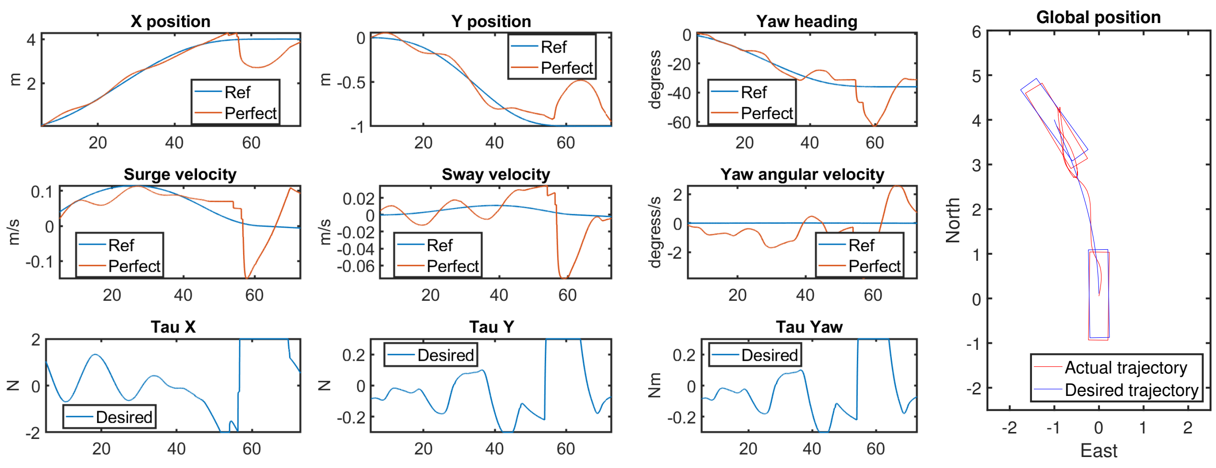


Figure 4.5: MCLab Path One

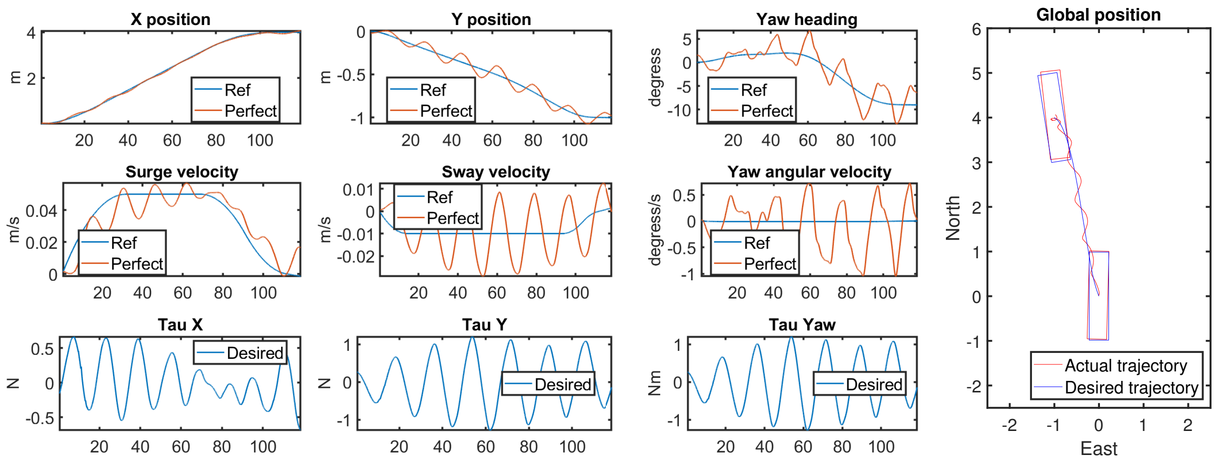


Figure 4.6: MCLab Path Two

The two last tests, 4.7 and 4.8, are similar to the two previous tests, but now the planner must handle obstacles. The paths will be almost the same as previously, but if the controller can't follow the reference, the model will collide with the obstacle. This seems to be the case in figure 4.7 since the vessel oscillates after it should stop. In figure 4.8 the oscillations are smaller, but the trajectory is planned with too little distance to the obstacle. As a consequence of this the vessel would have collided, even with perfect tracking and no oscillations. The results from both tests are similar to those from test two, shown in figure 4.6. The oscillations seem to be at almost the same frequency, but since the time horizon is different, there can be some differences that are difficult to see.

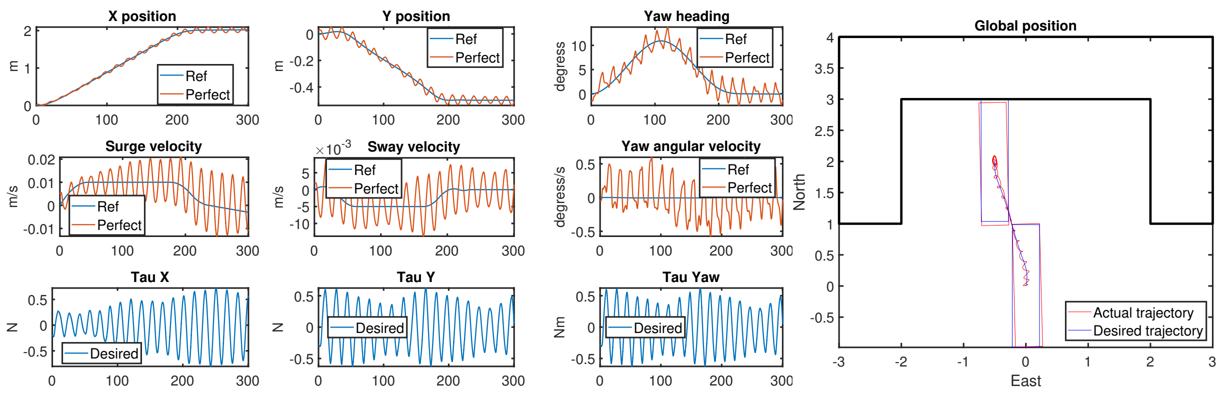


Figure 4.7: MCLab Path Three

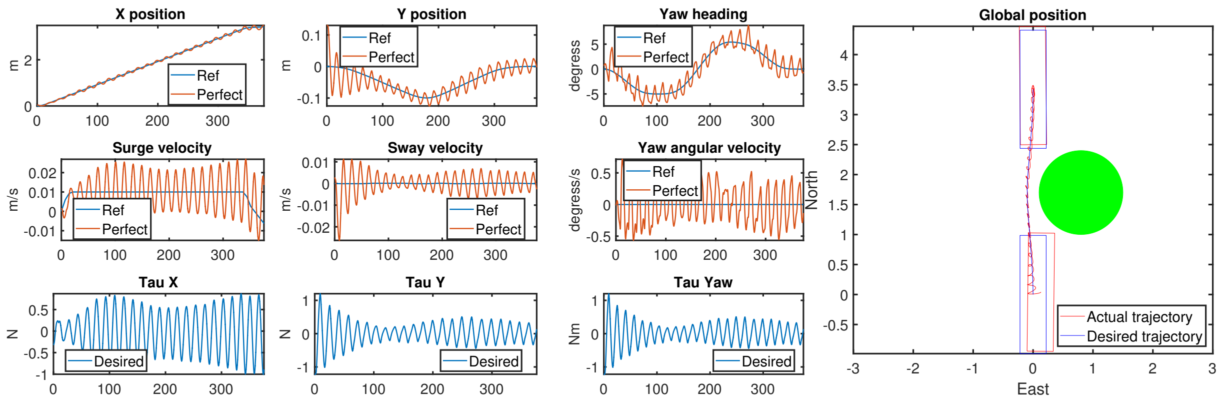


Figure 4.8: MCLab Path Four

The main result from the experimental testing is the same as testing with simulators. The reference produced by the planner can be followed with different control system. Similar to simulator testing, the control system will be as important for the ability to follow the reference as the planner. The clearest example of this is the DP-test in figure 4.3. The vessel oscillates as much in this test, as in any test using the planner.

Chapter 5

Concluding remarks

5.1 Conclusion

The goal of this master thesis was to continue the work from Johansen (2018) on path planning for auto docking of autonomous ferries. The focus in the previous project was on developing an environment for testing the planner. In this thesis, the main focus is on the actual planner and how to develop this.

One of the first decisions concerning the planner and simulation environment was sensors and sensor data. In this thesis, sensors can be divided into two types, sensors for positioning and sensors for monitoring the environment. Measurements on position are assumed to be accurate down to a few centimeters using Real Time Kinematics (RTK) and these can therefore be modeled using the actual measurements with white noise on top of these and continuous measurements. The sensors for environmental monitoring are in this thesis related to wind, waves and current. There are sensors available for monitoring this, offering both high accuracy and sampling frequency, that can be used for this. But, at least for waves and current, it is not normal to measure these today. It was therefore decided to not include updated real-time measurements as information for the planner, but some measurements could be available for the control system.

The next major decision was how the path planning should be solved. Multiple methods were investigated in section 1.3.3, but the preferred method was optimization based. This omitted the need for implementing a new guidance module in the simulators since the planner could output a full reference trajectory directly to the controller, and not a list of way-points. It also simplified the planner since it solved both the task of planning and optimizing in one step. Using a way-point method would give a list of way-points that later could be optimized.

The planner was implemented using a very simplified 3 DOF kinematic model as the basis of the optimization. This means that the planner can easily be used for multiple vessels by only changing constraints on states and control input. Since the planner now was tested on three different simulation models, this was an important point. For future planners made for specific vessels, including the kinetics will be more important and offer better performance since they will be more accurate. Using environmental disturbances will also be possible if using kinetics. Current can easily be added as only a speed disturbance, but wind and waves must be included as forces. Therefore, only current is used in the planner, and this is only included as an additional velocity in the state equations.

The planner is implemented using CasADi in Matlab. CasADi offers different methods for nonlinear modeling and optimization, but in this thesis, the Opti stack is used. This offers a simple method for setting up the optimization problem. The solver used for the actual optimization is IPOPT, an interior point optimization method. Using this framework gave good results for the optimization, but at this time, it is not very fast so it is difficult to use this in real time applications. Since the planner uses too much time at the current stage, and information about the weather is not updated very often, the planner in this thesis runs offline and plans the trajectory for the vessel to follow before it actually starts. It is possible to improve the time used for the optimization by only updating the problem and not defining it for each time-step, but at the current stage, this is hard to include in the simulations using Simulink. The plan is to use the planner to make different trajectories for a set of different current situations and then the ferry can select the trajectory planned for the current closest to the actual conditions.

A total of six planners were initially tested, but after comparing the results, only two were actually used further. These are both based on the Opti stack in CasADi and use the kinematic models in figures 2.2 and 2.3. The performance of these planners is very similar and the only real difference is in computation time. The planners both solve the task of planning a trajectory and optimizing the time for this trajectory. The planner can be used with multiple vessels, with only minor adjustments on the constraints for the optimization. This was tested in three simulators and CS3 in MCLab. The control system on all simulators and implemented in MCLab on CS3 can follow this trajectory, but testing shows clearly the importance of adjusting the constraints in the planner to the control system. If these are not adjusted to fit, the system will struggle to follow the planned trajectory. This was the case in both simulations and experimental tests where tuning the controller and thrust allocation was the most difficult part, regardless of the trajectory it should follow. The other main takeaway from the actual planners is related to the optimization. How the optimization problem is set up and defined will affect the performance of the solver, and can in some instances mean that the problem can’t be solved.

5.2 Further Work

The task in this thesis was to continue the work from Johansen (2018) and develop an algorithm for path planning in auto docking of autonomous ferries. An algorithm that can be used to solve this problem has been developed. But this planner still needs more work before it can be used in real applications. The first, and most important part, is related to the optimization used in the planner. When using CasADi, the performance of the optimization is linked to the size of the optimization parameters. Ensuring that all the optimization parameters are within a certain interval, for example between 0 and 100, and rather multiply the solution later with a known factor, can greatly improve the performance of the optimizer. It will also be necessary to further investigate what functions and expressions can be used in the optimization without too much negative effect on the performance, and if these functions are necessary, how to work around the challenges. An example of this is with current affecting only parts of the path. With the current version, the planner is supposed to optimize the time used on the trajectory, but when saying that half the map has current pushing the vessel towards the goal and the other half does not, it still chooses the part without current. This can be caused by the boolean expression in the differential equations, but it is necessary to spend more time to find the reason and solve the problem.

Another important part for future work is integrating the optimization better with Simulink to allow running online optimization of the solution. It is possible to define the problem offline and only update certain parameters and solve the optimization for each iteration. Simple testing using only Matlab shows that the time can then be decreased from 40 seconds when defining and solving the first time, to only a few seconds after updating the parameters and solving again. With possible improvements on computation time from defining the problem better and this improvement, it should be possible to run the optimization online. Another possible solution around this problem is using another simulation tool, but with all simulation models and the models from MCLab running in Simulink, this was not possible for now.

The model used in the planner should also be expanded to include waves and wind which will affect the vessel and the possible path. This will mean including much more vessel information in the planner, which for this thesis was unwanted, but it should improve the results to use vessel specific planners.

Another important challenge that is not addressed in this thesis is the problem of a moving reference trajectory. With the current setup, the planner will make a reference trajectory for the vessel to follow, but if the vessel for some reason can't follow, the reference would still continue. If then, for some reason, the vessel starts again it will rush to catch up with the reference, which can cause hazardous situations. A later version of the planner must handle this problem before it can be tested on real vessels.

Bibliography

Andersson, J. A. E., Gillis, J., Horn, G., Rawlings, J. B. & Diehl, M. (2018), 'CasADi – A software framework for nonlinear optimization and optimal control', *Mathematical Programming Computation* .

Aurenhammer, F. (1991), 'Voronoi diagrams - a survey of a fundamental geometric data structure', *ACM Comput. Surv.* **23**, 345–405.

Barraquand, J. & Latombe, J. . (1990), A monte-carlo algorithm for path planning with many degrees of freedom, *in* 'Proceedings., IEEE International Conference on Robotics and Automation', pp. 1712–1717 vol.3.

Bessiere, P., Ahuactzin, J. ., Talbi, E. . & Mazer, E. (1993), The "ariadne's clew" algorithm: global planning with local methods, *in* 'Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)', Vol. 2, pp. 1373–1380 vol.2.

Bitar, G., Breivik, M. & Lekkas, A. M. (2018), 'Energy-optimized path planning for autonomous ferries', *IFAC-PapersOnLine* **51**(29), 389 – 394. 11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2018.
URL: *http://www.sciencedirect.com/science/article/pii/S2405896318321451*

Bosch (2019). [Accessed 21 Jan. 2019].
URL: *https://www.bosch-mobility-solutions.com/en/products-and-services/passenger-cars-and-light-commercial-vehicles/driver-assistance-systems/predictive-emergency-braking-system/mid-range-radar-sensor-(mrr)/*

Brodtkorb, A. H., Nielsen, U. D. & Sørensen, A. J. (2018*a*), 'Online wave estimation using vessel motion measurements', *IFAC-PapersOnLine* **51**(29), 244 – 249. 11th IFAC Conference on Control Applications in Marine Systems, Robotics, and Vehicles CAMS 2018.
URL: *http://www.sciencedirect.com/science/article/pii/S2405896318321992*

Brodtkorb, A. H., Nielsen, U. D. & Sørensen, A. J. (2018*b*), 'Sea state estimation using vessel response in dynamic positioning', *Applied Ocean Research* **70**, 76 – 86.
URL: *http://www.sciencedirect.com/science/article/pii/S0141118717302481*

Bry, A. & Roy, N. (2011), Rapidly-exploring random belief trees for motion planning under uncertainty, *in* '2011 IEEE International Conference on Robotics and Automation', pp. 723–730.

