Kristian Nilsen

Planning, Scheduling and Control for Autonomous Passenger Ferry Operations in Urban Environments

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2023

Kristian Nilsen

Planning, Scheduling and Control for Autonomous Passenger Ferry Operations in Urban Environments

Master's thesis in Cybernetics and Robotics
Supervisor: Anastasios Lekkas
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Kunnskap for en bedre verden

Planning, Scheduling and Control for Autonomous Passenger Ferry Operations in Urban Environments

Author:
Kristian Nilsen

Master of Science in Cybernetics and Robotics
Submission date: June 2023
Supervisor: Anastasios Lekkas, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This master's thesis is the result of my work during the spring of 2023 at the Norwegian University of Science and Technology (NTNU), under the supervision of Anastasios Lekkas. The thesis is a summary of my findings exploring how autonomous passenger ferries can be used as an alternative method of transportation, using the city of Trondheim as an example. Using autonomous ferries in urban environments has gotten an increased attention lately, and the transportation company AtB wants to look at the possibility of using this. In this thesis, we have therefore made a simulation exploring how this can be accomplished. In order to try to utilize the available ferries most efficiently, we also incorporate a high-level mission planner using a field of research within Artificial Intelligence (AI) called AI-planning.

The thesis assumes that the reader has prior knowledge of basic control theory, such as Proportional-Integral-Derivative (PID)-controllers, as well as the basics of optimization. Background within the use of optimization for control, and modeling and motion control of marine vessels is beneficial, but relevant notation, methods and kinematics will be explained. All theory regarding AI-planning needed to understand the results will be presented.

The main contribution of this thesis is that it shows how a fleet of autonomous ferries can be used for transportation in the channels in Trondheim. The whole simulation is made in Python, and is not built on any existing source code. However, the AI-planning algorithm is taken from GitHub [1]. For optimization, the Python Application Programming Interface (API) for the open-source tool CasADi is used, while NumPy and SciPy are used for mathematical operations. Matplotlib is used for plotting, and the Python module subprocess is used to run the AI-planner from the simulation.

Acknowledgments

I would like to thank my supervisor Anastasios Lekkas for guiding me through this semester, and Eirin Aarem and the rest of AtB for providing relevant information and insight into the project. I would also like to thank Simon Lexau for helping me with the control system. Next, I have to thank my fellow students for their help and support, and especially Øystein Solbø and Henrik Nordhus for their help with AI-planning and control, respectively. Lastly, I want to thank my family for their support, and my girlfriend, Molly Vassbotn, for her massive support and love.

Kristian Nilsen
Trondheim, June 1, 2023

Abstract

Most urban areas are in some way connected to waterways because of their historical importance. However, in modern times the usage of waterways has lost terrain as it has been perceived to be too costly. With the increased capabilities of autonomous systems, this trend seems to change again with the introduction of autonomous passenger ferries. Several pilot projects regarding autonomous ferries have been launched around the world, and this has sparked the interest of the transportation company AtB. Trondheim faces a slight population shift as the old harbor area at Nyhavna is set to be transformed into living spaces, making AtB interested in exploring new methods of transportation.

The aim of this thesis is to investigate how small autonomous passenger ferries can be used as a method of transportation in the channels of Trondheim. In order to do this the thesis presents a simulation of a fleet of ferries. To make the simulation realistic, the model of an existing vessel called milliAmpere 1 is used, and a control system for the vessel is made. A stochastic simulation of passengers appearing is also made, and is based on information provided by AtB. Lastly an Artificial Intelligence (AI)-planner is used to distribute the ferries efficiently.

By simulating a day of operations for different scenarios it is shown how a fleet of ferries can handle the expected amount of passengers, with an acceptable traveling time. However, this requires a vessel with a capacity of at least 10 passengers, and an operational velocity of around $3m/s$. The AI-planner distributed the ferries in a satisfactory way, but it did not manage to handle more detailed planning problems. With lower-level actions, redundant parts could have been omitted, which could have increased efficiency beyond what can be expected with purely predefined routes. The control system was able to generate sensible trajectories between each port, and follow them closely. To further develop this project the control system has to be adjusted to work for the actual vessel that will be used as a ferry. Next, control problems such as collision avoidance have to be included. Lastly, it would be interesting to see if a planner could make the system more efficient.

Sammendrag

De fleste urbane områder er på en aller annen måte tilknyttet en vannvei, fordi dette har historisk sett vært viktig. I moderne tid har ikke vannveier blitt brukt like mye fordi det har vært ansett for å være for kostbart. Med utviklingen av autonome systemer har denne trenden sett ut til å snu med introduksjonen av autonome passasjerferger. Flere pilotprosjekter har blitt lansert rundt omkring i verden, noe som har vekket interessen til transportselskapet AtB. Trondheim står ovenfor et skifte i befolkningstetthet ettersom havneområdet Nyhavna skal gjøres om til boenheter. Dette har gjort at AtB ønsker å se på alternative måter for transport.

Målet for denne avhandlingen er å undersøke hvordan små autonome passasjerferger kan brukes som en transportmetode i kanalene i Trondheim. For å gjøre dette presenterer denne avhandlingen en simulering av en flåte med ferger. For å gjøre simuleringen realistisk er det brukt en modell av det eksisterende fartøyet milliAmpere 1, og et kontrollsystem tilpasset fartøyet er laget. En stokastisk simulering av ankommende passasjerer er også laget, og er bygd på informasjon fra AtB. Til slutt er en Kunstig Intelligens (KI)-planlegger brukt for å fordele fergene effektivt.

Ved å simulere en dag i bruk for ulike scenarioer blir det vist at fergene kan håndtere den forventede mengden med passasjerer, med en akseptabel reisetid. Samtidig krever dette et fartøy som har plass til minst 10 passasjerer og har en operativ hastighet på rundt $3m/s$. KI-planleggeren fordelte fergene på en tilfredsstillende måte, men klarte ikke å håndtere alternative formuleringer av planleggingsproblemet. Dette kunne ha økt effektiviteten utover det som kan forventes med rene predefinerte ruter. Kontrollsystemet klarte å generere fornuftige baner, og følge dem tett. For å videreføre prosjektet må kontrollsystemet justeres for å virke for det fartøyet som faktisk vil bli brukt som ferge. Det neste vil være å inkludere kontrollproblemer som kollisjonsunngåelse. Til slutt ville det vært interessant å se om en planlegger kan gjøre systemet enda mer effektivt.

Table of Contents

List of Figures

List of Tables

List of Acronyms

NTNU	Norwegian University of Science and Technology	1
AI	Artificial Intelligence	2
STRIPS	Stanford Research Institute Problem Solver	2
POPF	Partial Order Planning Forwards	2
OPTIC	Optimising Preferences and Time-Dependent Costs	2
AUV	Autonomous Underwater Vehicle	2
MAV	Micro Aerial Vehicle	2
UAV	Unmanned Aerial Vehicle	3
GA+TP	Goal Allocation and Temporal Planning	3
DHRTA	Decentralised Heterogeneous Robot TaskAllocator	3
ASV	Autonomous Surface Vehicle	3
MIP	Mixed-Integer Programming	3
MPC	Model Predictive Control	3
DP	Dynamic Positioning	3
PID	Proportional-Integral-Derivative	3
OCP	Optimal Control Problem	5
PDDL	Planning Domain Definition Language	6
EHC	Enforced Hill-Climbing	8
FF	Fast Forward	8
LP	Linear Programming	9
SNAME	Society of Naval Architects and Marine Engineers	10
DOF	Degrees of Freedom	10
NED	North, East, Down	10

List of Symbols

$\boldsymbol{\eta}$	Position and Euler angle vector
$\boldsymbol{\nu}$	Linear and angular velocity vector
$\boldsymbol{\tau}$	Forces on vessel from actuators
$\boldsymbol{f}$	Decomposed force vector
$\boldsymbol{u}$	Input to vessel actuators
$\boldsymbol{x}$	State vector
ψ	Heading/yaw
N	Yaw moment
r	Yaw rate
u	Surge
v	Sway
X	Surge force
x	North position in the NED-frame
Y	Sway force
y	East position in the NED-frame

Chapter 1

Introduction

1.1 Motivation and Background

In recent years there has been a lot of interest in smart cities [2], which are cities that use information and technology to make themselves more livable. One major part of a smart city is to make transportation both more efficient and less polluting. A solution to this is using electric autonomous ferries, which has gotten an increased amount of attention lately [3]. Most urban areas are in some way connected to waterways, but in modern times these have been utilized less because of how costly it has been [4]. Small ferries without a crew, or with limited need for human interaction mean that this is no longer an issue.