CasADi (2016).
URL: *https://github.com/casadi/casadi/blob/3.1.0/docs/examples/matlab/direct_multiple_shooting.m*

CasADi (2017).
URL: *https://web.casadi.org/blog/ocp/coding*

Cokelet, E. D., Meinig, C., Lawrence-Slavas, N., Stabeno, P. J., Mordy, C. W., Tabisola, H. M., Jenkins, R. & Cross, J. N. (2015), The use of saildrones to examine spring conditions in the bering sea, *in* 'OCEANS 2015 - MTS/IEEE Washington', pp. 1–7.

Datawell BV (2019). [Accessed 18 Jan. 2019].
URL: *http://www.datawell.nl/Home.aspx*

Dijkstra, E. W. (1959), 'A note on two problems in connexion with graphs', *Numerische Mathematik* **1**(1), 269–271.
URL: *https://doi.org/10.1007/BF01386390*

Dolgov, D., Thrun, S., Montemerlo, M. & Diebel, J. (2010), 'Path planning for autonomous vehicles in unknown semi-structured environments', *The International Journal of Robotics Research* **29**(5), 485–501.
URL: *https://doi.org/10.1177/0278364909359210*

Fossen, T. I. & Perez, T. (2004), 'Marine systems simulator (mss)'.
URL: *https://github.com/cybergalactic/MSS*

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

Hernàndez, J. D. (2017), Online Path Planning for Autonomous Underwater Vehicles Under Motion Constraints, PhD thesis, Universitat de Girona. Departament d'Arquitectura i Tecnologia de Computadors.

Hernández, J. D., Vidal, E., Vallicrosa, G., Galceran, E. & Carreras, M. (2015), Online path planning for autonomous underwater vehicles in unknown environments, *in* '2015 IEEE International Conference on Robotics and Automation (ICRA)', pp. 1152–1157.

Hsu, D., Latombe, J. . & Motwani, R. (1997), Path planning in expansive configuration spaces, *in* 'Proceedings of International Conference on Robotics and Automation', Vol. 3, pp. 2719–2726 vol.3.

Idland, T. K. (2015), Marine cybernetics vessel cs saucer:-design, construction and control, Master's thesis, NTNU.

Johansen, T. (2018), Autonomous path planning for auto docking of ferries.
URL: *https://www.researchgate.net/publication/330360394*

Kamon, I., Rivlin, E. & Rimon, E. (1996), A new range-sensor based globally convergent navigation algorithm for mobile robots, *in* 'Proceedings of IEEE International Conference on Robotics and Automation', Vol. 1, pp. 429–435 vol.1.

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

Khatib, O. (1985), Real-time obstacle avoidance for manipulators and mobile robots, *in* 'Proceedings. 1985 IEEE International Conference on Robotics and Automation', Vol. 2, pp. 500–505.

Khatib, O. (1986), 'Real-time obstacle avoidance for manipulators and mobile robots', *The International Journal of Robotics Research* **5**(1), 90–98.
URL: *https://doi.org/10.1177/027836498600500106*

Kuffner, J. J. & LaValle, S. M. (2000), Rrt-connect: An efficient approach to single-query path planning, *in* 'Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)', Vol. 2, pp. 995–1001 vol.2.

Lavalle, S. M. (1998), Rapidly-exploring random trees: A new tool for path planning, Technical report, Department of Computer Science Iowa State University.

LaValle, S. M. & Kuffner, J. J. (1999), Randomized kinodynamic planning, *in* 'Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)', Vol. 1, pp. 473–479 vol.1.

Lekkas, A. M., Roald, A. L. & Breivik, M. (2016), 'Online path planning for surface vehicles exposed to unknown ocean currents using pseudospectral optimal control', *IFAC-PapersOnLine* **49**(23), 1 – 7. 10th IFAC Conference on Control Applications in Marine SystemsCAMS 2016.
URL: *http://www.sciencedirect.com/science/article/pii/S2405896316319000*

Lumelsky, V. J. & Stepanov, A. A. (1987), 'Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape', *Algorithmica* **2**(1), 403–430.
URL: *https://doi.org/10.1007/BF01840369*

Manley, J. E. (1997), Development of the autonomous surface craft "aces", *in* 'Oceans '97. MTS/IEEE Conference Proceedings', Vol. 2, pp. 827–832 vol.2.

Manley, J. E. (2008), Unmanned surface vehicles, 15 years of development, *in* 'OCEANS 2008', pp. 1–4.

Manley, J. E., Marsh, A., Cornforth, W. & Wiseman, C. (2000), Evolution of the autonomous surface craft autocat, *in* 'OCEANS 2000 MTS/IEEE Conference and Exhibition. Conference Proceedings (Cat. No.00CH37158)', Vol. 1, pp. 403–408 vol.1.

Mayne, D. Q., Rawlings, J. B., Rao, C. V. & Scokaert, P. O. (2000), 'Constrained model predictive control: Stability and optimality', *Automatica* **36**(6), 789–814.

Meinig, C., Lawrence-Slavas, N., Jenkins, R. & Tabisola, H. M. (2015), The use of saildrones to examine spring conditions in the bering sea: Vehicle specification and mission performance, *in* 'OCEANS 2015 - MTS/IEEE Washington', pp. 1–6.

Nocedal, J. & Wright, S. J. (2006), *Numerical optimization*, Springer series in operation research and financial engineering, 2nd ed. edn, Springer, New York.

NTNU (2017). [Accessed 24 Feb. 2019].
URL: *https://github.com/NTNU-MCS/MC_Lab_Handbook*

Pedersen, B., Pedersen, T., Klug, H., Van der Borg, N., Kelley, N. & Dahlberg, J. (1999), 'Wind speed measurement and use of cup anemometry', *Recommended Practices for Wind Turbine Testing and Evaluation* .

Richalet, J., Rault, A., Testud, J. & Papon, J. (1978), 'Model predictive heuristic control: Applications to industrial processes', *Automatica* **14**(5), 413 – 428.
URL: *http://www.sciencedirect.com/science/article/pii/0005109878900018*

Schiaretti, M., Chen, L. & Negenborn, R. R. (2017), Survey on autonomous surface vessels: Part i - a new detailed definition of autonomy levels, *in* T. Bektaş, S. Coniglio, A. Martinez-Sykora & S. Voß, eds, 'Computational Logistics', Springer International Publishing, Cham, pp. 219–233.

Spange, J. (2016), 'Autonomous docking for marine vessels using a lidar and proximity sensors'.

Stentz, A. (1994), Optimal and efficient path planning for partially-known environments, *in* 'Proceedings of the 1994 IEEE International Conference on Robotics and Automation', pp. 3310–3317 vol.4.

Stentz, A. T. (1995), The focussed d* algorithm for real-time replanning, *in* 'Proceedings of the International Joint Conference on Artificial Intelligence'.

Teknisk Ukeblad (2018*a*). [Accessed 27 Aug. 2018].
URL: *https://www.tu.no/artikler/ntnu-autonomi-er-en-stor-satsing-for-oss-vi-har-100-som-tar-doktorgraden/440081?key=kjHFpS99*

Teknisk Ukeblad (2018*b*). [Accessed 17 Des. 2018].
URL: *https://www.tu.no/artikler/verdens-forste-helt-autonome-fergeseilas-gjennomfort-teknologien-er-100-prosent-klar/452610*

Teknisk ukeblad (2018*c*). [Accessed 17 Des. 2018].
URL: *https://www.tu.no/artikler/na-kan-folgefonn-seile-automatisk-fra-havn-til-havn-og-legge-til-selv/452355*

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A. & Mahoney, P. (2006), 'Stanley: The robot that won the darpa grand challenge', *Journal of Field Robotics* **23**(9), 661–692.

VANECK, T. W., RODRIGUEZ-ORTIZ, C. D., SCHMIDT, M. C. & MANLEY, J. E. (1996), 'Automated bathymetry using an autonomous surface craft', *Navigation* **43**(4), 407–419.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2161-4296.1996.tb01929.x*

Wächter, A. (2009), Short tutorial: getting started with ipopt in 90 minutes, *in* 'Dagstuhl Seminar Proceedings', Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Zhu, M. & Wen, Y.-Q. (2019), 'Design and analysis of collaborative unmanned surface-aerial vehicle cruise systems', *Journal of Advanced Transportation* **2019**, 1–10.

Appendix A

Planner Test Parameters

Test 1

Test 1.1 and 1.4			
Initial North Position	0	Heading max	90°
Initial East Position	0	Heading min	$-$ 90°
Initial Speed	0	Speed max	5 m/s
Initial Heading Angle	0	Speed min	$-$ 5 m/s
Final North Position	800	Heading speed max	0.55°$/s$
Final East Position	1000	Heading speed min	$-$ 0.55°$/s$
Final Speed	0	Acceleration max	0.05 m/s^2
Final Heading Angle	90°	Acceleration min	$-$ 0.05 m/s^2
Jerk max[1]	0.0005 m/s^3	Heading acceleration max[1]	0.02°$/s^2$

([1] $-$ *Only Opti stack*)

Table A.1: Description Test Scenario One

Test 1.2 and 1.5			
Initial North Position	0	Heading max	90°
Initial East Position	0	Heading min	$-$ 90°
Initial Speed	0	Speed max	5 m/s
Initial Heading Angle	0	Speed min	$-$ 5 m/s
Initial Heading Speed	0	Heading speed max	0.55°$/s$
Final North Position	800	Heading speed min	$-$ 0.55°$/s$
Final East Position	1000	Acceleration max	0.05 m/s^2
Final Speed	0	Acceleration min	$-$ 0.05 m/s^2
Final Heading Angle	90°	Heading acceleration max	0.02°$/s^2$
Final Heading Speed	0	Heading acceleration min	$-$ 0.02°$/s^2$
Jerk max[1]	0.0005 m/s^3	Angular jerk max[1]	0.0003°$/s^3$

([1] – *Only Opti stack*)

Table A.2: Description Test Scenario One

Test 1.3 and 1.6			
Initial North Position	0	Heading max	90°
Initial East Position	0	Heading min	$-$ 90°
Initial Heading Angle	0	Surge max	5 m/s
Initial Surge	0	Surge min	$-$ 5 m/s
Initial Sway	0	Sway max	0.01 m/s
Initial Yaw	0	Sway min	$-$ 0.01 m/s
Final North Position	800	Yaw max	0.55°$/s$
Final East Position	1000	Yaw min	$-$ 0.55°$/s$
Final Heading Angle	90°	Surge acceleration max	0.05 m/s^2
Final Surge	0	Surge acceleration min	$-$ 0.05 m/s^2
Final Sway	0	Sway acceleration max	0.001 m/s^2
Final Yaw	0	Sway acceleration min	$-$ 0.001 m/s^2
Yaw acceleration min	$-$ 0.02°$/s^2$	Yaw acceleration max	0.02°$/s^2$
Surge jerk max[1]	0.0005 m/s^3	Sway jerk max[1]	0.00005 m/s^3
Yaw jerk max[1]	0.0003°$/s^3$		

([1] – *Only Opti stack*)

Table A.3: Description Test Scenario One

Test 2

Test 2.1 and 2.4			
Initial North Position	0	Heading max	90°
Initial East Position	0	Heading min	$-$ 90°
Initial Speed	0	Speed max	5 m/s
Initial Heading Angle	0	Speed min	$-$ 5 m/s
Final North Position	800	Heading speed max	0.55°$/s$
Final East Position	1000	Heading speed min	$-$ 0.55°$/s$
Final Speed	0	Acceleration max	0.05 m/s^2
Final Heading Angle	90°	Acceleration min	$-$ 0.05 m/s^2
Jerk max[1]	0.0005 m/s^3	Heading acceleration max[1]	0.02°$/s^2$

([1] $-$ *Only Opti stack*)

Table A.4: Description Test Scenario Two

Test 2.2 and 2.5			
Initial North Position	0	Heading max	90°
Initial East Position	0	Heading min	$-$ 90°
Initial Speed	0	Speed max	5 m/s
Initial Heading Angle	0	Speed min	$-$ 5 m/s
Initial Heading Speed	0	Heading speed max	0.55°$/s$
Final North Position	800	Heading speed min	$-$ 0.55°$/s$
Final East Position	1000	Acceleration max	0.05 m/s^2
Final Speed	0	Acceleration min	$-$ 0.05 m/s^2
Final Heading Angle	90°	Heading acceleration max	0.02°$/s^2$
Final Heading Speed	0	Heading acceleration min	$-$ 0.02°$/s^2$
Jerk max[1]	0.0005 m/s^3	Angular jerk max[1]	0.0003°$/s^3$

([1] $-$ *Only Opti stack*)

Table A.5: Description Test Scenario Two

Test 2.3 and 2.6			
Initial North Position	0	Heading max	90°
Initial East Position	0	Heading min	$-$ 90°
Initial Heading Angle	0	Surge max	5 m/s
Initial Surge	0	Surge min	$-$ 5 m/s
Initial Sway	0	Sway max	0.01 m/s
Initial Yaw	0	Sway min	$-$ 0.01 m/s
Final North Position	800	Yaw max	0.55°$/s$
Final East Position	1000	Yaw min	$-$ 0.55°$/s$
Final Heading Angle	90°	Surge acceleration max	0.05 m/s^2
Final Surge	0	Surge acceleration min	$-$ 0.05 m/s^2
Final Sway	0	Sway acceleration max	0.001 m/s^2
Final Yaw	0	Sway acceleration min	$-$ 0.001 m/s^2
Yaw acceleration min	$-$ 0.02°$/s^2$	Yaw acceleration max	0.02°$/s^2$
Surge jerk max^1	0.0005 m/s^3	Sway jerk max^1	0.00005 m/s^3
Yaw jerk max^1	0.0003°$/s^3$		

(^{1} – *Only Opti stack*)

Table A.6: Description Test Scenario Two

Test 3

Test 3.1 and 3.4			
Initial North Position	0	Heading max	90°
Initial East Position	0	Heading min	$-$ 90°
Initial Speed	1.5	Speed max	1.5 m/s
Initial Heading Angle	0	Speed min	0 m/s
Final North Position	240	Heading speed max	0.55°$/s$
Final East Position	90	Heading speed min	$-$ 0.55°$/s$
Final Speed	0	Acceleration max	0.05 m/s^2
Final Heading Angle	90°	Acceleration min	$-$ 0.05 m/s^2
Jerk max^1	0.0005 m/s^3	Heading acceleration max^1	0.02°$/s^2$

(^{1} – *Only Opti stack*)

Table A.7: Description Test Scenario Three

Test 3.2 and 3.5			
Initial North Position	0	Heading max	$90°$
Initial East Position	0	Heading min	$-90°$
Initial Speed	1.5	Speed max	$1.5\ m/s$
Initial Heading Angle	0	Speed min	$0\ m/s$
Initial Heading Speed	0	Heading speed max	$0.55°/s$
Final North Position	240	Heading speed min	$-0.55°/s$
Final East Position	90	Acceleration max	$0.05\ m/s^2$
Final Speed	0	Acceleration min	$-0.05\ m/s^2$
Final Heading Angle	$90°$	Heading acceleration max	$0.02°/s^2$
Final Heading Speed	0	Heading acceleration min	$-0.02°/s^2$
Jerk max[1]	$0.0005\ m/s^3$	Angular jerk max[1]	$0.0003°/s^3$

([1] – *Only Opti stack*)

Table A.8: Description Test Scenario Three

Test 3.3 and 3.6			
Initial North Position	0	Heading max	$90°$
Initial East Position	0	Heading min	$-90°$
Initial Heading Angle	0	Surge max	$1.5\ m/s$
Initial Surge	1.5	Surge min	$0\ m/s$
Initial Sway	0	Sway max	$0.01\ m/s$
Initial Yaw	0	Sway min	$-0.01\ m/s$
Final North Position	240	Yaw max	$0.55°/s$
Final East Position	90	Yaw min	$-0.55°/s$
Final Heading Angle	$90°$	Surge acceleration max	$0.05\ m/s^2$
Final Surge	0	Surge acceleration min	$-0.05\ m/s^2$
Final Sway	0	Sway acceleration max	$0.001\ m/s^2$
Final Yaw	0	Sway acceleration min	$-0.001\ m/s^2$
Yaw acceleration min	$-0.02°/s^2$	Yaw acceleration max	$0.02°/s^2$
Surge jerk max[1]	$0.0005\ m/s^3$	Sway jerk max[1]	$0.00005\ m/s^3$
Yaw jerk max[1]	$0.0003°/s^3$		

([1] – *Only Opti stack*)

Table A.9: Description Test Scenario Three

Appendix B

Planner Test Results

Test One

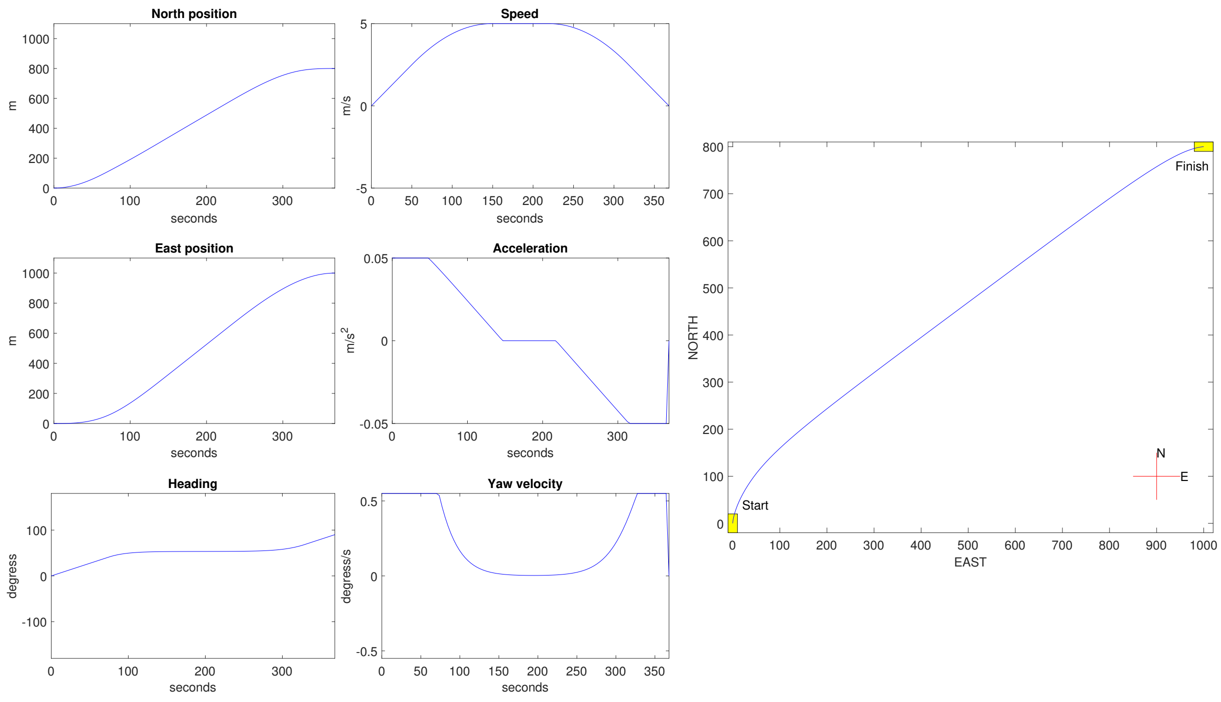


Figure B.1: Test 1.1

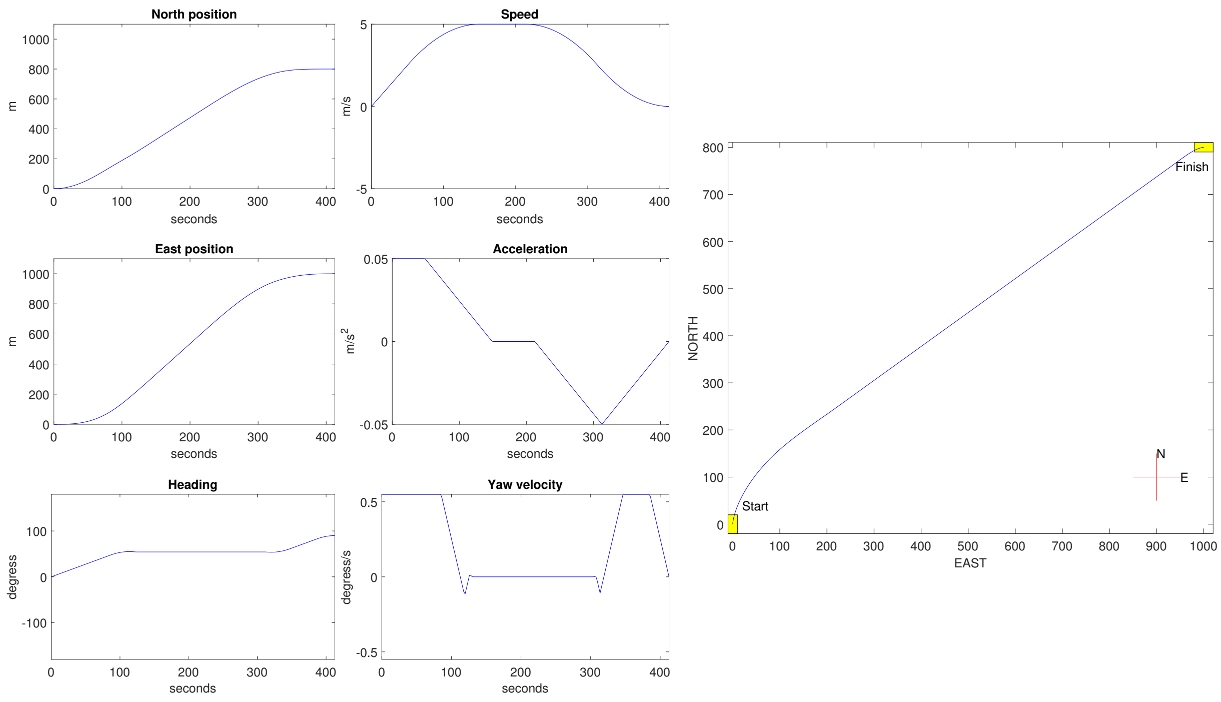


Figure B.2: Test 1.4

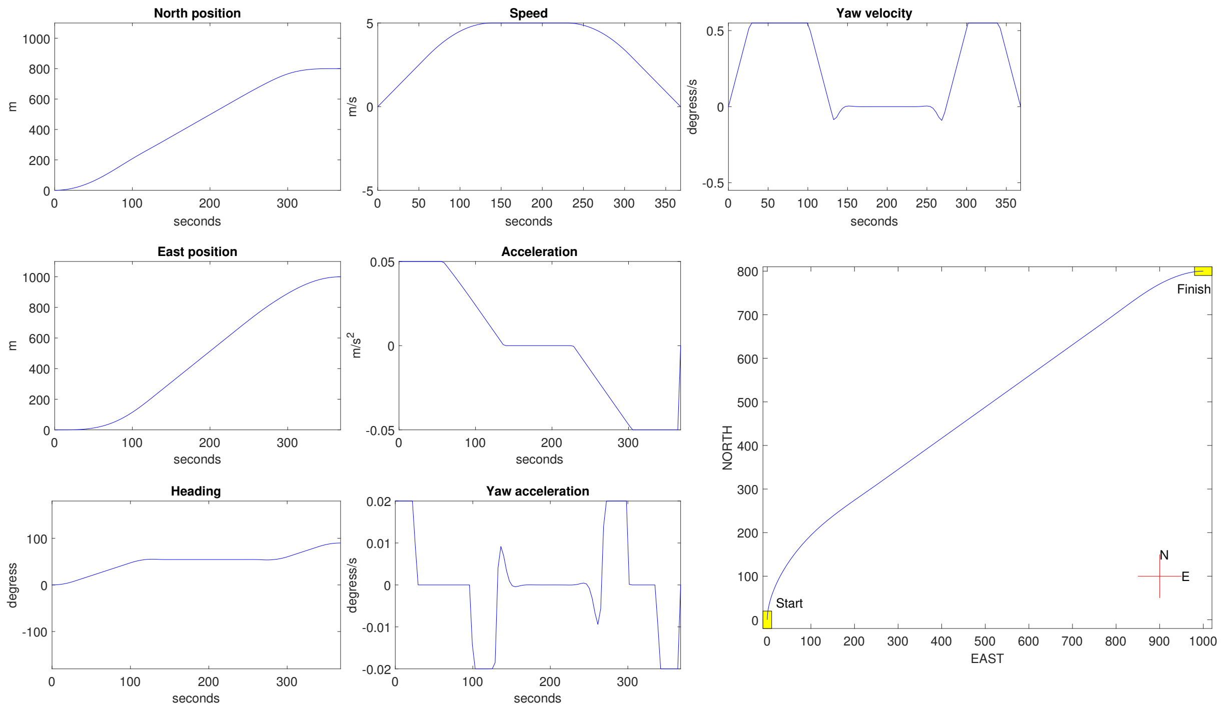


Figure B.3: Test 1.2

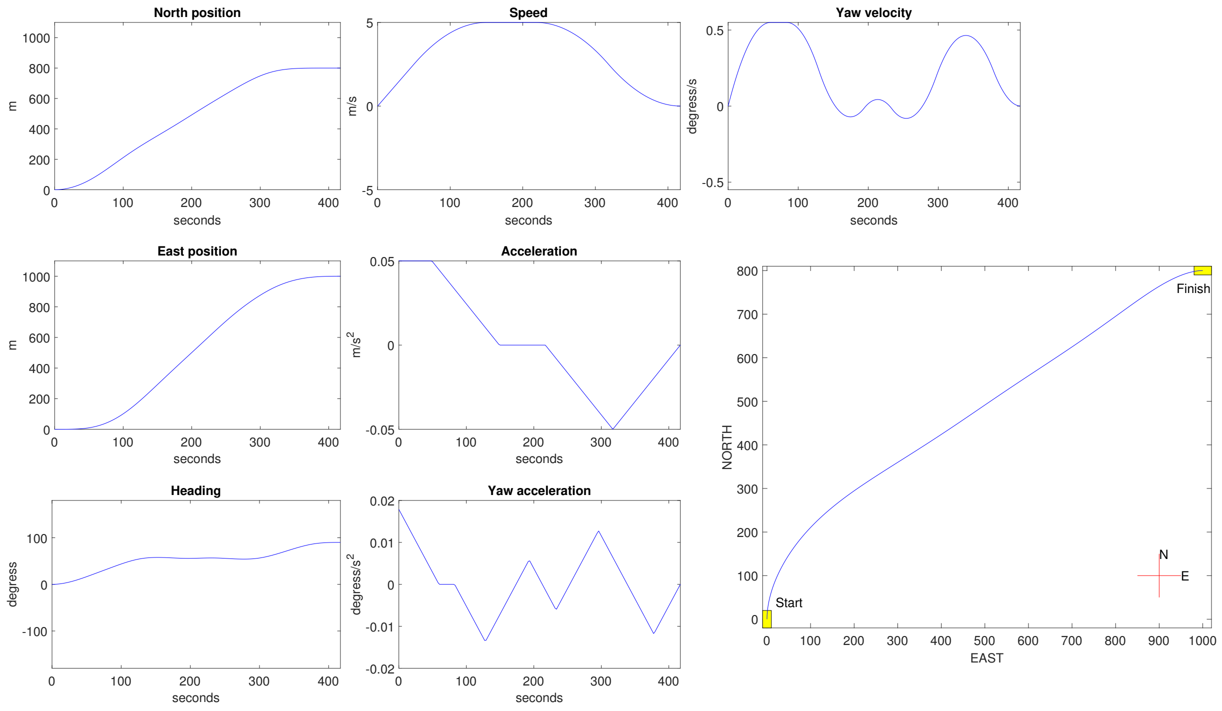


Figure B.4: Test 1.5

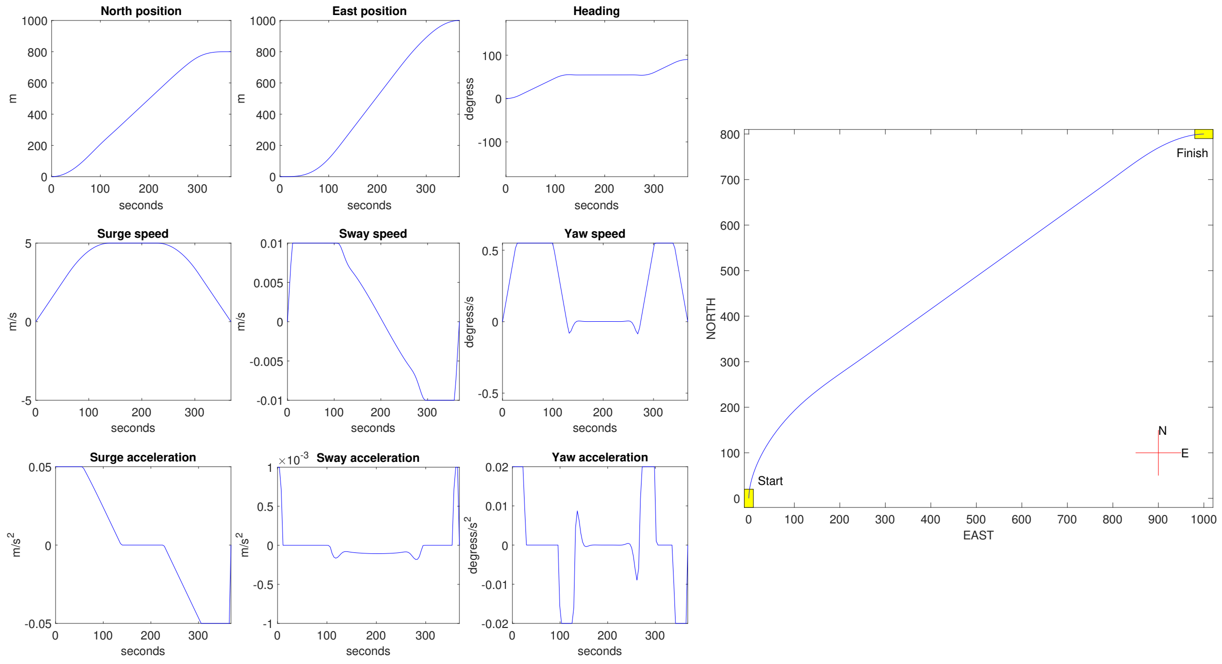


Figure B.5: Test 1.3