There have been several projects in the last years using autonomy for ferries. Already in 2018 the car-ferry Falco, made by Rolls-Royce Marine (now part of Kongsberg Gruppen), was the first to perform a fully autonomous transit, outside of Turku in Finland [5]. The same year ABB demonstrated the use of autonomy for a passenger ferry outside of the Helsinki harbor [6]. Both these projects were tested outside of regular service, but in 2020 Kongsberg Maritime started using fully automatic control for the Bastø Fosen VI ferry, which is a passenger ferry doing regular routes [7]. An ongoing small passenger ferry project is the milliAmpere project developed at Norwegian University of Science and Technology (NTNU). This project resulted in the company Zeabuz, which had trial missions in Trondheim in 2022 [8]. In the summer of 2023, Torghatten and Brødrene Aa will launch a fully autonomous ferry service in Stockholm using the technology provided by Zeabuz. In the beginning the ferry will always have a captain, but eventually the ferry will only be monitored from land [9, 10]. Another project is the Roboat project in Amsterdam, which aims to make small autonomous vessels able to handle tasks like passenger transportation and rubbish collection [11].

Trondheim is a city with a river and a channel running through the city. According to Trondheim Kommune a part of the city, called Nyhavna, is set to be transformed from a harbor area to living spaces [12]. There is a single road connecting this part of town to the city center, and it does not have the capacity to handle the increased traffic. Therefore, the transportation company AtB wants to look at the possibility of using small autonomous passenger ferries as an alternative to cars and buses, since the western side of Nyhavna borders the waterways of Trondheim.

While manned ferries and buses are limited by work shifts of the personnel, an autonomous ferry can wait dormant until it is needed. This presents an opportunity, where, for example, two ferries can follow predefined routes, while a third ferry waits and is ready to handle extra needed capacity. It can also take over a route as soon as another ferry gets into some sort of trouble. Autonomy can also make the recharging of ferries more seamless. When an active ferry needs recharging, a standby ferry can already be en route to take over for this ferry. These kinds of intelligent decisions can be handled by a high-level mission planner such as an Artificial Intelligence (AI)-planner. Even though such high-level planning is an important part of autonomy, it has received less attention than control and navigation [13].

1.2 Past Work

There have been developed a number of AI-planning methods, also called automated planning methods [14]. Already in 1971 the first AI-planning algorithm, called the Stanford Research Institute Problem Solver (STRIPS), was developed and tested on a mobile robot operating in a controlled environment [15]. Since then, the field of AI-planning has developed a lot. One area that has benefited from automated planning is space exploration [16, 17], where signals have significant time delays. This leads to operators not being able to control everything, and automatic solutions become necessary.

Unlike STRIPS, some of the newer methods can handle temporal actions and numeric-fluents, meaning that every action has a duration and can modify numeric variables. Two of the most popular planners among these newer methods are Partial Order Planning Forwards (POPF) [18] and its successor Optimising Preferences and Time-Dependent Costs (OPTIC) [19]. These planners have been used in real missions for Autonomous Underwater Vehicle (AUV) [20, 21], Micro Aerial Vehicle (MAV) [22] and Unmanned Aerial Vehicle (UAV) [23]. Even newer approaches have found that combining several planners into portfolios [24, 25] makes more robust planners. Using AI-planners with temporal actions for a fleet of cooperative agents is an ongoing research area [26]. Some have used POPF [27], while others have come up with methods specialized for multi-agent purposes [28].

In marine applications AI-planning has mainly been used for environment sampling missions [29, 30, 31]. Planners made specifically for mission planning for a fleet of AUVs includes Goal Allocation and Temporal Planning (GA+TP) [32], and Decentralised Heterogeneous Robot TaskAllocator (DHRTA) [33]. These planners combine temporal planning with a goal distribution scheme. Using AI-planning for Autonomous Surface Vehicle (ASV) has not gotten the same attention, but it has been used for an autonomous container ship [13].

Another way to perform high-level mission planning is to use optimization, where the problem is modeled using Mixed-Integer Programming (MIP). This has been done for a single UAV [34], a fleet of UAVs [35], and military air missions [36]. It is also possible to use more classical algorithms, or newer variations of these, such as the multiple traveling salesman problem [37].

The lower level control hierarchy used in this thesis is described further in Section 1.3 and Chapter 3. At the top, there is a trajectory planner, which is used for both docking and transit. Until recent years, research on automatic docking has been limited. Some earlier papers use target tracking [38] or artificial potential fields [39]. These papers do not take the vessel dynamics into account and their use is quite limited. Some articles take the vessel dynamics into account, and use artificial neural networks [40, 41] or Model Predictive Control (MPC) [42] to control the vessel. However, these methods do not include collision avoidance with the harbor layout. This can be taken into account by modeling the harbor as a convex area [43, 44], such as has been done for the milliAmpere 1 [45].

In harbors with multiple obstacles a single convex set is impossible to find. Therefore, some papers have included maneuvering in such complex environments [46, 47], as done for the milliAmpere 1 [48]. This is done by using graph searching methods that can find a proper path [49, 50]. There are several other path planning methods for transit [51], such as Voronoi diagrams [52], cell decomposition [53] or deep reinforcement learning [54].

In the middle of the control hierarchy is a Dynamic Positioning (DP)-controller used to track the planned trajectory. The simplest method for DP is a simple Proportional-Integral-Derivative (PID) controller with feedforward terms [55, 44].

These methods do not require a lot of computation, but do not use the vessel model explicitly either. More complex methods use MPC [56], nonlinear MPC [57] or backstepping [58], and thereby use the model explicitly. However, they require more computations.

At the bottom of the control hierarchy lies the thrust allocation. There are several ways to do this [59, 60], but the most usual ways are the Pseudo-inverse method [61] or a variation of the Quadratic Programming method [62]. The simplest way is the Pseudo-inverse method which is simple matrix operations. This yields low computational complexity, but an unconstrained solution. A Quadratic Programming method can include constraints, but its computational complexity is much higher. The model constraints also have to be linearized, which can be limiting. For the milliAmpere 1, an effective nonlinear optimization problem is proposed because of the nonlinear nature of the thrusters [63].

1.3 Objectives and Contributions

The primary objective of this thesis is to see if it is possible to use a fleet of small autonomous passenger ferries as an alternative to cars and buses when traveling to and from Nyhavna in Trondheim. In order to do this, a simulation is made where a number of passengers appear throughout the day, wanting to use the service. Then, a system consisting of a central planner and several vessels is made to transport the passengers. Each vessel has its own control system. The system hierarchy is further described in Chapter 3.

To make the simulation realistic, a trajectory planner, a trajectory tracker and a thrust allocation algorithm are implemented as described in Section 3.6, Section 3.7 and Section 3.8. The milliAmpere 1 model is used as a vessel model since there is published a number of articles about the vessel and its capabilities [45]. A picture of the milliAmpere 1 is shown in Figure 1.1. This combination gives realistic movements, and a realistic amount of time the vessel uses when traveling between the ports. Major aspects like collision avoidance, sensor uncertainty and environmental forces are omitted for simplicity.

The secondary objective is to see how AI-planning can make intelligent decisions as a high-level mission planner. In order to reach this objective, an AI-planner is used to delegate different tasks, such as following a route connected to a timetable. In addition, the high-level actions are translated into control actions by dividing the actions into control tasks.