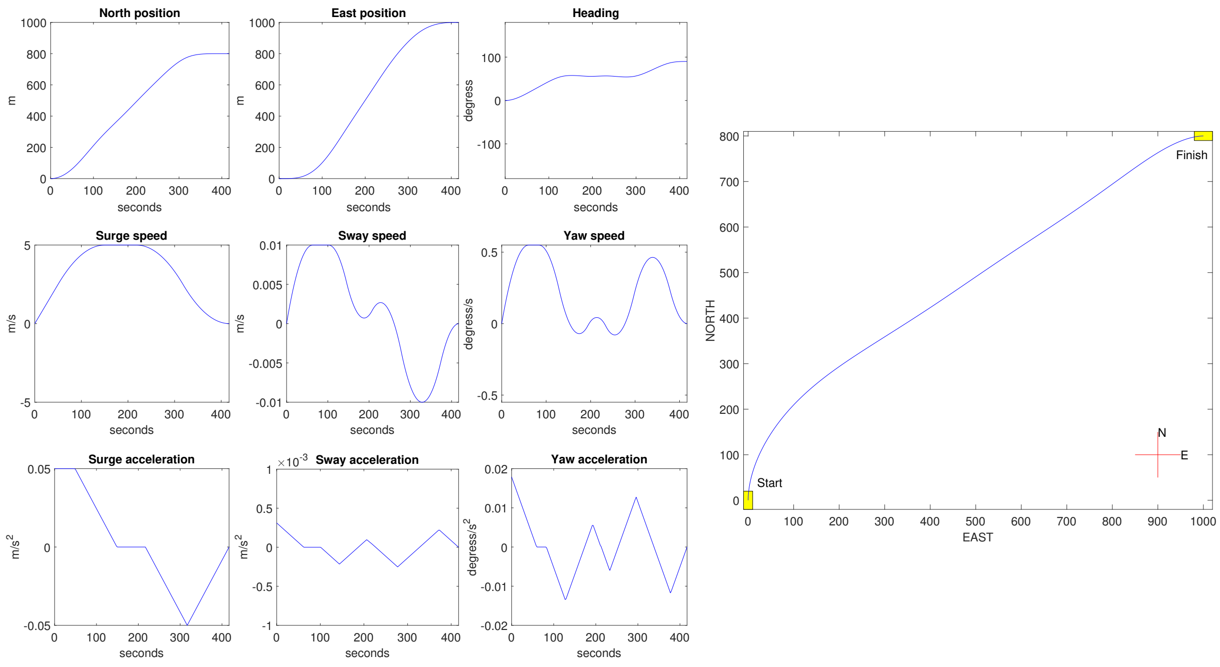


Figure B.6: Test 1.6

Test Two

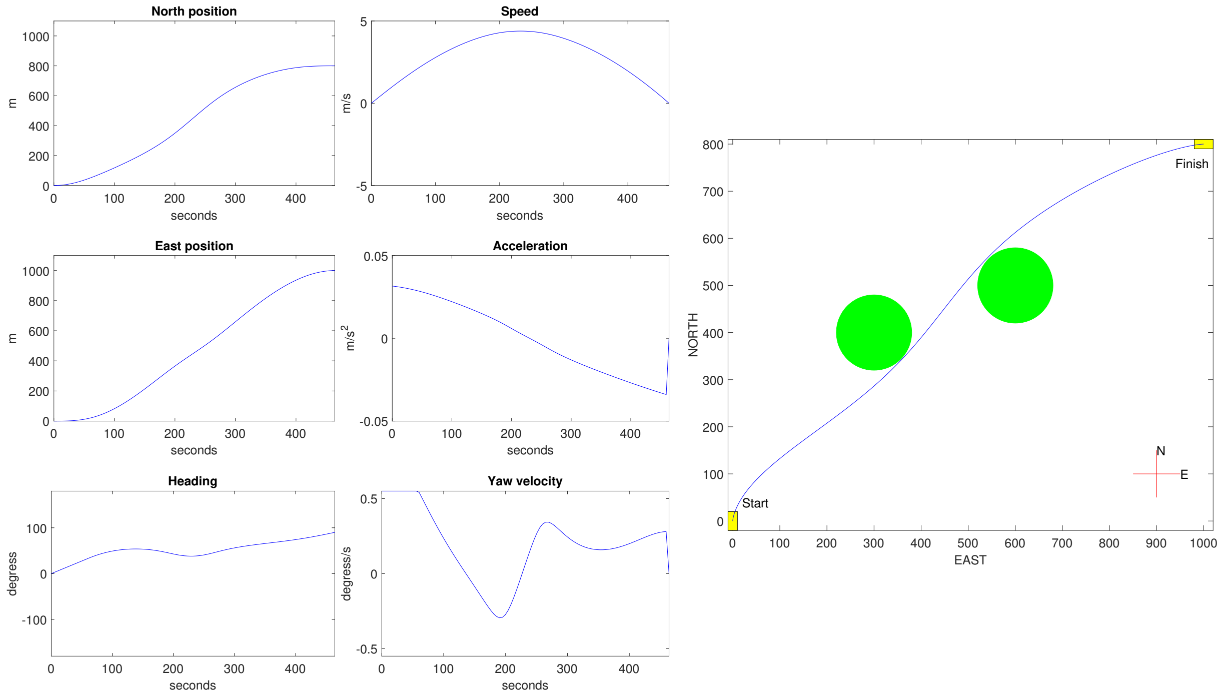


Figure B.7: Test 2.1

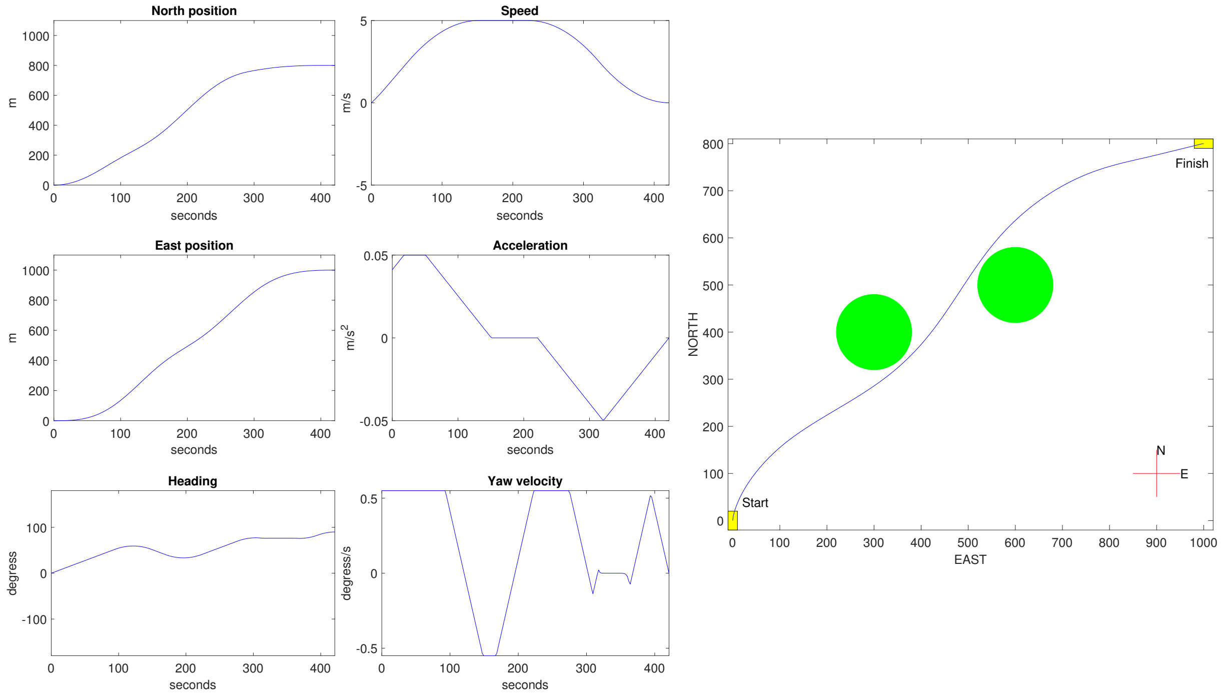


Figure B.8: Test 2.4

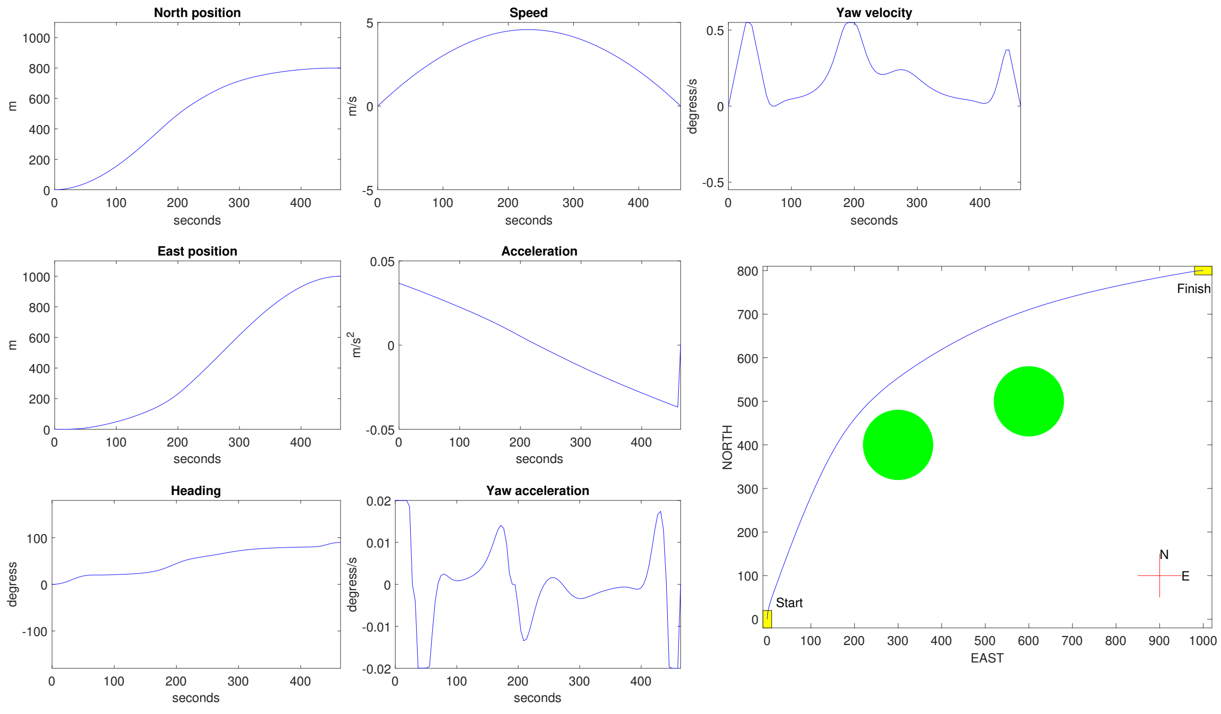


Figure B.9: Test 2.2

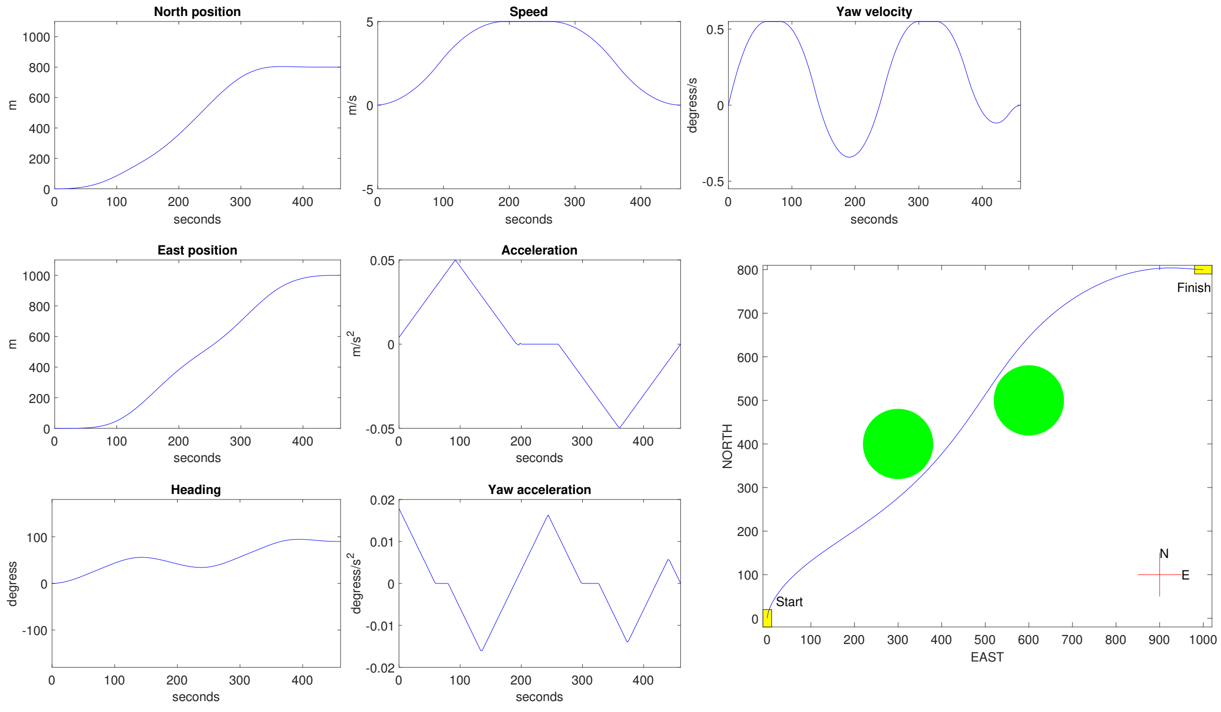


Figure B.10: Test 2.5

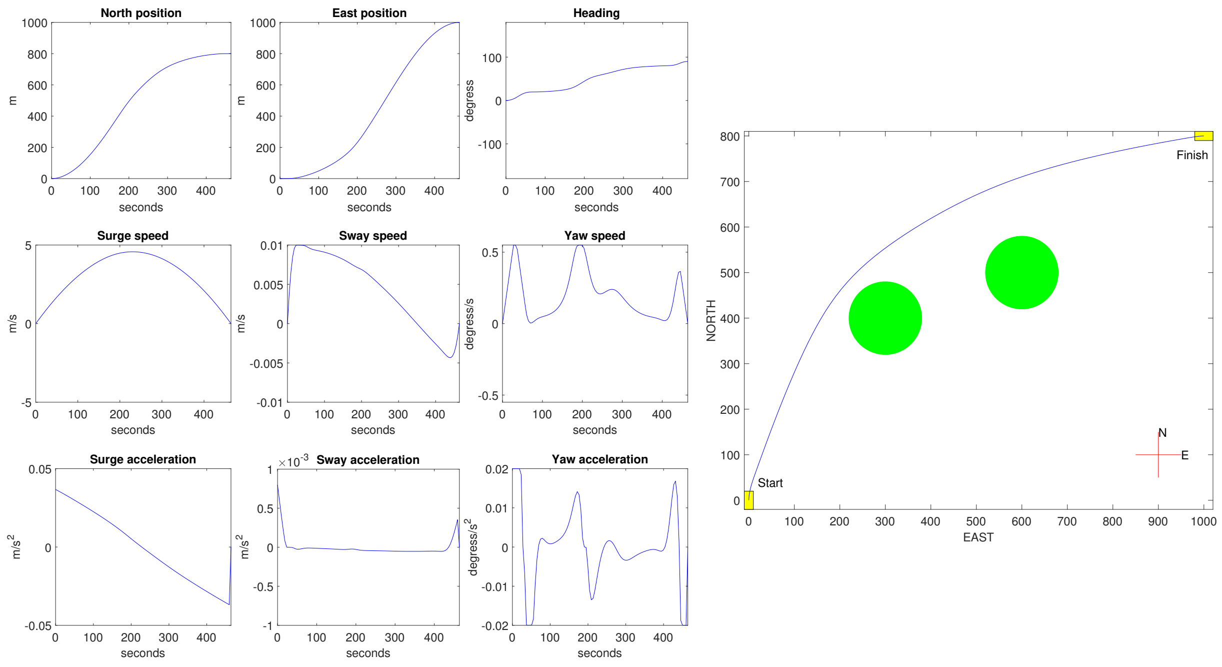


Figure B.11: Test 2.3

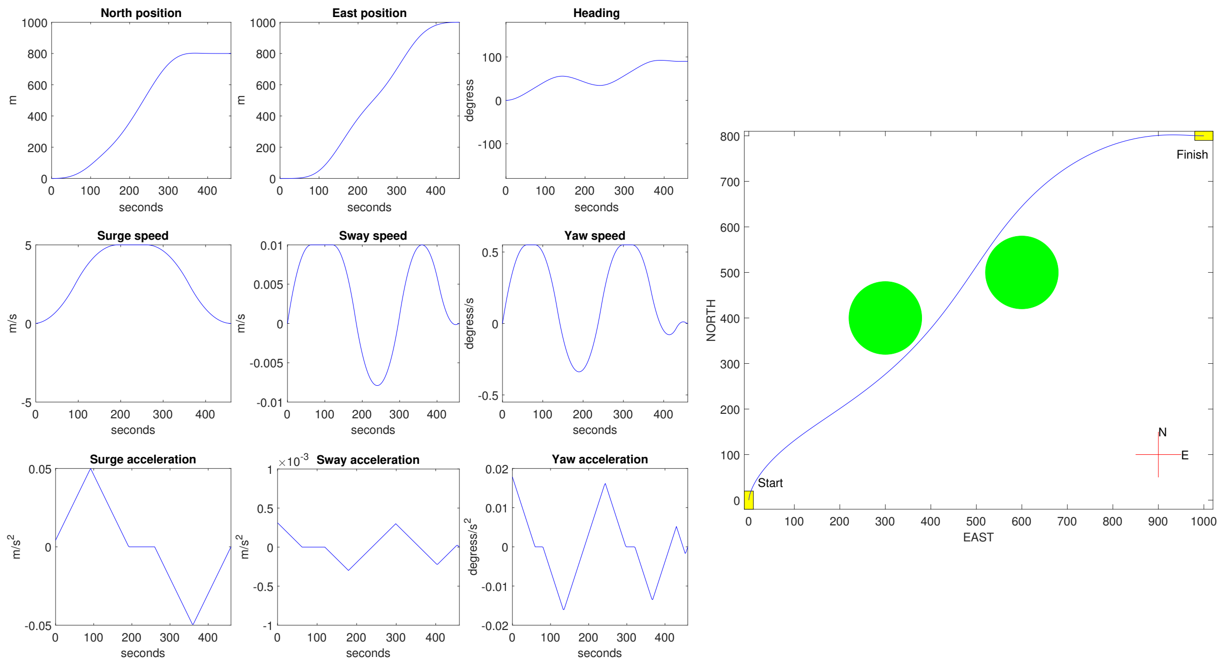