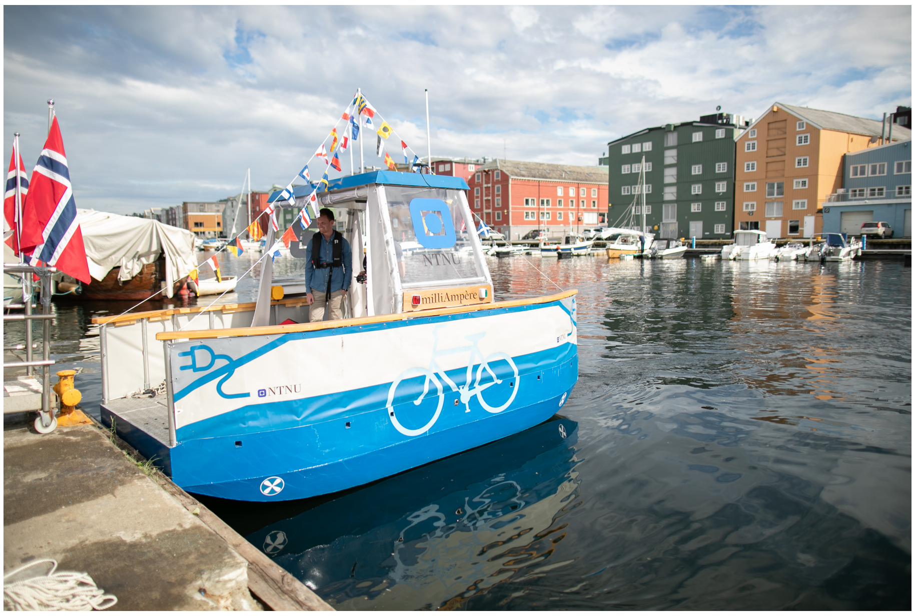


Figure 1.1: milliAmpere 1 (Photo credit: Kai T. Dragland [64]).

1.4 Thesis Overview

This thesis is divided into five chapters, starting with some necessary background in Chapter 2. This includes an introduction to AI-planning, how to make an AI-planning problem, and how the AI-planner OPTIC solves a problem. In addition, the background chapter introduces theory about marine vessel models, and the model we use for milliAmpere 1. Lastly, there are some theory about how to model and solve an Optimal Control Problem (OCP). In Chapter 3 the system architecture and every single module in it is presented along with the world the system operates within. At the top of the system architecture is an AI-planner, and at the bottom is the control system for each vessel. A control system consists of a trajectory planner based on an OCP, a trajectory tracker based on DP, and a thrust allocation scheme based on nonlinear optimization. How the planner is communicating with the control system is also described. How the system performs is explored in Chapter 4. First, we look at the average waiting time for passengers and distribution of the ferries throughout the day for five different scenarios. Next, we look closer at what happens in the evening rush for two of the scenarios. Then we look at how the control system performs during a single trip. Lastly, we discuss some challenges encountered when using AI-planning. In Chapter 5 the thesis is concluded, and future work is described.

Chapter 2

Background

In this chapter the necessary theory about AI-planning, as well as how we model the kinematics of a maritime vessel is presented. The model we use for milliAmpere 1 is also presented. Lastly, we look at how an OCP can be modeled and how it can be solved with a direct collocation method.

2.1 AI-Planning

AI-planning, or automated planning, is a branch of AI that consists of algorithms that do deliberate planning based on some sort of reasoning [14]. In this case, a plan is an ordered sequence of actions that takes the system from an initial state to a state where some goals are satisfied. The earliest AI-planner is the STRIPS planner [15] which models the problem as a search tree, where every reachable state is a node. The A-star algorithm [65] is then used to search the tree. In this case, the reasoning is simply to find the shortest plan possible with a guided search.

There are different ways to model a planning problem. One of the most common ways to do it is with the Planning Domain Definition Language (PDDL) [14], which is presented in Section 2.1.1. It can also be done as a task network [14], or as a proposition graph such as in Graphplan [66].

As of today, there has been developed a lot of different planners. None of them supports all requirements (which are explained in Section 2.1.1), but in this thesis the OPTIC planner is chosen. The necessary theory behind OPTIC is presented in Section 2.1.2, and the reason for choosing it is presented in Section 4.4.

2.1.1 PDDL

To model a planning problem we use the PDDL. There is a couple of different versions, but for our purpose we use PDDL2.1 [67]. A planner needs two PDDL documents to produce a plan. The first one is the domain file where the capabilities of the system are described, while the second is the problem file where a specific problem is described. The domain file we use is presented in Appendix E.

In a domain file we find requirements, types, predicates, functions and actions. The requirements are everything the planner needs to be able to handle to solve the problem. Several requirements exist, but for our case we need "durative-actions", which means that every action has a time duration, and "numeric-fluents", which means that we can use numbers. The types defined in the domain file are what kinds of objects that can exist. Examples of this could be "vessel" and "location".

Predicates are simple facts about the world that are either true or false. This can, for example, be that a specific vessel is at a specified location. Since we use "numeric-fluents" we can also use functions, which is a different way to state a fact about the world. A function in PDDL is a number that can be connected to an object. As an example, a function can say how many passengers a vessel has onboard. While predicates can be set to true or false by actions, functions can be increased or decreased by a number, or assigned to a number by actions. The number can either be constant or a combination of functions using addition, subtraction, multiplication or division.

An action takes a set of parameters and has preconditions and effects. Since we use "durative-actions" the actions also have a duration. A typical action is the move action, where a vessel moves between two locations. In this case the parameters are the vessel and the two locations. A condition for this action is that the vessel is at the location it travels from. An effect of the action is that the vessel is now at the location it travels to. When using "numeric-fluents" can conditions also be that a function has to be greater than, less than or equal to a number. At the same time, effects can be an increase, decrease or assignment of a function. The duration of an action can be a constant number, a function or a combination of functions. Another difference when using "durative-actions" is that the conditions can either be at the start, the end or overall. The effects can also either happen at the start or at the end. Looking at the move action, a condition at the start is still that the vessel is at the start location. An end condition can be that the location the vessel moves to is available, while an overall condition can be that the vessel is undocked. An effect at the beginning is that the vessel is no longer at the start location and an end effect is that the vessel is at the end location.

The problem file consists of objects, the initial state, the goals and since we use "numeric-fluents" there is also a metric. Objects are the specification of types. As an example, there can be two objects, "v1" and "v2", of the type vessel. The initial state is all predicates that are true in the initial state and all initial values of functions. The goal is all predicates that have to be true in an end state, and what values selected functions must be equal to, greater than or less than. Lastly, the metric is what the planner should minimize or maximize. This is either a single function or a combination of functions. If a metric is unspecified, the planner only attempts to find a plan with the shortest total duration. If it is specified, it tries to optimize the metric, as well as the total duration.

2.1.2 OPTIC and Its Predecessors

OPTIC [19] is an AI-planning algorithm that is an extension of another AI-planning algorithm called POPF [18]. POPF is again based on the AI-planner COLIN [68], which is based on CRIKEY3 [69], a newer version of CRIKEY [70]. In addition, POPF has two updates, namely Stochastic-POPF [71] and POPF2 [72], where OPTIC is based on POPF2. Since OPTIC uses concepts inherited all the way from CRIKEY, this subsection will explain some relevant concepts from all the predecessors of OPTIC, as well as what is new in OPTIC.

At the heart of AI-planning we find search techniques. CRIKEY introduced a search technique based on Enforced Hill-Climbing (EHC), introduced in the non-temporal AI-planner Fast Forward (FF) [73]. EHC sets the initial state as the current state, and then performs a search to find a state with a lower heuristic than the current state. The found state is set to the current state, and a new search is performed. In FF the search is a breadth-first search, while CRIKEY uses a best-first search.

As opposed to STRIPS, where the user has to find a heuristic function, FF makes a heuristic function automatically. To do this something called relaxed Graphplan is used as a reachability analysis. The first step in Graphplan is to expand a graph with reachable facts, here called propositions. In the first layer all propositions of the initial state are contained, in the next layer are all propositions possible to reach with one action, and so on [66]. In relaxed Graphplan it is not included that actions also delete propositions, which makes the expansion faster, and it means that the distance to the goals in the graph is an absolute minimum. In Metric-FF it is shown how this graph can include numerical values as well [74], and this is also used by

CRIKEY.

To use non-temporal search, CRIKEY divides temporal actions into something called snap-actions. This means that every action is split into a start action and an end action. The duration, d, of an action is enforced by ensuring that the end action is performed exactly at d time after the start action. In CRIKEY3 the duration is also used explicitly in relaxed Graphplan, as started actions are added as part of the state. This also resulted in the layers of the proposition graph having a timestamp, and the distance to the goals being described in time and not the number of actions.

Continuous effects on variables were supported from COLIN, where a Linear Programming (LP) solver is used to find a lower and upper bound on the numerical variables that are added to the proposition graph. The LP solver is also used to find the earliest start time an action can have, when dependent on a continuous variable.

To reduce something called early commitment, POPF makes partial-order plans, as opposed to total-order plans. Early commitment often happens in forward search as actions are chosen one at a time when searching toward the goal. This can lead to bad choices early on. In a partial order plan, the actions are not in a strict total order unless they have to. This can be intuitive in temporal planning as plans do not have to be a sequence, but rather independent actions with a start time and a duration. To do this, POPF adds the most recent achievers and deleters of every fact in a state, as well as the most recent changers of a numerical variable. An LP solver is then used to find the earliest timestamp this state can have in the proposition graph, and makes sure it is consistent with all temporal and numerical constraints.

In Stochastic-POPF two important functionalities are added. The first is the addition of a metric in the heuristic function, which was previously only the time to all goals were fulfilled. Now, the cost of each action with respect to change in the metric is added as well, by not including states reachable with a too large cost. What is considered a too large cost leads to the next change, which is the introduction of something called Any-Time Search. Just like Metric-FF, POPF will attempt to find a solution with weighted A* (WA*) from the initial state if EHC fails. WA* is a version of A* where a bias is used to prioritize states that are closer to the goal. In Stochastic-POPF the Any-Time search works by first performing a EHC search to try to find a solution. Regardless of the resulting solution, a WA* search is performed, where states that increase the cost above the previously lowest cost are not taken into account. When a new solution is found it is presented, and a new WA* search is performed. The search is stopped for good when all states have been visited.

Even though POPF is able to build partial-ordered plans with facts, the ordering of changes in numerical variables is still total. However, in POPF2 the state representation is changed to include a queue of desired effects on each variable. What is considered desirable is based on how the variable appears in the problem. If goals or preconditions of actions require that a variable is above a constant, higher values of the variable are desired. The queue is then used to check when a precondition for an action is met, as opposed to adding the action after the variable is no longer changed.

While POPF tries to find the shortest plan, OPTIC also facilitates the use of preferences. This can be goals or preconditions that are preferable to be satisfied, but do not have to be. It can also be that it is preferable that one fact become true after another. If the preference is not met, a user-defined cost is added to the total cost of a plan. In addition, OPTIC aims to support time-dependent goals, where being late to meet a deadline has an increasing cost. In order to achieve this, an automaton is made for each preference to track its status. The automaton's position is then added to the states in the problem. A dummy step is added to the plan every time the preconditions of a transition in the automaton are met. A MIP solver is then used to encode information about the dummy steps. This is done to minimize the total cost by finding the optimal timestamps of the dummy steps.

2.2 Vessel Modeling

This section presents the Society of Naval Architects and Marine Engineers (SNAME)[75] notation used to describe the state of a marine vessel. The general kinematic equations used for marine vessels, based on the theory from the Handbook of Marine Craft, Hydrodynamics and Motion Control by Thor I. Fossen [76], will then be presented. Lastly, the model we use for milliAmpere 1 is presented.

2.2.1 SNAME Notation

The SNAME notation is used to describe the state of a marine vessel, and an overview of the symbols used is shown in Table 2.1. In Figure 2.1 the different Degrees of Freedom (DOF) are shown. The positions and Euler angles are given with respect to the North, East, Down (NED) reference frame, which is shown in Figure 2.2. To denote the state of the vessel we use $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{\eta}^\top & \boldsymbol{\nu}^\top \end{bmatrix}^\top$, where $\boldsymbol{\eta}$ contains positions and Euler angles, and $\boldsymbol{\nu}$ contains velocities. In calm sea, heave,

Table 2.1: The SNAME notation used to describe the state of a marine vessel.

DOF	Forces and moments	Linear and angular velocities	Positions and Euler angles
surge	X	u	x
sway	Y	v	y
heave	Z	w	z
roll	K	p	ϕ
pitch	M	q	θ
yaw	N	r	ψ

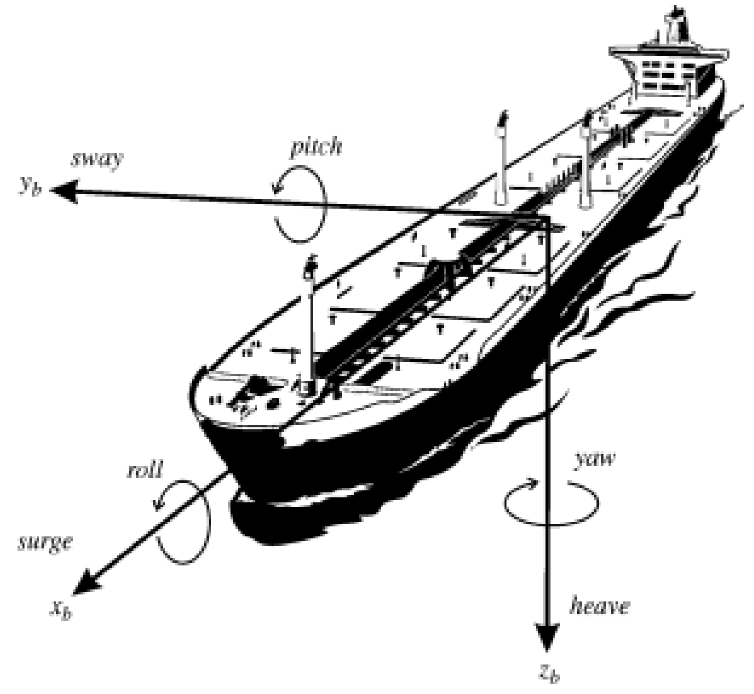


Figure 2.1: An illustration of the six DOF using the SNAME notation, taken from the Handbook of Marine Craft, Hydrodynamics and Motion Control by Thor I. Fossen [76].

roll and pitch can be approximated to zero. This is called the 3-DOF model, and is a useful simplification we will use.

2.2.2 Vessel Kinematics

The general kinematic equations for the 3-DOF model are given in Equation (2.1). $\boldsymbol{R}$ is the rotation matrix from body to NED, while $\boldsymbol{M}$ is the inertia matrix, $\boldsymbol{C}$ is the Coriolis matrix, and $\boldsymbol{D}$ is the dampening matrix. $\boldsymbol{\tau}$ is the input forces vector given by the vessel's actuators, and $\boldsymbol{\tau}_{env}$ is the forces on the vessel from the environment,

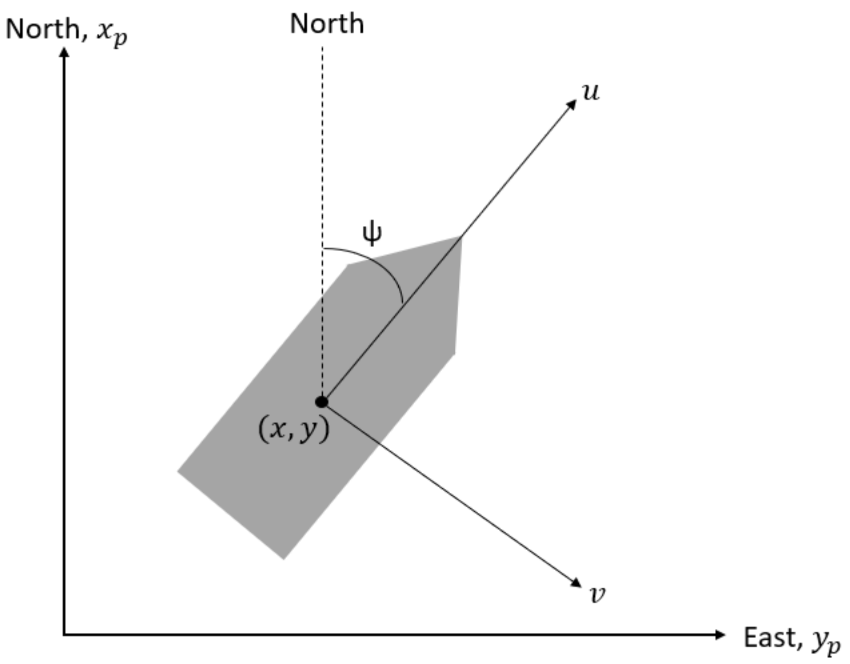


Figure 2.2: An example of a vessel position with respect to the NED reference frame.

such as wind and waves.