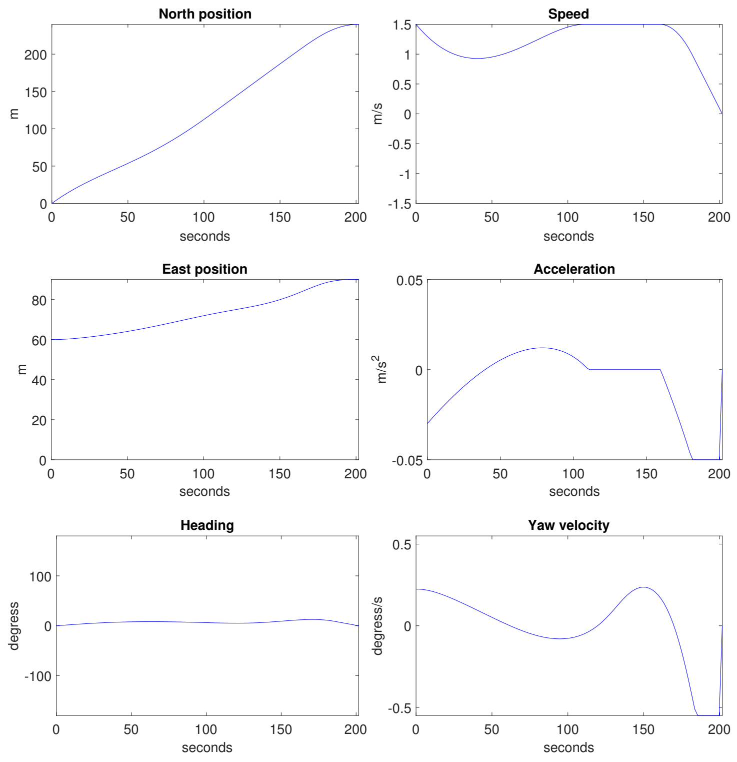


Figure B.12: Test 2.6

Test Three



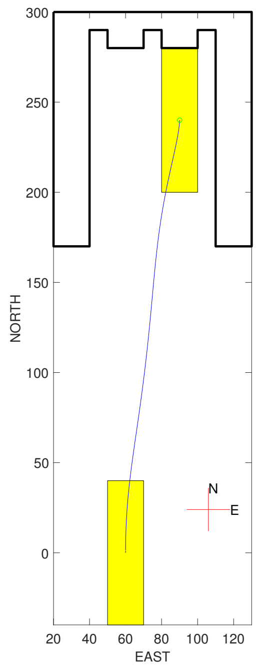


Figure B.13: Test 3.1

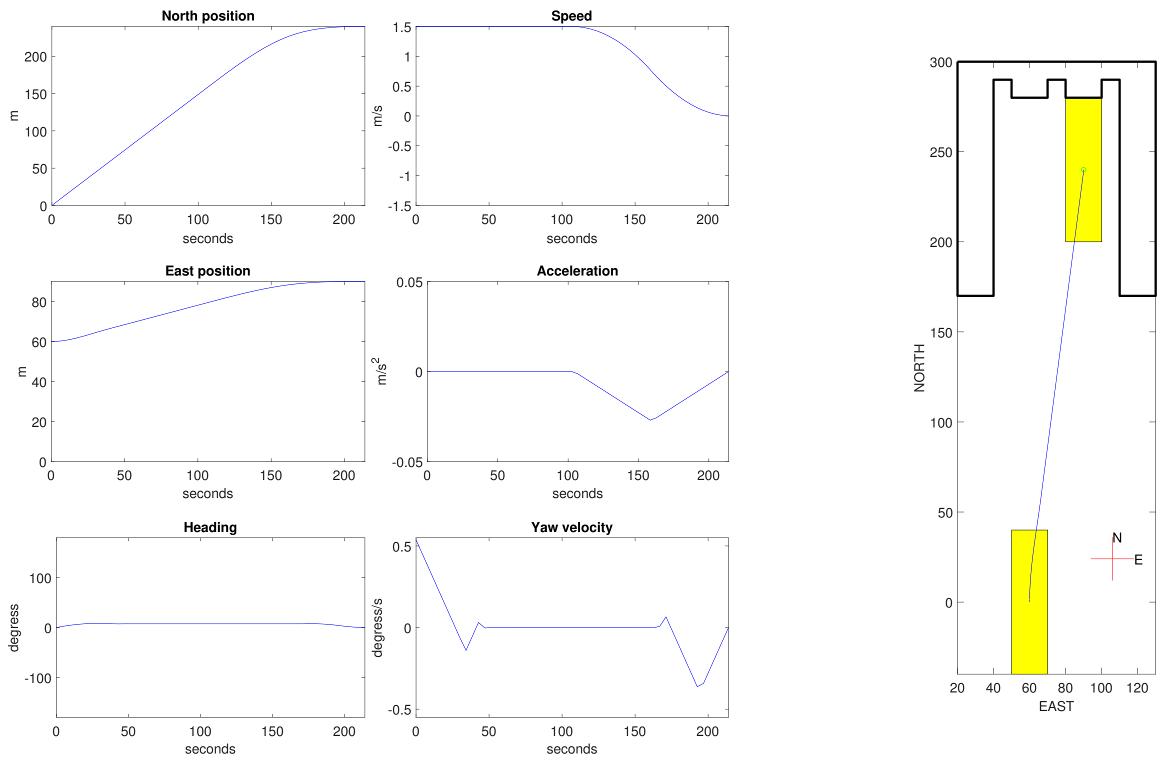


Figure B.14: Test 3.4

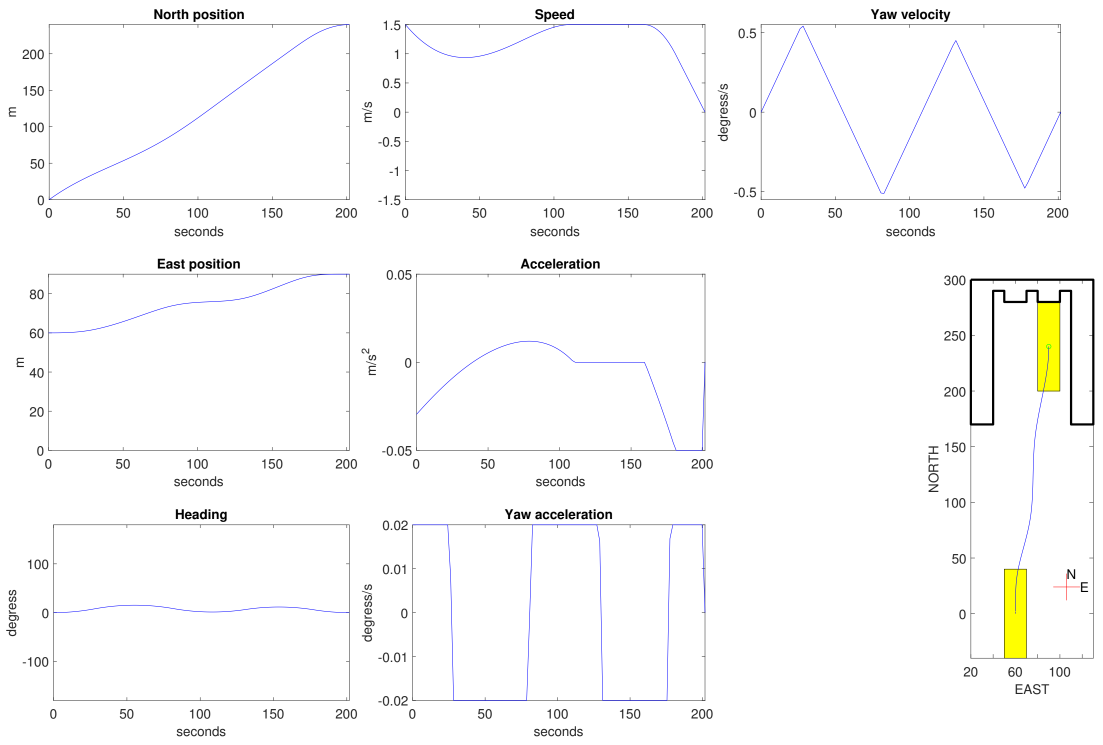


Figure B.15: Test 3.2

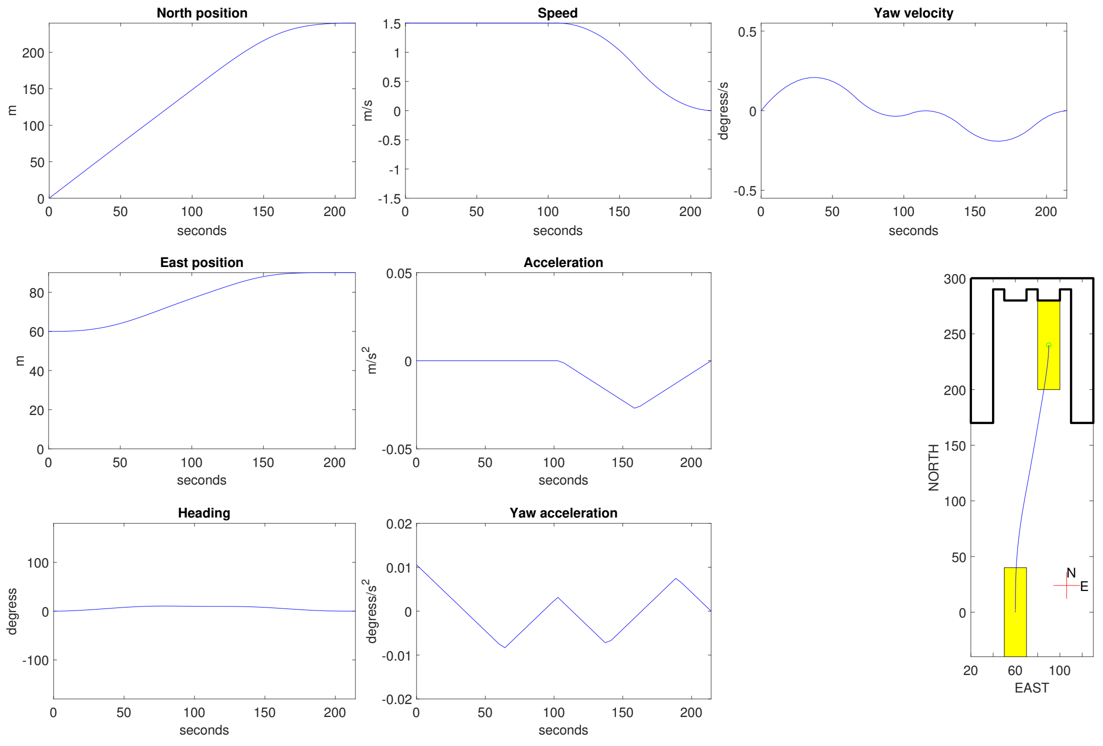


Figure B.16: Test 3.5

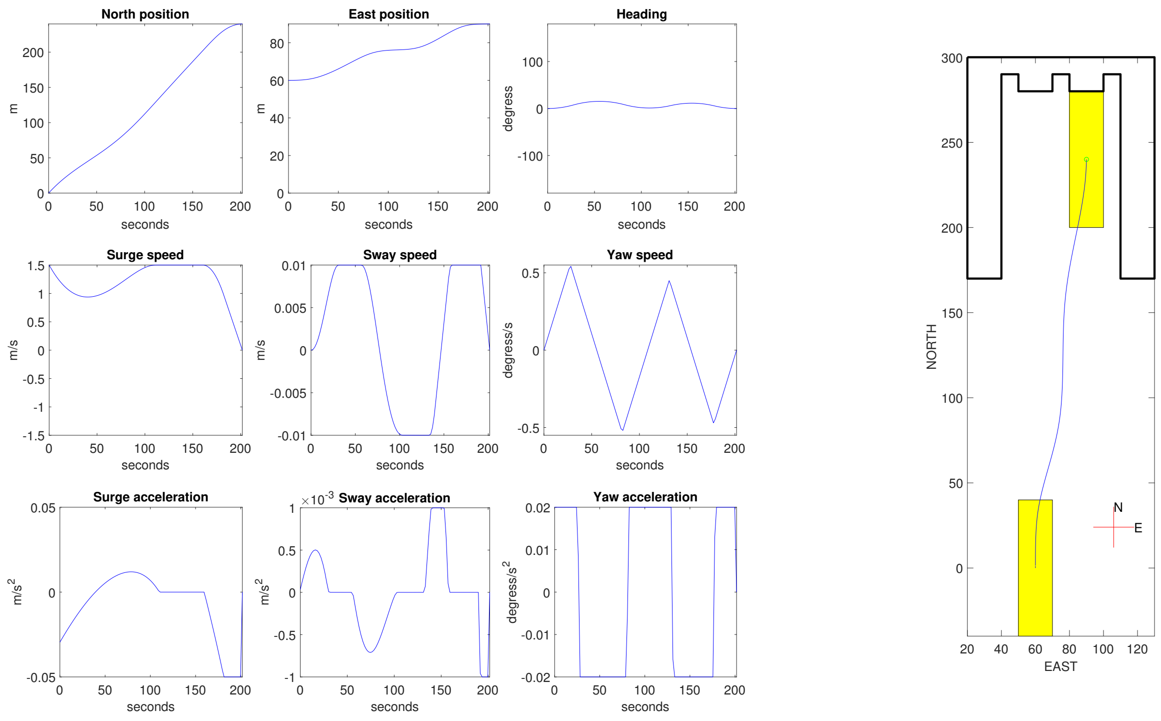
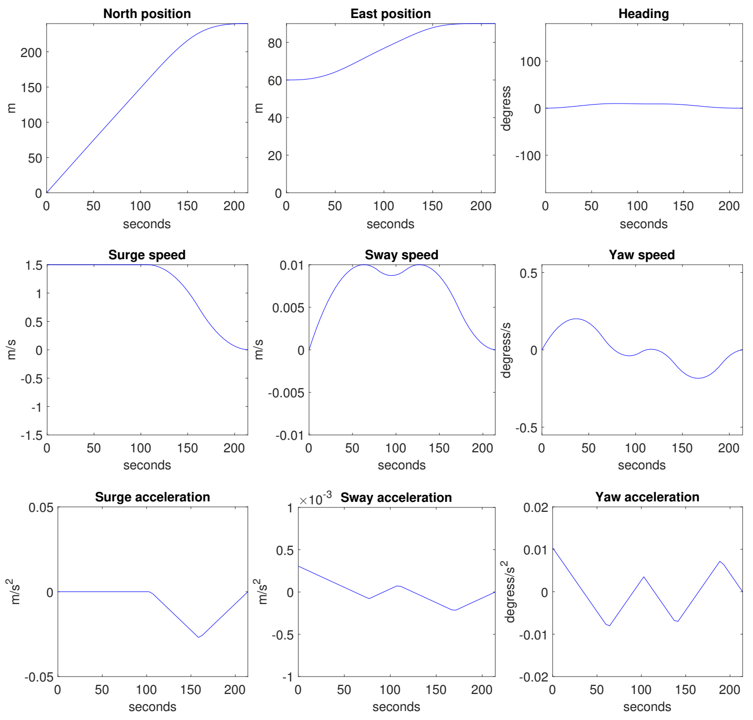