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\,\boldsymbol{\nu} \tag{2.1a}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\,\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\,\boldsymbol{\nu} = \boldsymbol{\tau} + \boldsymbol{\tau}_{env} \tag{2.1b}$$

2.2.3 Vessel Model milliAmpere 1

The model we use for milliAmpere 1 is the surge-decoupled model found in Optimization Based System Identification for the milliAmpere Ferry by Anders Aglen Pedersen [77]. It follows Equation (2.1), and the matrices are given in Equation (2.2), where the elements in $\boldsymbol{D}$ and all parameters are given in Appendix A.

$$\boldsymbol{M} = \begin{bmatrix} m_{11} & 0 & 0 \\ 0 & m_{22} & m_{23} \\ 0 & m_{32} & m_{33} \end{bmatrix} \tag{2.2a}$$

$$\boldsymbol{C} = \begin{bmatrix} 0 & 0 & -m_{22}v - m_{23}r \\ 0 & 0 & m_{11}u \\ m_{22}v + m_{23}r & -m_{11}u & 0 \end{bmatrix} \tag{2.2b}$$

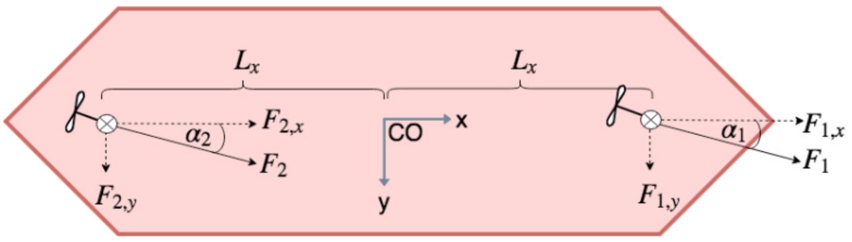


Figure 2.3: Thruster configuration of milliAmpere 1, taken from Control allocation for double-ended ferries with full-scale experimental results [63].

$$\boldsymbol{D} = \begin{bmatrix} d_{11}(\boldsymbol{\nu}) & 0 & 0 \\ 0 & d_{22}(\boldsymbol{\nu}) & d_{23}(\boldsymbol{\nu}) \\ 0 & d_{32}(\boldsymbol{\nu}) & d_{33}(\boldsymbol{\nu}) \end{bmatrix} \tag{2.2c}$$

The milliAmpere 1 has two azimuth thrusters located $L_x = 1.8m$ from the vessel's center. An illustration is shown in Figure 2.3. Each thruster can move around 360°, and has a maximum thrust of $T_{max} = 500N$. Since this makes the vessel too slow for our case $T_{max} = 1500N$ is used instead. Because it takes some time for the azimuth thrusters to rotate, the front thruster is only used backward and sideways. This means that the angle α_1, which can be seen in Figure 2.3, is limited to $\left[-\pi, -\frac{\pi}{2}\right] \cup \left[\frac{\pi}{2}, \pi\right]$. Likewise, the back thruster is not used backward, such that $\alpha_2 \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. The input vector to the vessel is denoted $\boldsymbol{u} = \begin{bmatrix} T_1 & T_2 & \alpha_1 & \alpha_2 \end{bmatrix}^\top$.

The forces from the two thrusters, F_1 and F_2, can be decomposed into $F_{1,x}$, $F_{1,y}$, $F_{2,x}$ and $F_{2,y}$, as can be seen in Figure 2.3. This gives us the decomposed force vector $\boldsymbol{f} = \begin{bmatrix} F_{1,x} & F_{1,y} & F_{2,x} & F_{2,y} \end{bmatrix}^\top$. Furthermore, $\boldsymbol{f}$ relates to the input force vector $\boldsymbol{\tau} = \boldsymbol{B}\boldsymbol{f}$, where $\boldsymbol{B}$ is given in Equation (2.3).

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & L_x & 0 & -L_x \end{bmatrix} \tag{2.3}$$

2.3 Optimal Control

Optimal control is used to find an optimal trajectory and control sequence for a vehicle, by using mathematical optimization. In this section, we will first look at how this can be modeled with continuous dynamics, and then we show how the problem can be solved with a direct collocation method.

2.3.1 Continuous Optimal Control Problems

An OCP can be formulated as in Equation (2.4) [78, 79].

$$
\begin{aligned}
\min_{\boldsymbol{x}(\cdot),\boldsymbol{u}(\cdot)} \quad & \int_{t_0}^{t_0+T} L\left(\boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right)\right) dt && \text{(2.4a)}\\
\textit{subject to} \quad & \dot{\boldsymbol{x}}\left(t\right) = \boldsymbol{f}\left(\boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right)\right), \quad t \in [t_0, t_0+T] && \text{(2.4b)}\\
& \boldsymbol{x}\left(t_0\right) = \bar{\boldsymbol{x}}_0 && \text{(2.4c)}\\
& \boldsymbol{h}\left(\boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right)\right) \leq \boldsymbol{0}, \quad t \in [t_0, t_0+T] && \text{(2.4d)}
\end{aligned}
$$

$\boldsymbol{x}$ is the state vector and $\boldsymbol{u}$ is the control input. In the objective function, in Equation (2.4a), the function L is the loss function we want to minimize for each time instance from t_0 to $t_0 + T$. The objective function is constrained by the state dynamics given by the function $\boldsymbol{f}$ in Equation (2.4b), as well as the initial state $\bar{\boldsymbol{x}}_0$, in Equation (2.4c). Lastly, the trajectory is constrained by a function $\boldsymbol{h}$ in Equation (2.4d).

2.3.2 Direct Collocation Method

To solve an OCP a direct collocation method can be used. The idea behind direct methods is to discretize the whole system into N steps, such that it is possible to solve it using finite-dimensional nonlinear programming. This means that instead of looking at the continuous trajectory $\boldsymbol{x}\left(\cdot\right)$, we look at a finite number of discretized states $\boldsymbol{x}_k$ and inputs $\boldsymbol{u}_k$ at times t_k, where $k = 0, \ldots, N$.

Each collocation interval $[t_k, t_{k+1}]$ is divided into a set of collocation points $t_{k,i}$, where $i = 0, \ldots, d$. For each collocation interval we have a polynomial $\boldsymbol{p}_k\left(t, \boldsymbol{v}_k\right)$ which approximates the state trajectory. $\boldsymbol{v}_{k,i}$ is then the approximation of the state at time $t_{k,i}$. We can then add the constraint given in Equation (2.5) [80].

$$
\boldsymbol{c}\left(\boldsymbol{v}_k, \boldsymbol{x}_k, \boldsymbol{u}_k\right) = \begin{bmatrix} \boldsymbol{v}_{k,0} - \boldsymbol{x}_k \\ \dot{\boldsymbol{p}}_k\left(t_{k,1}, v_k\right) - \boldsymbol{f}\left(\boldsymbol{v}_{k,1}, \boldsymbol{u}_k\right) \\ \vdots \\ \dot{\boldsymbol{p}}_k\left(t_{k,d}, v_k\right) - \boldsymbol{f}\left(\boldsymbol{v}_{k,d}, \boldsymbol{u}_k\right) \end{bmatrix} = \boldsymbol{0} \tag{2.5}
$$

To make sure the trajectory is consistent over all intervals we also have to add the constraint given in Equation (2.6).

$$\boldsymbol{p}_k\left(t_{k+1}, \boldsymbol{v}_k\right) - \boldsymbol{x}_{k+1} = \mathbf{0} \tag{2.6}$$

Lastly, the objective function over each collocation interval, $\int_{t_k}^{t_{k+1}} L\left(\boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right)\right) dt$, has to be approximated by a quadratic formula using the same collocation points. We denote this term $l_k\left(\boldsymbol{v}_k, \boldsymbol{x}_k, \boldsymbol{u}_k\right)$. This results in the nonlinear programming problem in Equation (2.7).

$$\begin{aligned}
\min_{\boldsymbol{v},\boldsymbol{x},\boldsymbol{u}} \quad & \sum_{k=0}^{N-1} l_k\left(\boldsymbol{v}_k, \boldsymbol{x}_k, \boldsymbol{u}_k\right) && && \text{(2.7a)}\\
\textit{subject to} \quad & \boldsymbol{x}_0 - \bar{\boldsymbol{x}}_0 = \mathbf{0} && && \text{(2.7b)}\\
& \boldsymbol{c}\left(\boldsymbol{v}_k, \boldsymbol{x}_k, \boldsymbol{u}_k\right) = \mathbf{0}, && k = 0, \ldots, N-1 && \text{(2.7c)}\\
& \boldsymbol{p}_k\left(t_{k+1}, \boldsymbol{v}_k\right) - \boldsymbol{x}_{k+1} = \mathbf{0}, && k = 0, \ldots, N-1 && \text{(2.7d)}\\
& \boldsymbol{h}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) \leq \mathbf{0}, && k = 0, \ldots, N-1 && \text{(2.7e)}
\end{aligned}$$

Chapter 3

Methodology

In this chapter the system and the world it operates within are presented. An overview of the system is given in Figure 3.1, where the control block is further expanded in Figure 3.2. The planner node uses AI-planning to generate plans, and is presented in Section 3.2. For control, the trajectory planner described in Section 3.6 and the control allocation described in Section 3.8 use optimization, while the DP-controller presented in Section 3.7 use a simple PID controller. The vessel nodes are an implementation of the model in Section 2.2.3. Lastly, the plan delegation node and plan executor nodes presented in Section 3.3 and Section 3.4, as well as the task interpreter nodes presented in Section 3.5, use knowledge about the world and translate between the planner and the control system.