Figure B.17: Test 3.3



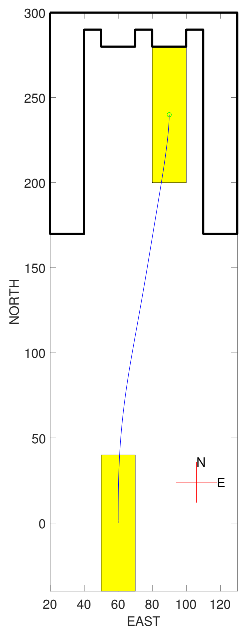


Figure B.18: Test 3.6

Appendix C

Parameters for Simulator Testing

Ferry Testing

Test 1

Planner Constraints		
u	$-\ 1.5\ m/s$	$1.5\ m/s$
v	$-\ 0.01\ m/s$	$0.01\ m/s$
w	$-\ 0.55\ degrees/s$	$0.55\ degrees/s$
a_u	$-\ 0.05\ m/s^2$	$0.05\ m/s^2$
a_v	$-\ 0.001\ m/s^2$	$0.001\ m/s^2$
a_w	$-\ 0.02\ degrees/s^2$	$0.02\ degrees/s^2$
j_u	$-\ 0.0005\ m/s^3$	$0.0005\ m/s^3$
j_v	$-\ 0.00005\ m/s^3$	$0.00005\ m/s^3$
j_w	$-\ 0.0003\ degrees/s^3$	$0.0003\ degrees/s^3$

Table C.1: Planner Constraints Test One Ferry Model

Control Parameters	
$K_{P_{Surge}}$	10^5
$K_{P_{Sway}}$	$4 \cdot 10^5$
$K_{P_{Yaw}}$	$4 \cdot 10^7$
$K_{D_{Surge}}$	10^6
$K_{D_{Sway}}$	$2 \cdot 10^7$
$K_{D_{Yaw}}$	$2 \cdot 10^7$
$K_{I_{Surge}}$	10^3
$K_{I_{Sway}}$	10^3
$K_{I_{Yaw}}$	10^3

Table C.2: Control Parameters Test One Ferry Model

Test 2

Planner Constraints		
U	$-\ 4\ m/s$	$4\ m/s$
w	$-\ 0.4\ degrees/s$	$0.4\ degrees/s$
a_U	$0\ m/s^2$	$0.05\ m/s^2$
a_w	$-\ 1\ degrees/s^2$	$1\ degrees/s^2$
j_U	$-\ 0.0005\ m/s^3$	$0.0005\ m/s^3$
j_w	$-\ 0.0003\ degrees/s^3$	$0.0003\ degrees/^3$

Table C.3: Planner Constraints Test Two Ferry Model

Control Parameters	
$K_{P_{Heading}}$	10^8
$K_{D_{Heading}}$	$5 \cdot 10^5$
$K_{P_{Sideslip}}$	$2 \cdot 10^{-2}$
$K_{I_{Sideslip}}$	$3 \cdot 10^{-4}$
$K_{P_{Speed}}$	10^6
$K_{I_{Speed}}$	10^3

Table C.4: Control Parameters Test Two Ferry Model

Test 3

Planner Constraints		
U	$-\ 5\ m/s$	$5\ m/s$
w	$-\ 0.4\ degrees/s$	$0.4\ degrees/s$
a_U	$0\ m/s^2$	$0.05\ m/s^2$
a_w	$-\ 1\ degrees/s^2$	$1\ degrees/s^2$
j_U	$-\ 0.0005\ m/s^3$	$0.0005\ m/s^3$
j_w	$-\ 0.0003\ degrees/s^3$	$0.0003\ degrees/^3$

Table C.5: Planner Constraints Test Three Ferry Model

Control Parameters	
$K_{P_{Heading}}$	10^8
$K_{D_{Heading}}$	$5 \cdot 10^5$
$K_{P_{Sideslip}}$	$2 \cdot 10^{-2}$
$K_{I_{Sideslip}}$	$3 \cdot 10^{-4}$
$K_{P_{Speed}}$	10^6
$K_{I_{Speed}}$	10^3

Table C.6: Control Parameters Test Three Ferry Model

R/V Gunnerus Testing

Test 1

Planner Constraints		
u	$-\ 0.7\ m/s$	$0.7\ m/s$
v	$-\ 0.01\ m/s$	$0.01\ m/s$
w	$-\ 0.2\ degrees/s$	$0.2\ degrees/s$
a_u	$-\ 0.02\ m/s^2$	$0.02\ m/s^2$
a_v	$-\ 0.001\ m/s^2$	$0.001\ m/s^2$
a_w	$-\ 0.01\ degrees/s^2$	$0.01\ degrees/s^2$
j_u	$-\ 0.00005\ m/s^3$	$0.00005\ m/s^3$
j_v	$-\ 0.00005\ m/s^3$	$0.00005\ m/s^3$
j_w	$-\ 0.00003\ degrees/s^3$	$0.00003\ degrees/s^3$

Table C.7: Planner Constraints Test One R/V Gunnerus

Control Parameters	
$K_{P_{Surge}}$	$4 \cdot 10^4$
$K_{P_{Sway}}$	$5 \cdot 10^5$
$K_{P_{Yaw}}$	$4 \cdot 10^6$
$K_{D_{Surge}}$	$3 \cdot 10^6$
$K_{D_{Sway}}$	10^5
$K_{D_{Yaw}}$	10^5
$K_{I_{Surge}}$	10^2
$K_{I_{Sway}}$	10^3
$K_{I_{Yaw}}$	10^3

Table C.8: Control Parameters Test One R/V Gunnerus

Test 2

Planner Constraints		
U	$-\ 4\ m/s$	$4\ m/s$
w	$-\ 0.55\ degrees/s$	$0.55\ degrees/s$
a_U	$0\ m/s^2$	$0.1\ m/s^2$
a_w	$-\ 1\ degrees/s^2$	$1\ degrees/s^2$
j_U	$-\ 0.0005\ m/s^3$	$0.0005\ m/s^3$
j_w	$-\ 0.0003\ degrees/s^3$	$0.0003\ degrees/^3$

Table C.9: Planner Constraints Test Two R/V Gunnerus

Control Parameters	
$K_{P_{Heading}}$	10^7
$K_{D_{Heading}}$	10^5
$K_{P_{Sideslip}}$	10^{-2}
$K_{I_{Sideslip}}$	0
$K_{P_{Speed}}$	10^6
$K_{I_{Speed}}$	10^7

Table C.10: Control Parameters Test Two R/V Gunnerus

MCSim Testing

Test 1

Planner Constraints		
u	$-$ 0.15 m/s	0.15 m/s
v	$-$ 0.05 m/s	0.05 m/s
w	$-$ 3 $degrees/s$	3 $degrees/s$
a_u	$-$ 0.1 m/s^2	0.1 m/s^2
a_v	$-$ 0.1 m/s^2	0.1 m/s^2
a_w	$-$ 10 $degrees/s^2$	10 $degrees/s^2$
j_u	$-$ 0.0005 m/s^3	0.0005 m/s^3
j_v	$-$ 0.0005 m/s^3	0.0005 m/s^3
j_w	$-$ 0.003 $degrees/s^3$	0.003 $degrees/s^3$

Table C.11: Planner Constraints Test One MCSim

Control Parameters	
$K_{P_{Surge}}$	10
$K_{P_{Sway}}$	20
$K_{P_{Yaw}}$	30
$K_{D_{Surge}}$	80
$K_{D_{Sway}}$	80
$K_{D_{Yaw}}$	80
$K_{I_{Surge}}$	5
$K_{I_{Sway}}$	1
$K_{I_{Yaw}}$	10

Table C.12: Control Parameters Test One MCSim

Test 2

Planner Constraints		
U	$-$ 0.5 m/s	0.5 m/s
w	$-$ 3 $degrees/s$	3 $degrees/s$
a_U	0 m/s^2	0.1 m/s^2
a_w	$-$ 1 $degrees/s^2$	1 $degrees/s^2$
j_U	$-$ 0.0005 m/s^3	0.0005 m/s^3
j_w	$-$ 0.0003 $degrees/s^3$	0.0003 $degrees/^3$

Table C.13: Planner Constraints Test Two MCSim

Control Parameters	
$K_{P_{Heading}}$	20
$K_{D_{Heading}}$	10
$K_{P_{Sideslip}}$	0.2
$K_{I_{Sideslip}}$	0.02
$K_{P_{Speed}}$	50
$K_{I_{Speed}}$	20

Table C.14: Control Parameters Test Two MCSim

Test 3

Planner Constraints		
U	$-\ 0.5\ m/s$	$0.5\ m/s$
w	$-\ 3\ degrees/s$	$3\ degrees/s$
a_U	$0\ m/s^2$	$0.1\ m/s^2$
a_w	$-\ 1\ degrees/s^2$	$1\ degrees/s^2$
j_U	$-\ 0.0005\ m/s^3$	$0.0005\ m/s^3$
j_w	$-\ 0.0003\ degrees/s^3$	$0.0003\ degrees/^3$

Table C.15: Planner Constraints Test Three MCSim

Control Parameters	
$K_{P_{Heading}}$	20
$K_{D_{Heading}}$	10
$K_{P_{Sideslip}}$	0.2
$K_{I_{Sideslip}}$	0.02
$K_{P_{Speed}}$	50
$K_{I_{Speed}}$	20

Table C.16: Control Parameters Test Three MCSim

Test 4

Planner Constraints		
U	$-\ 0.5\ m/s$	$0.5\ m/s$
w	$-\ 3\ degrees/s$	$3\ degrees/s$
a_U	$0\ m/s^2$	$0.1\ m/s^2$
a_w	$-\ 1\ degrees/s^2$	$1\ degrees/s^2$
j_U	$-\ 0.0005\ m/s^3$	$0.0005\ m/s^3$
j_w	$-\ 0.0003\ degrees/s^3$	$0.0003\ degrees/^3$

Table C.17: Planner Constraints Test Four MCSim

Control Parameters	
$K_{P_{Heading}}$	20
$K_{D_{Heading}}$	10
$K_{P_{Sideslip}}$	0.2
$K_{I_{Sideslip}}$	0.02
$K_{P_{Speed}}$	50
$K_{I_{Speed}}$	20

Table C.18: Control Parameters Test Four MCSim

Appendix D

Parameters for Testing in MCLab

Test 1 - 4

Planner Constraints		
u	$-\ 0.15\ m/s$	$0.15\ m/s$
v	$-\ 0.05\ m/s$	$0.05\ m/s$
w	$-\ 3\ degrees/s$	$3\ degrees/s$
a_u	$-\ 0.1\ m/s^2$	$0.1\ m/s^2$
a_v	$-\ 0.1\ m/s^2$	$0.1\ m/s^2$
a_w	$-\ 10\ degrees/s^2$	$10\ degrees/s^2$
j_u	$-\ 0.0005\ m/s^3$	$0.0005\ m/s^3$
j_v	$-\ 0.0005\ m/s^3$	$0.0005\ m/s^3$
j_w	$-\ 0.003\ degrees/s^3$	$0.003\ degrees/s^3$

Table D.1: Planner Constraints Test 1-4 MCLab

Control Parameters	
$K_{P_{Surge}}$	3.8714
$K_{P_{Sway}}$	7.5319
$K_{P_{Yaw}}$	1.4955
$K_{D_{Surge}}$	36.9495
$K_{D_{Sway}}$	28.4064
$K_{D_{Yaw}}$	4.1977
$K_{I_{Surge}}$	0.2068
$K_{I_{Sway}}$	0.4024
$K_{I_{Yaw}}$	0.0746
$\tau_{max_{surge}}$	2
$\tau_{max_{sway}}$	2
$\tau_{max_{yaw}}$	0.3

Table D.2: Control Parameters Test 1-4 MCLab

Initial and Final Conditions	
Initial North Position	0
Initial East Position	0
Initial Speed	0
Initial Heading Angle	0
Initial Heading Speed	0
Final North Position	4
Final East Position	-1
Final Speed	0
Final Heading Angle	$-36°$
Final Heading Speed	0

Table D.3: Planner Initial and Final Conditions Test 1-4 MCLab

Test 5 - 6

Planner Constraints		
u	$-\ 0.05\ m/s$	$0.05\ m/s$
v	$-\ 0.01\ m/s$	$0.01\ m/s$
w	$-\ 1\ degrees/s$	$1\ degrees/s$
a_u	$-\ 0.01\ m/s^2$	$0.01\ m/s^2$
a_v	$-\ 0.01\ m/s^2$	$0.01\ m/s^2$
a_w	$-\ 1\ degrees/s^2$	$1\ degrees/s^2$
j_u	$-\ 0.0001\ m/s^3$	$0.0001\ m/s^3$
j_v	$-\ 0.0001\ m/s^3$	$0.0001\ m/s^3$
j_w	$-\ 0.001\ degrees/s^3$	$0.001\ degrees/s^3$

Table D.4: Planner Constraints Test 5-6 MCLab

Control Parameters	
$K_{P_{Surge}}$	3.8714
$K_{P_{Sway}}$	7.5319
$K_{P_{Yaw}}$	1.4955
$K_{D_{Surge}}$	36.9495
$K_{D_{Sway}}$	28.4064
$K_{D_{Yaw}}$	4.1977
$K_{I_{Surge}}$	0.2068
$K_{I_{Sway}}$	0.4024
$K_{I_{Yaw}}$	0.0746
$\tau_{max_{surge}}$	2
$\tau_{max_{sway}}$	2
$\tau_{max_{yaw}}$	0.3

Table D.5: Control Parameters Test 5-6 MCLab

Initial and Final Conditions	
Initial North Position	0
Initial East Position	0
Initial Speed	0
Initial Heading Angle	0
Initial Heading Speed	0
Final North Position	4
Final East Position	-1
Final Speed	0
Final Heading Angle	$-9°$
Final Heading Speed	0

Table D.6: Planner Initial and Final Conditions Test 5-6 MCLab

Test 7 - 8

Planner Constraints		
u	$-\ 0.01\ m/s$	$0.01\ m/s$
v	$-\ 0.005\ m/s$	$0.005\ m/s$
w	$-\ 0.286\ degrees/s$	$0.286\ degrees/s$
a_u	$-\ 0.001\ m/s^2$	$0.001\ m/s^2$
a_v	$-\ 0.001\ m/s^2$	$0.001\ m/s^2$
a_w	$-\ 0.5\ degrees/s^2$	$0.5\ degrees/s^2$
j_u	$-\ 0.00001\ m/s^3$	$0.00001\ m/s^3$
j_v	$-\ 0.00001\ m/s^3$	$0.00001\ m/s^3$
j_w	$-\ 0.0001\ degrees/s^3$	$0.0001\ degrees/s^3$

Table D.7: Planner Constraints Test 7-8 MCLab

Control Parameters	
$K_{P_{Surge}}$	3.8714
$K_{P_{Sway}}$	6.5319
$K_{P_{Yaw}}$	1.12
$K_{D_{Surge}}$	36.9495
$K_{D_{Sway}}$	38.4064
$K_{D_{Yaw}}$	14.1977
$K_{I_{Surge}}$	0.2068
$K_{I_{Sway}}$	0.2024
$K_{I_{Yaw}}$	0.0746
$\tau_{max_{surge}}$	2
$\tau_{max_{sway}}$	2
$\tau_{max_{yaw}}$	0.3