3.1 Map and Passenger Distribution

In this section the modeling of the relevant environment is presented. First, the map of the area and which ports are used are shown, and the information about the area given to the system is presented. Then the distribution of passengers is presented in detail.

3.1.1 Map of the Area

A map of the northern part of Trondheim is shown in Figure 3.3, where the three ports that passengers can travel from are marked. The eastern part of the operational area is the outlet of the river Nidelva. However, we assume that the effect of the water stream is low, such that the environmental forces can be omitted. For the

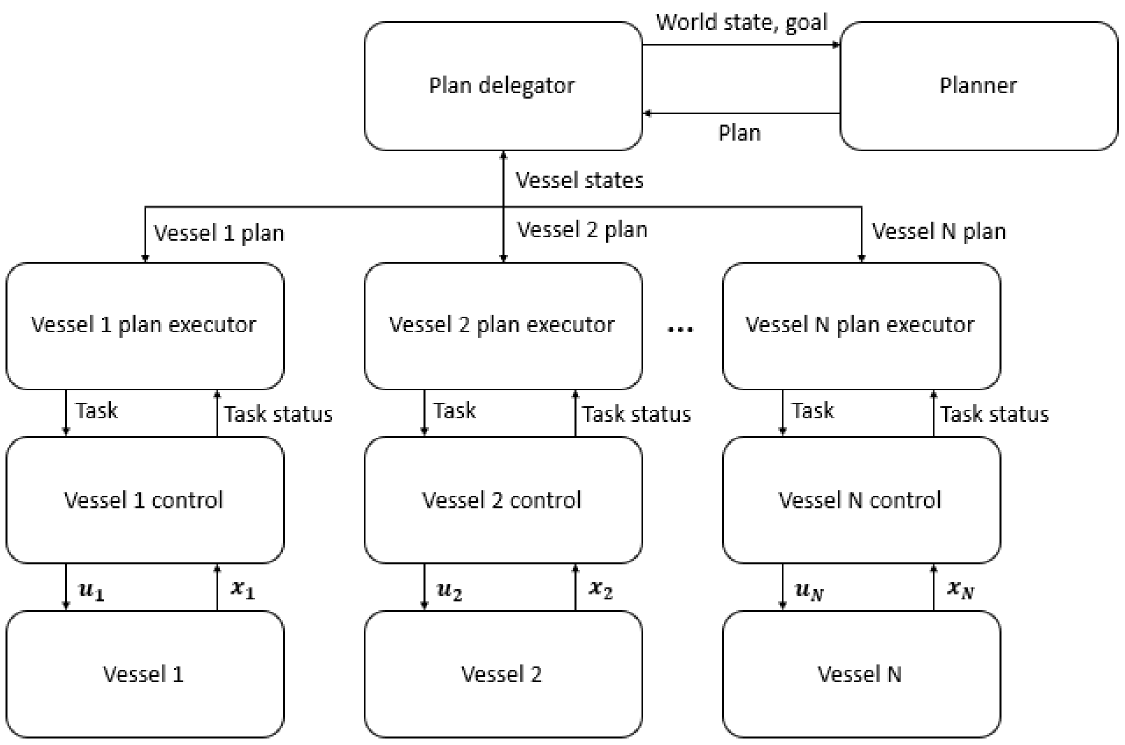


Figure 3.1: Overview of the system. The vessel model is presented in Section 2.2.3, and the control block is expanded in Figure 3.2.

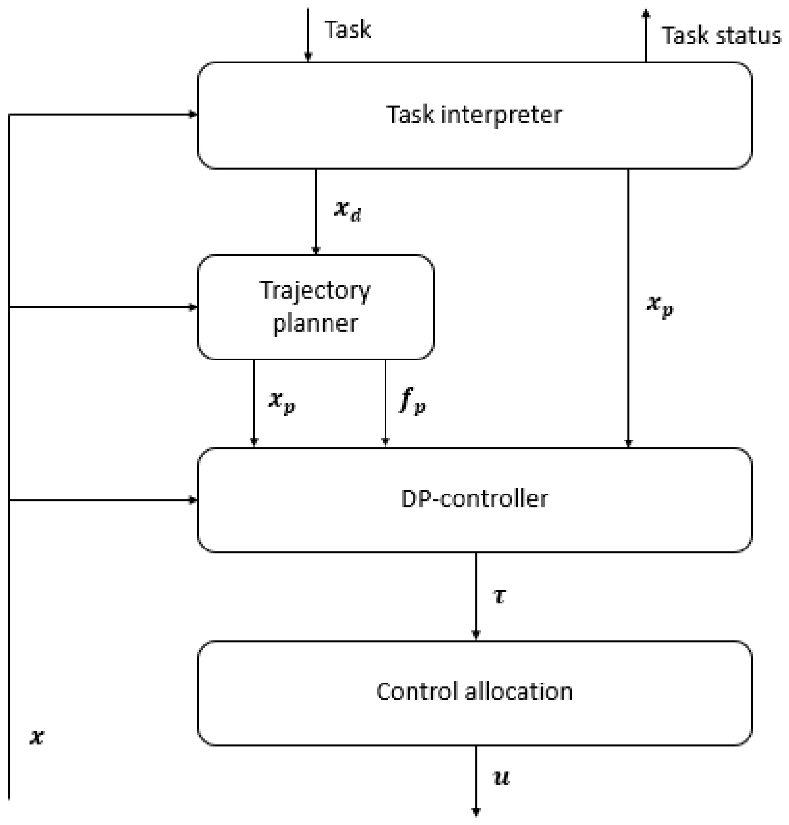


Figure 3.2: An overview of the control system of a single vessel. This is an expansion of the control block in Figure 3.1.

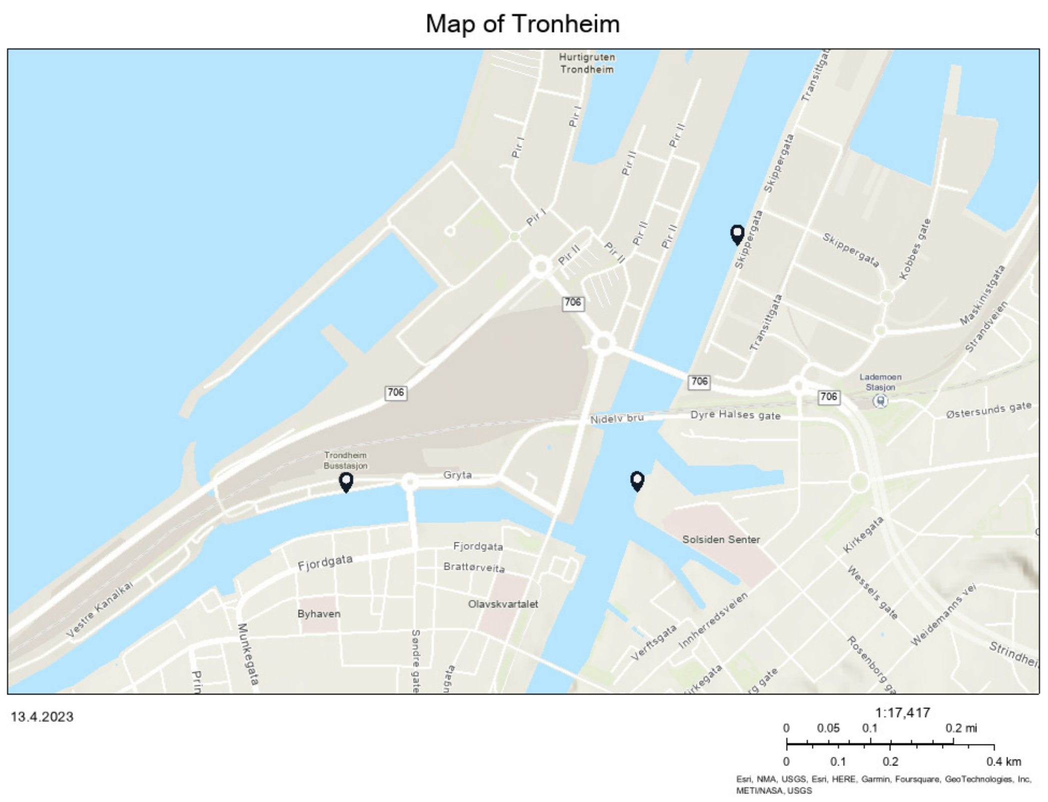


Figure 3.3: A map of Trondheim taken from ArcGIS. Marked to the North is the port for Nyhavna, to the west is Trondheim central station (Trondheim S) and in the middle is Solsiden.

task interpreter, described in Section 3.5, several overlapping convex areas have been made manually. This is used during transit between ports to ensure a collision-free path. Since the vessel always operates within the same area, where there are few changes in static obstacles, the convex areas do not have to be made dynamically. As long as the vessels are within these areas there are no collisions with static objects.

The possibility of collisions with each other, or other vessels is omitted for simplicity. Since the channels are wide enough for two of our vessels to pass each other, the convex areas could be split in two to allow collision-free passing. The ports could also be made bigger in order to fit more vessels next to each other. However, the overall performance of the system would probably be very similar, but this is something that should be looked at in future work. Both because collision avoidance is an important aspect of an autonomous vessel, but also because the planner might increase efficiency if a delay is detected.

3.1.2 Passenger Distribution

To add passengers for the system to serve, we have to make some assumptions. First, we simplify and say that every minute a number of passengers appear at each port ready to use the service. The number is taken randomly from a Poisson distribution. We use this distribution because it is a discrete non-negative distribution. The expected value of the Poisson distribution is found from the expected number of people traveling the given hour, the given trip and the time to departure.

Where passengers want to travel to and from is decided by several factors. Before the service is potentially launched and run for some time, we have to guess the distribution. Firstly, we have assumed that there are mostly living areas at Nyhavna. Secondly, we know that Solsiden is more accessible with other methods of transportation from both the other ports. This makes it a little less relevant to our service. Our full guess is shown in Table 3.1. What time of day passengers travel is also hard to guess before launching a service, but we use the average distribution for all AtB services given in Figure 3.4.

Table 3.1: A guess on how passengers will be distributed among the possible trips in the morning rush (7:00-9:00), evening rush (14:00-17:00) and the rest of the day.

Trip	**Morning rush**	**Evening rush**	**Outside rush hours**
Nyhavna to Solsiden	14%	1%	15%
Nyhavna to Trondheim S	46%	10%	30%
Solsiden to Nyhavna	1%	14%	5%
Solsiden to Trondheim S	9%	20%	5%
Trondheim S to Nyhavna	10%	46%	30%
Trondheim S to Solsiden	20%	9%	15%

The timetable for the ferries is given in Appendix D. We think passengers are most likely to come to the ports around when a ferry leaves. However, since we do not know how long before departure passengers arrive, we approximate it linearly. The idea is to have a coefficient, tt, that is multiplied with the expected number of people for a given trip at a given minute. An illustration is shown in Figure 3.5. We assume passengers are most likely to come 2 minutes before the ferry departs, and least likely to come 2 minutes after. The likelihood is then increasing and decreasing linearly between these points. This has to be done around 1, such that the total likelihood doesn't increase or decrease. As an example, we assume that we expect 10 passengers per minute. When using the coefficients from Figure 3.5, instead of expecting 10 passengers every minute we expect 1 passenger at the low point and 19 at the high point. This of course adds up to $2 \cdot 10$ passengers, such that the total

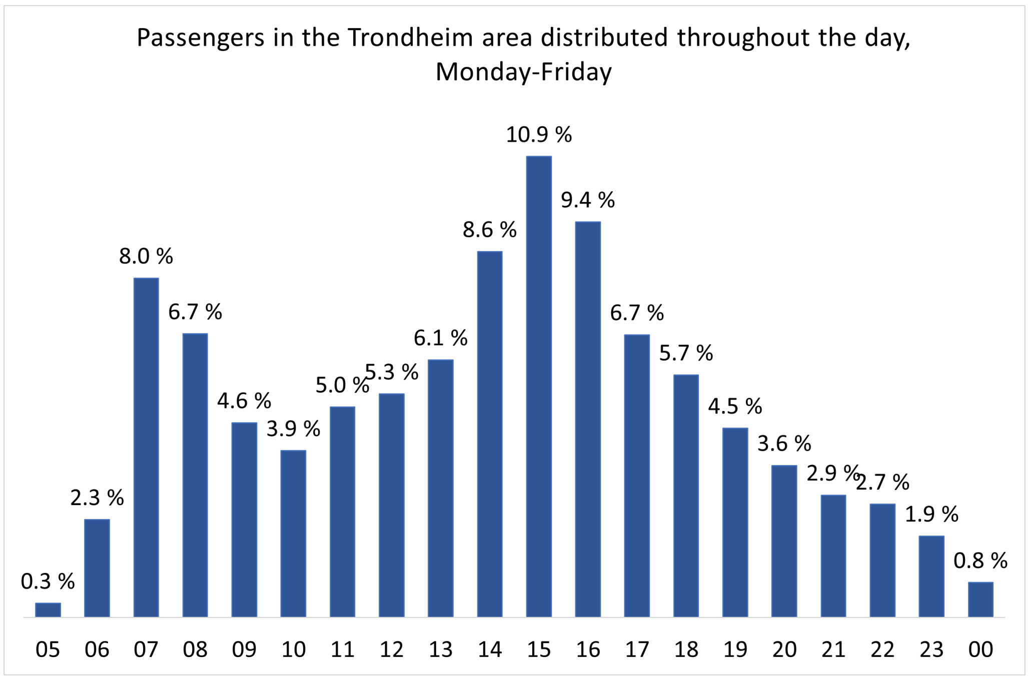


Figure 3.4: Distribution of passengers throughout the day. Provided by AtB.

expected number is unchanged.

The resulting expected value, λ, is then given in Equation (3.1). Here P is the expected total amount of people traveling that day. h is the distribution throughout the day. $mh = 60$ is minutes in an hour. td_h is the distribution for each trip that hour. tt is the time to departure coefficient. The normal distribution, denoted $normal$, is used to give an extra degree of randomness which can be set by the developer by adjusting the variance σ^2.

$$\lambda = \frac{normal(P * h, \sigma^2)}{mh} \cdot td_h \cdot tt \tag{3.1}$$

3.2 Planner

The planner node has one task, and that is to take the state of the world and what we want to achieve, and then produce a plan. First, the planner transforms the information to a PDDL document (described in Section 2.1.1). Then it calls the OPTIC planner (described in Section 2.1.2). Lastly, it returns the resulting plan in a readable format. In this section the representation of the world state, the goals and actions are presented. While PDDL is a quite expressive language, the planners have some trouble with solving more complicated problems. Therefore, in Section 4.4, some limitations we have encountered, and the reason OPTIC was

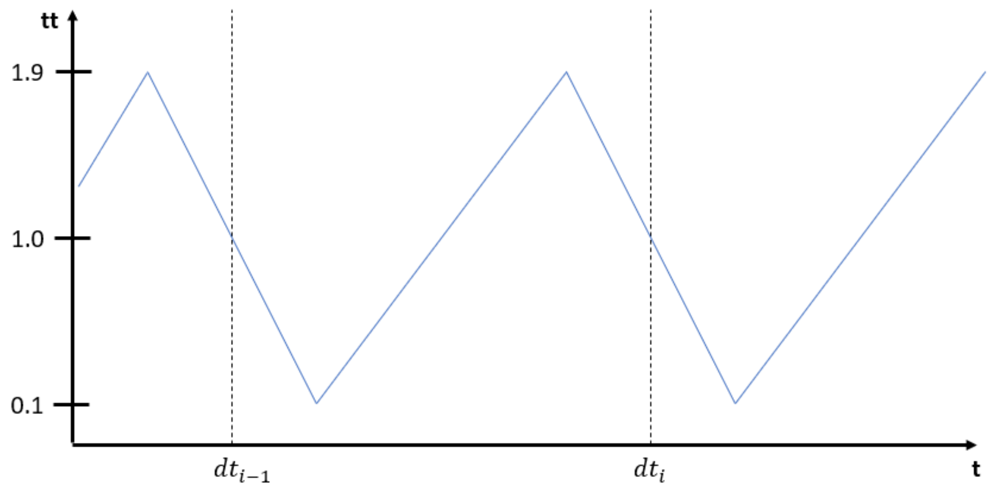


Figure 3.5: An illustration of the timetable coefficient, *tt*, used to push the expected amount of people towards the departure time denoted *dt*. It goes from 0.1 to 1.9, which is symmetrical around 1. This means that the total amount of expected passengers is unchanged.

chosen, are presented. The planning domain used is shown in Appendix E.