Table D.8: Control Parameters Test 7-8 MCLab

Initial and Final Conditions	
Initial North Position	0
Initial East Position	0
Initial Speed	0
Initial Heading Angle	0
Initial Heading Speed	0
Final North Position	2.015
Final East Position	-0.5
Final Speed	0
Final Heading Angle	0°
Final Heading Speed	0

Table D.9: Planner Initial and Final Conditions Test 7-8 MCLab

Test 9

Planner Constraints		
u	$-\ 0.01\ m/s$	$0.01\ m/s$
v	$-\ 0.0001\ m/s$	$0.0001\ m/s$
w	$-\ 0.5\ degrees/s$	$0.5\ degrees/s$
a_u	$-\ 0.01\ m/s^2$	$0.01\ m/s^2$
a_v	$-\ 0.0001\ m/s^2$	$0.0001\ m/s^2$
a_w	$-\ 0.5\ degrees/s^2$	$0.5\ degrees/s^2$
j_u	$-\ 0.00005\ m/s^3$	$0.00005\ m/s^3$
j_v	$-\ 0.000001\ m/s^3$	$0.000001\ m/s^3$
j_w	$-\ 0.0001\ degrees/s^3$	$0.0001\ degrees/s^3$

Table D.10: Planner Constraints Test 9 MCLab

Control Parameters	
$K_{P_{Surge}}$	3.8714
$K_{P_{Sway}}$	6.5319
$K_{P_{Yaw}}$	1.12
$K_{D_{Surge}}$	36.9495
$K_{D_{Sway}}$	38.4064
$K_{D_{Yaw}}$	14.1977
$K_{I_{Surge}}$	0.2068
$K_{I_{Sway}}$	0.2024
$K_{I_{Yaw}}$	0.0746
$\tau_{max_{surge}}$	2
$\tau_{max_{sway}}$	2
$\tau_{max_{yaw}}$	0.3

Table D.11: Control Parameters Test 9 MCLab

Initial and Final Conditions	
Initial North Position	0
Initial East Position	0
Initial Speed	0
Initial Heading Angle	0
Initial Heading Speed	0
Final North Position	3.4
Final East Position	0
Final Speed	0
Final Heading Angle	0°
Final Heading Speed	0

Table D.12: Planner Initial and Final Conditions Test 9 MCLab

Appendix E

Results from Testing in MCLab

Path One

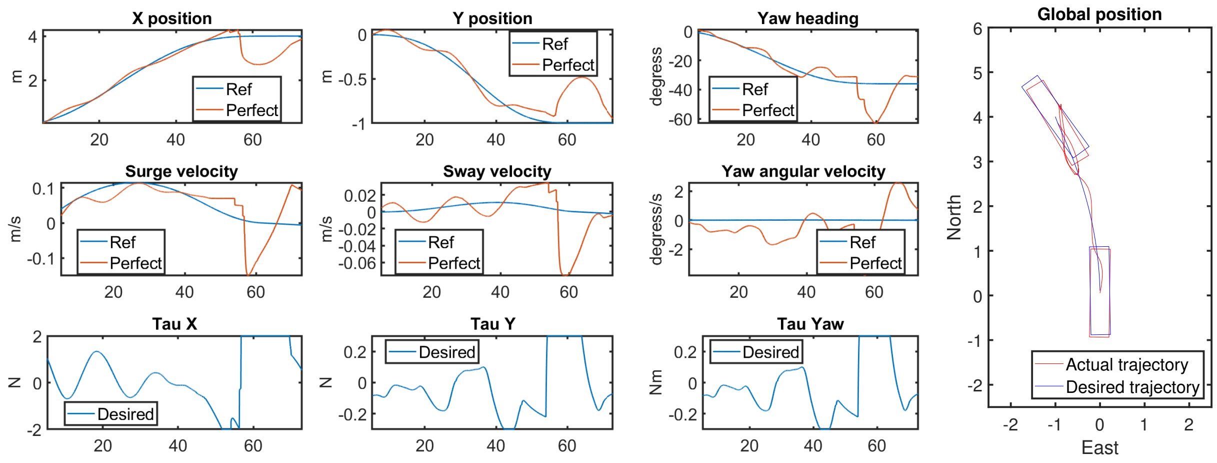


Figure E.1: MCLab Test One

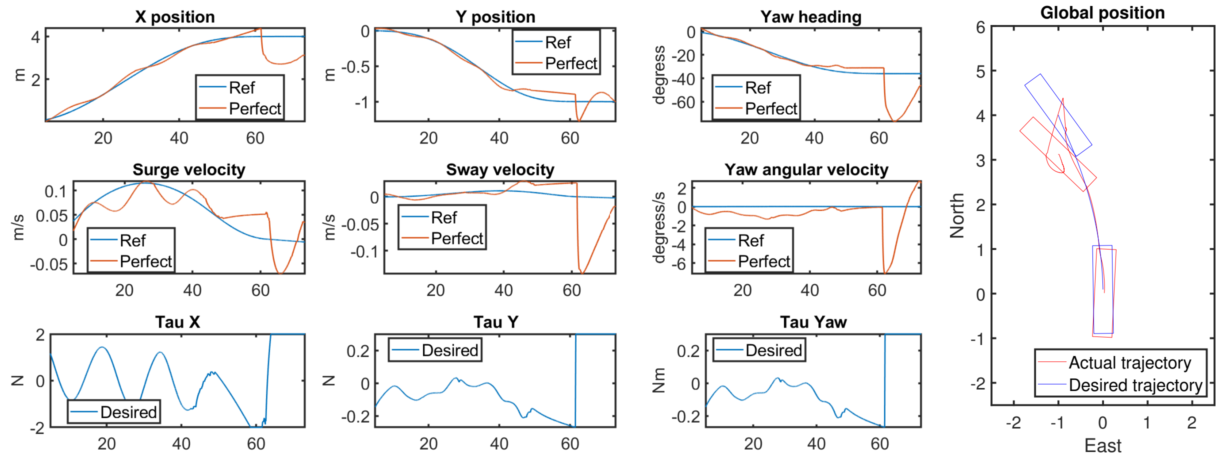


Figure E.2: MCLab Test Two

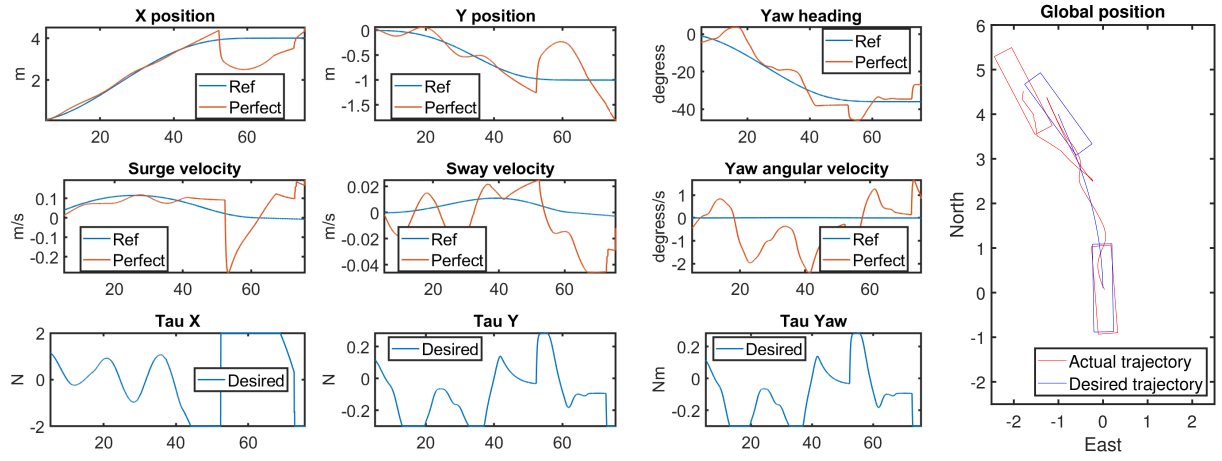


Figure E.3: MCLab Test Three

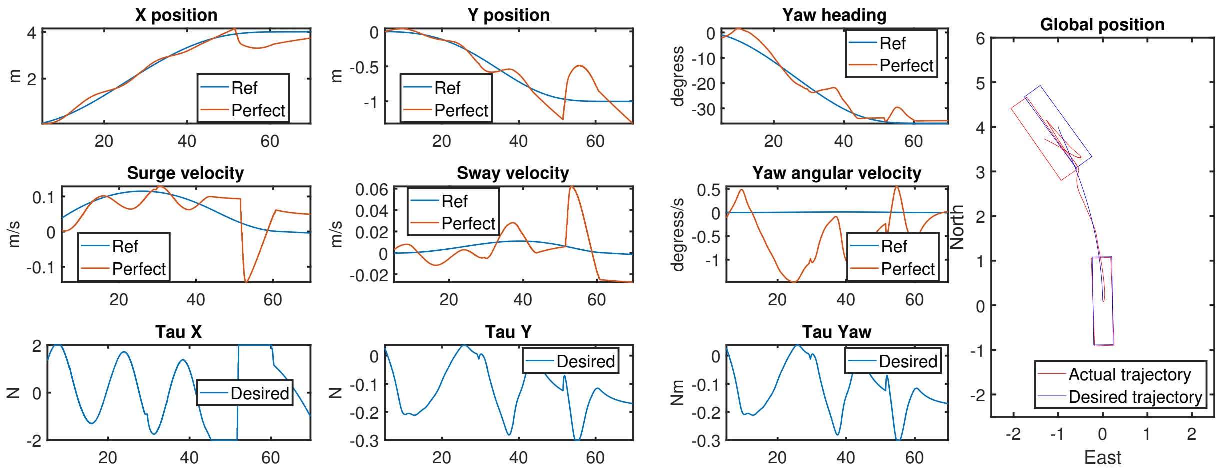


Figure E.4: MCLab Test Four

Path Two

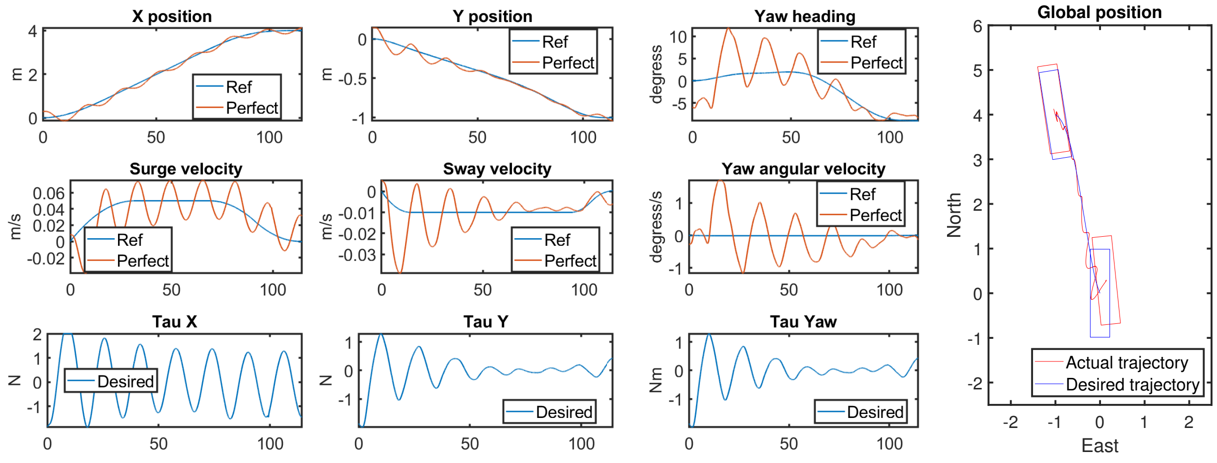


Figure E.5: MCLab Test Five

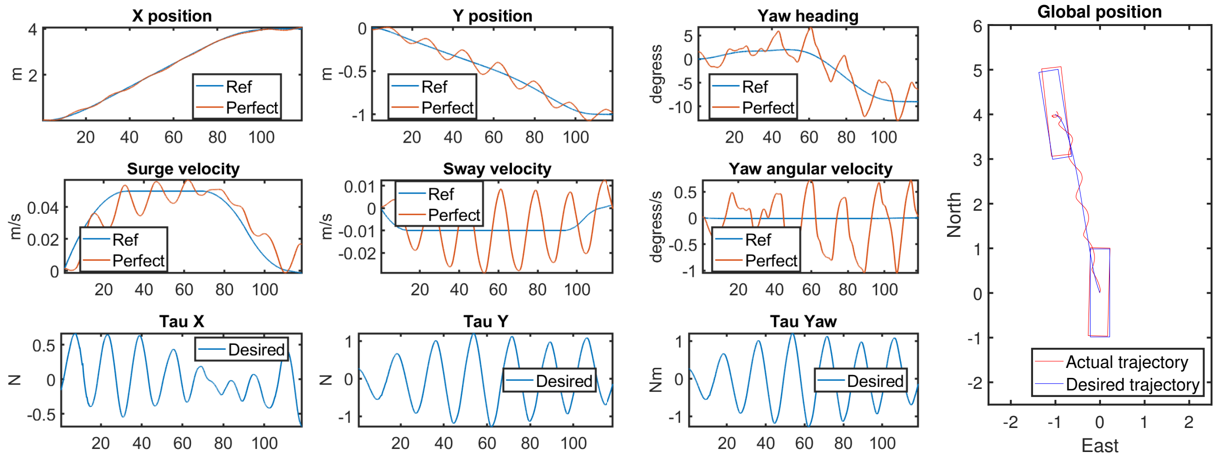


Figure E.6: MCLab Test Six

Path Three

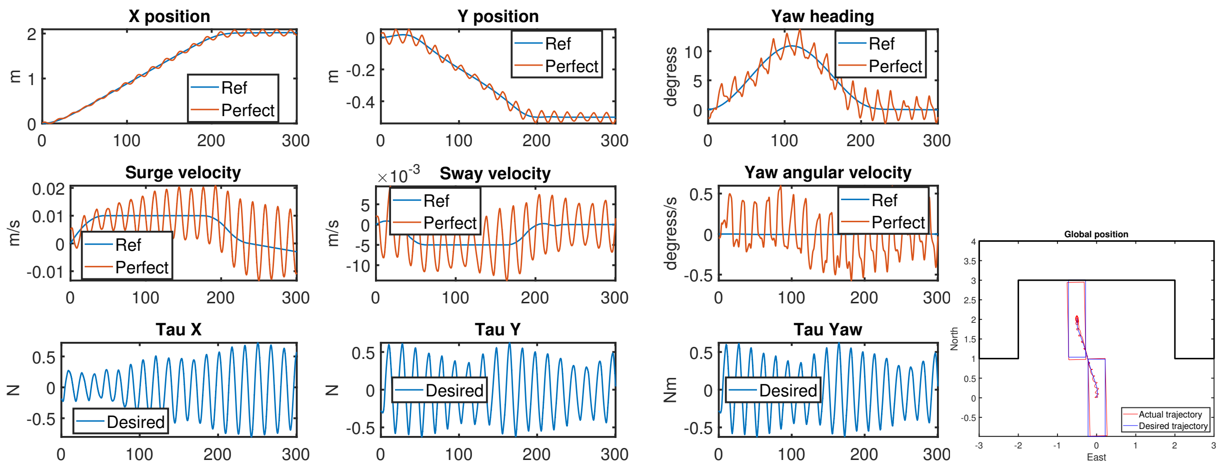


Figure E.7: MCLab Test Seven

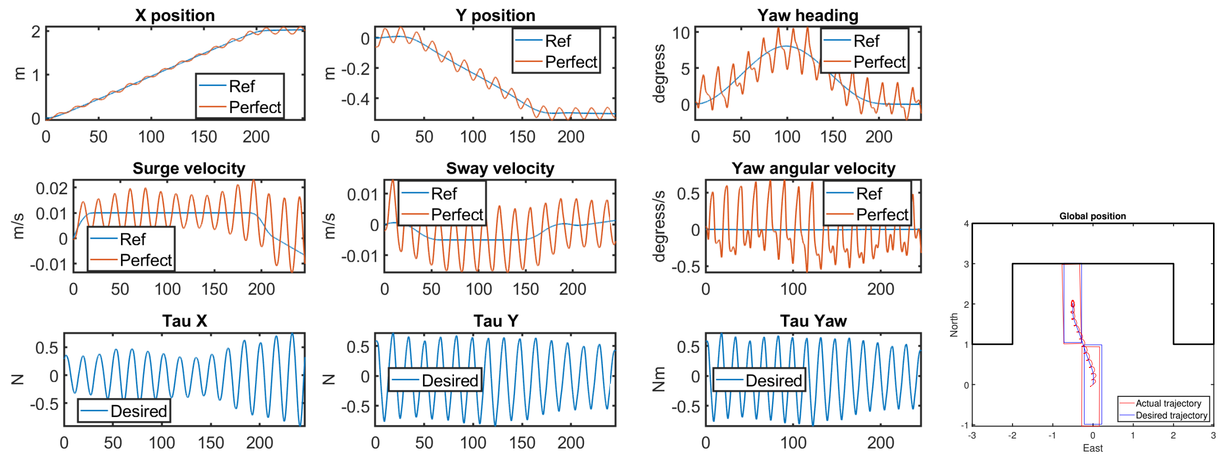


Figure E.8: MCLab Test Eight

Path Four

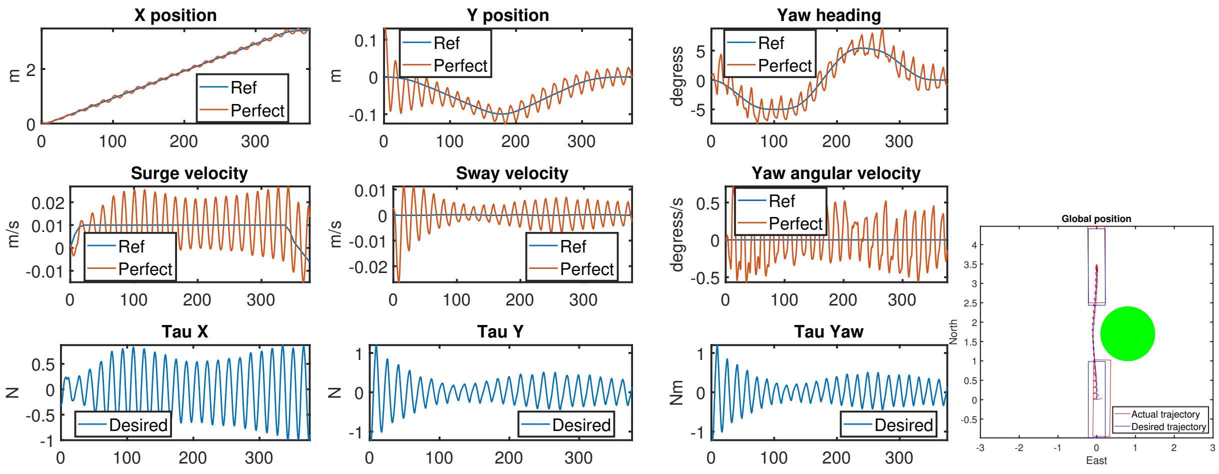


Figure E.9: MCLab Test Nine

Appendix F

Matlab Code

image_process_point.m

```
function [lines, corners_sort] = image_process_point(north_start,east_start,north_finish,
    east_finish)
filepath=fullfile("C:\Users\Thomas\Documents\MATLAB\Master_thesis\kai_flakk.png");
% filepath=fullfile("C:\Users\Thomas\Documents\MATLAB\Master_thesis\liten_kai.png");
image = imread(filepath);
grayimage = rgb2gray(image)*0.5;
MAP = grayimage < 0.5; % (0,0) is top left corner in original image
H = detectHarrisFeatures(MAP);
corners_round = round(H.Location,-1);

left_edge = min(corners_round(:,1));
right_edge = max(corners_round(:,1));
bottom_edge = min(corners_round(:,2));
top_edge = max(corners_round(:,2));

corners_sort = zeros(size(corners_round,1)+1,size(corners_round,2));

start = find(corners_round(:,1) == left_edge);
corners_sort(1,1) = left_edge;
corners_sort(1,2) = max(corners_round(start,2));
corners_sort(size(corners_sort,1),:) = corners_sort(1,:);
entry = find(corners_round(start,2)==corners_sort(1,2));
corners_round(entry,:) = [];
for i = 2: size(corners_sort,1)-1
    if mod(i,2)==0
       next = corners_sort(i-1,1);
       possible = find(corners_round(:,1)==next);
       if length(possible)==1
           corners_sort(i,:) = corners_round(possible,:);
           corners_round(possible,:) = [];
       else
           next_entry = find(corners_round(possible,2)==min(corners_round(possible,2)));
           corners_sort(i,:) = corners_round(next_entry,:);
           corners_round(next_entry,:) = [];
       end
    else
       next = corners_sort(i-1,2);
       possible = find(corners_round(:,2)==next);
       if length(possible)==1
           corners_sort(i,:) = corners_round(possible,:);
```

```
            corners_round(possible,:) = [];
        else
            next_entry = find(corners_round(possible,1)==min(corners_round(possible,1)));
            corners_sort(i,:) = corners_round(next_entry,:);
            corners_round(next_entry,:) = [];
        end
    end
endlines = zeros(size(corners_sort,1)-1,6);
amount_lines = size(corners_sort,1)-1;
for i = 1:amount_lines
    lines(i,[1:4]) = [corners_sort(i,:), corners_sort(i+1,:)];
    lines(i,5) = point_to_line(corners_sort(i,:), corners_sort(i+1,:),east_finish,
        north_finish);
    lines(i,6) = point_to_line_CasADi(corners_sort(i,:), corners_sort(i+1,:),east_finish,
        north_finish);
end
end
```

image_process_circle.m

```
function [radius, center_point] = image_process_circle()
filepath=fullfile("C:\Users\Thomas\Documents\MATLAB\Master_thesis\islands.png");
image = imread(filepath);
grayimage = rgb2gray(image)*0.5;
MAP = grayimage < 0.5; % (0,0) is top left corner in original image
stats = regionprops('table',MAP,'Centroid','MajorAxisLength','MinorAxisLength');
center_point = stats.Centroid;
diameters = mean([stats.MajorAxisLength stats.MinorAxisLength],2);
radius = diameters/2;
end
```

point_to_line_CasADi.m

```
function d = point_to_line_CasADi(v1, v2,east, north)
    pt = [east, north];
    north1 = v1(2);
    east1 = v1(1);
    north2 = v2(2);
    east2 = v2(1);
    bool_test = (dot(v1-v2,pt-v2)*dot(v2-v1,pt-v1)>=0);
    d_perp = abs((north2-north1)*east - (east2-east1)*north + east2*north1 - north2*east1
        )/sqrt((north2-north1)^2+(east2-east1)^2);
    d1 = sqrt((north1-north).^2 + (east1-east).^2);
    d2 = sqrt((north2-north).^2 + (east2-east).^2);
    d = d_perp*(bool_test==1) + min(d1,d2)*(bool_test~=1);
end
```

MPC_Opti.m

```
import casadi.*

opti = casadi.Opti(); % Optimization problem

% ---- decision variables ---------
X = opti.variable(4,N+1); % state trajectory
north   = X(1,:); % NORTH
east   = X(2,:); % EAST
speed = X(3,:);   % Surge speed
heading = X(4,:); % Heading
U = opti.variable(2,N+1); % control trajectory
```

```matlab
a = U(1,:); % Acceleration
omega = U(2,:); % Yaw rate
T = opti.variable();
opti.minimize(T);% + sum(a.^2) + sum(omega.^2));

% Differential equations (Will depend on the model)
f = @(x,u) [x(3)*cos(x(4)); x(3)*sin(x(4)); u(1); u(2)];

dt = T/N; % length of a control interval
for k=1:N % loop over control intervals
   % Runge-Kutta 4 integration
   k1 = f(X(:,k),         U(:,k));
   k2 = f(X(:,k)+dt/2*k1, U(:,k));
   k3 = f(X(:,k)+dt/2*k2, U(:,k));
   k4 = f(X(:,k)+dt*k3,   U(:,k));
   x_next = X(:,k) + dt/6*(k1+2*k2+2*k3+k4);
   opti.subject_to(X(:,k+1)==x_next); % close the gaps
   opti.subject_to((a(k+1)-a(k))/dt >= jerk_min);
   opti.subject_to((a(k+1)-a(k))/dt <= jerk_max);
   opti.subject_to((omega(k+1)-omega(k))/dt >= angularAcc_min);
   opti.subject_to((omega(k+1)-omega(k))/dt <= angularAcc_max);
end

% ---- control constraints -----------
opti.subject_to(a_min<=a<=a_max);
opti.subject_to(omega_min<=omega<=omega_max);

%Circle obstacles
opti.subject_to(obstacle1(north,east)>radius1 + 10);
opti.subject_to(obstacle2(north,east)>radius2 + 10);

%Rectangular obstacles
for i = 1:size(lines,1)
    if lines(i,5) > 10
        for t = 1:N+1
             opti.subject_to(point_to_line_CasADi(lines(i,[1:2]), lines(i,[3:4]),east(t),
                 north(t))>=5);
        end
    end
end

% ---- boundary conditions --------
% Initial conditions
opti.subject_to(north(1)==north_start);   % start at position 0 ...
opti.subject_to(east(1)==east_start);
opti.subject_to(heading(1)==heading_start);
opti.subject_to(speed(1)==speed_start);

opti.subject_to(speed_min <= speed <= speed_max);
% opti.subject_to(heading_min <= heading <= heading_max);
opti.subject_to(north(N+1)==north_finish);   % start at position 0 ...
opti.subject_to(east(N+1)==east_finish);
opti.subject_to(heading(N+1)==heading_finish);
opti.subject_to(speed(N+1)==speed_finish);

opti.subject_to(omega(N+1)== 0);
opti.subject_to(a(N+1)==0);

opti.subject_to(T>=0);
opti.set_initial(T,shortest_time);

options = struct;
options.ipopt.print_level = 3;
opti.solver('ipopt', options); % set numerical backend

sol = opti.solve();   % actual solve
```

```

% Extract results
x1_opt = sol.value(north);
x2_opt = sol.value(east);
x3_opt = sol.value(speed);
x4_opt = rad2pipi(sol.value(heading));
u1_opt = sol.value(a);
u2_opt = sol.value(omega);
time = sol.value(T);
steps = sol.value(dt);
timegrid = (0:steps:time);
```

MPC_multiple_shooting.m

```
import casadi.*

% Declare model variables
x1 = SX.sym('x1'); % NORTH
x2 = SX.sym('x2'); % EAST
x3 = SX.sym('x3'); % Surge speed
x4 = SX.sym('x4'); % Heading
x = [x1; x2; x3; x4];
u1 = SX.sym('u1'); % Acceleration
u2 = SX.sym('u2'); % Yaw rate
u = [u1; u2];

% Model equations
xdot = [x3*cos(x4); x3*sin(x4); u1; u2];

% Objective term
L = u2^2 + u1^2;

%Circle
L = L + 5e-2/obstacle1(x1,x2) + 5e-2/obstacle2(x1,x2);

%Rectangles
for i = 1:size(lines,1)
    if lines(i,5) > 10
        L = L + 1e-2/point_to_line_CasADi(lines(i,[1:2]), lines(i,[3:4]),x2,x1);
    end
end
% Continuous time dynamics
f = Function('f', {x, u}, {xdot, L});

% Formulate discrete time dynamics
if false
   % CVODES from the SUNDIALS suite
   dae = struct('x',x,'p',u,'ode',xdot,'quad',L);
   opts = struct('tf',T/N);
   F = integrator('F', 'cvodes', dae, opts);
else
   % Fixed step Runge-Kutta 4 integrator
   DT = T/N/M;
   f = Function('f', {x, u}, {xdot, L});
   X0 = MX.sym('X0', 4);
   U = MX.sym('U', 2);
   X = X0;
   Q = 0;
   for j=1:M
       [k1, k1_q] = f(X, U);
       [k2, k2_q] = f(X + DT/2 * k1, U);
       [k3, k3_q] = f(X + DT/2 * k2, U);
       [k4, k4_q] = f(X + DT * k3, U);
       X=X+DT/6*(k1 +2*k2 +2*k3 +k4);
```

```
        Q = Q + DT/6*(k1_q + 2*k2_q + 2*k3_q + k4_q);
    end
    F = Function('F', {X0, U}, {X, Q}, {'x0','p'}, {'xf', 'qf'});
end

% Evaluate at a test point
Fk = F('x0',[0.2; 0.3; 0.4; 0.5],'p',0.4);
disp(Fk.xf)
disp(Fk.qf)

% Start with an empty NLP
w={};
w0 = [];
lbw = [];
ubw = [];
J = 0;
g={};
lbg = [];
ubg = [];

% "Lift" initial conditions
Xk = MX.sym('X0', 4);
w = {w{:}, Xk};
lbw = [lbw; north_start; east_start; speed_start; heading_start];
ubw = [ubw; north_start; east_start; speed_start; heading_start];
w0 = [w0; north_start; east_start; speed_start; heading_start];

% Formulate the NLP
for k=0:N-2
    % New NLP variable for the control
    Uk = MX.sym(['U_' num2str(k)], 2);
    w = {w{:}, Uk};
    lbw = [lbw; a_min; omega_min];
    ubw = [ubw;  a_max; omega_max];
    w0 = [w0;  0; 0];

    % Integrate till the end of the interval
    Fk = F('x0', Xk, 'p', Uk);
    Xk_end = Fk.xf;
    J=J+Fk.qf;

    % New NLP variable for state at end of interval
    Xk = MX.sym(['X_' num2str(k+1)], 4);
    w = [w, {Xk}];
    lbw = [lbw; -inf; -inf; speed_min; heading_min];
    ubw = [ubw;  inf;  inf; speed_max; heading_max];
    w0 = [w0; 0; 0; 0; 0];

    % Add equality constraint
    g = [g, {Xk_end-Xk}];
    lbg = [lbg; 0; 0; 0; 0];
    ubg = [ubg; 0; 0; 0; 0];
end
k=N-1;
% New NLP variable for the control
Uk = MX.sym(['U_' num2str(k)], 2);
w = {w{:}, Uk};
lbw = [lbw; a_min; omega_min];
ubw = [ubw;  a_max; omega_max];
w0 = [w0;  0; 0];

% Integrate till the end of the interval
Fk = F('x0', Xk, 'p', Uk);
Xk_end = Fk.xf;
J=J+Fk.qf;

```

```
% New NLP variable for state at end of interval
Xk = MX.sym(['X_' num2str(k+1)], 4);
w = [w, {Xk}];
lbw = [lbw; north_finish; east_finish; speed_finish; heading_finish];
ubw = [ubw; north_finish; east_finish; speed_finish; heading_finish];
w0 = [w0; north_finish; east_finish; speed_finish; heading_finish];

% Add equality constraint
g = [g, {Xk_end-Xk}];
lbg = [lbg; 0; 0; 0; 0];
ubg = [ubg; 0; 0; 0; 0];


% Create an NLP solver
prob = struct('f', J, 'x', vertcat(w{:}), 'g', vertcat(g{:}));
options = struct;
options.ipopt.print_level = 3;
solver = nlpsol('solver', 'ipopt', prob, options);

% Solve the NLP
sol = solver('x0', w0, 'lbx', lbw, 'ubx', ubw,...
            'lbg', lbg, 'ubg', ubg);
w_opt = full(sol.x);

% Plot the solution
x1_opt = w_opt(1:6:end);
x2_opt = w_opt(2:6:end);
x3_opt = w_opt(3:6:end);
x4_opt = w_opt(4:6:end);
u1_opt = w_opt(5:6:end);
u2_opt = w_opt(6:6:end);
time = T;
timegrid = linspace(0, T, N+1);
```

Thomas Johansen

NTNU
Norwegian University of
Science and Technology