3.2.1 World State

The physical objects in the world state are vessels, ports and possibly also passengers. The vessel can either be at a port or between two ports. If there are any passengers onboard when the planner starts they are added, together with what vessel they are on and where they want to go.

In addition, the planner receives a number of routes that have to be covered. How many are given by the plan delegation described in Section 3.3. The routes are either planned routes following the timetable given in Appendix D, or predicted extra needed capacity between two ports. Since Solsiden is between the other ports, the possible extra routes are between Solsiden and one of the other ports, or taking all three ports. While the planned routes have a start port, the extra routes can start from any end.

3.2.2 Goals

Supporting priorities in goals is a requirement in PDDL that OPTIC is said to support. However, we were not able to make it work with the available version. Therefore, all the goals we add have to be possible to reach at the same time for the planner to find a solution. This means that we try to not add more goals than necessary. However, some goals are essential. The most important goals are to deliver passengers already onboard to their desired destination, cover the planned routes and make sure the vessel is either taking a route or docked at a port. Lastly, the extra routes are stated such that there is a capacity that needs to be covered. It is up to the plan delegation, described in Section 3.3, to give the planner a capacity it can manage to cover. Since we want to minimize travel time, we include this as a metric.

3.2.3 Actions

The simplest actions are the two move actions, where there is one for moving between two ports and one for moving to a port. Going between two ports has a duration based on tests of how long it takes from starting the undocking to completing the docking. To simplify, the time it takes to move to a port is half of what it takes to move between the two ports the ferry is between. Next, is the "let passengers off" action, which is mostly used in the planner to set the predicate "reachedDestination" for the onboard passengers. In theory, it could be omitted because the vessels open their doors when reaching a port to let off passengers anyway. However, there are no if-statements in PDDL, so the "reachedDestination" predicate can either be set by a move action all of the time or not at all. Since the vessel can move without passengers, we need a separate action to set this predicate. The action is always 15 seconds long.

Taking a route can be divided into move, let off passengers and take passengers onboard actions in PDDL. However, as described in Section 4.4, this has several challenges. Therefore, we look at taking a route as an effect and not an action itself. This can be done because once the route has begun, there is no planning involved. The vessel just goes from port to port following a route. To take a route the vessel only has to be at the port where the route starts, and not be occupied with another route. The action also has to use the two next ports to indicate that onboard passengers can get out. Since we only have three ports we only need the next two, but this can become a problem with more ports.

When switching from one to two routes after 6 o'clock there are two routes starting at Nyhavna. The first start at 06:02 and the second at 06:12. To make sure the first route is taken by the closest vessel, we introduce some turn functions. Each route has a turn and cannot be taken before the previous has been delegated to a vessel. This could become a problem if routes are starting at different locations at the same time. However, the routes are made such that when we are replanning, a vessel is either at the starting port or on its way toward the port. If a recharging scheme is introduced, where a vessel might have to go to recharge, this might not hold.

The actions for taking extra routes are similar to taking a planned route, except for the fact that the vessel can start at any point on the route. Since taking a route between Solsiden and another port means that there are twice as many departures as taking all three ports, the amount of capacity covered is twice as much. Therefore, we have one action for taking all three ports and one for going between two ports.

3.3 Plan Delegator

The plan delegator node is monitoring the whole system, and decides when it is time to replan. Since there are not added any events where replanning is triggered, there is a replanning every hour. Both the predicted number of passengers and rush time status are changed every hour, so this is a sensible frequency to replan. The timetable does not have a set frequency throughout the day, and the frequency changes also happen around the same time.

When it is time to replan, the plan delegator gets the status of each vessel from the plan executors. It needs to know if the vessels are at a port or between two ports, and if there are any passengers onboard. The plan delegator also reads the timetable and decides if there are one or two routes, and where they start. Lastly, it looks at the predicted number of passengers, and calculates the extra needed capacity for each trip. In this particular case, a trip refers to the stretches between Nyhavna and Solsiden, and Trondheim S and Solsiden. If a passenger is going between Nyhavna and Trondheim S, it takes both trips. To find the needed capacity the plan delegator first finds how many passengers that are expected to take each of the two trips. Then it divides the number by three, to get the number per 20 minutes, which is the time it takes to go a full round. Then it finds how many full ferries this corresponds to. This gives a form of unit capacity. Taking a timetable route means that a ferry covers one unit capacity for both trips. The remaining capacity has to be filled by any remaining ferries. Since the planner will crash if it receives an unsolvable problem, the plan delegator also makes sure the extra capacity is manageable. The goals given to the planner are described in Section 3.2.2, and the resulting plan is split into actions for each vessel.

3.4 Plan Executor

Each plan executor receives a plan in the form of a sequence of actions relevant to the given vessel. This plan is split into a sequence of simpler tasks that are given to the controller, one at a time. There are 5 types of tasks. The first is transit between two ports, which requires the vessel to be undocked. The second is therefore to undock. In a real system this would include things like closing the doors, but in the simulation, this is a rather short action where the vessel is just given some initial thrust. The third task is docking, which is simply the last part of the trip between two ports. In real life this task would include opening the doors. The fourth task is to load and unload passengers. This task is as simple as standing still with the doors open until it is time to go, which is either after a minimum of $15s$ or until the timetable says the vessel should leave. A direction is also added to make sure the passengers know where the ferry is headed. The last task is to wait and stand still.

The move actions are translated to a task sequence of undock, transit and dock, and if it is not a move between ports action the undock task is omitted. If this is the last action for this vessel in the plan, there is also a load task, without any direction. This is added to make sure that any undetected passengers are let off. A "let passengers off" action is translated to the same task. Taking a planned route is translated to a repeating sequence of load, undock, transit and dock tasks. Where the load action is given a time and direction based on the timetable. The taking extra capacity actions are translated to the same sequence except it always has a loading time of the minimum of $15s$.

3.5 Task Interpreter

The task interpreter receives a task, as described in Section 3.4, and translates it to references to the controller. For the transit and docking tasks this is a desired state given to the trajectory planner, while for the load and wait tasks the DP-controller is given a state to track directly. The start of the transit task is the same as the last part of an undocking phase in the sense that the vessel leaves a dock. Therefore, the undocking task is very short and only includes giving the vessel some thrust forward. It could have been omitted, but it is used to say that the vessel is undocked. It also works like a placeholder for other functions important for a real system, such as closing the doors. When the task interpreter detects that a task is completed, it requests another task. If there are no new tasks, it receives a wait task until there is a new plan available.

The desired state, $\boldsymbol{x_d} = \begin{bmatrix}\boldsymbol{\eta_d}^\top & \boldsymbol{\nu_d}^\top\end{bmatrix}^\top$, given to the trajectory planner is different for the docking and transit task. For the docking task, $\boldsymbol{\eta_d}$ is the dock of a port and $\boldsymbol{\nu_d}$ is the zero vector. The transit task is a little more complex. First, the task interpreter runs a depth-first search to find a path of overlapping convex areas, as described in Section 3.1.1, between two ports. These convex areas have a connection point between them, which we use as $\boldsymbol{\eta_d}$. Since the vessel should continue after reaching a connection point, $\boldsymbol{\nu_d}$ is set to $\begin{bmatrix}3 & 0 & 0\end{bmatrix}^\top$, which is around the vessel's maximum velocity. The vessel has reached the next area when it is within $9m$ of the connection point, and the task is done when reaching the last area. For the docking task, it is done when the vessel is within $30cm$ of the docking point, and with a heading no more than $5°$ from the desired heading.

The state, $\boldsymbol{x_p} = \begin{bmatrix}\boldsymbol{\eta_p}^\top & \boldsymbol{\nu_p}^\top\end{bmatrix}^\top$, given to the DP-controller is often the same for both the load and wait tasks. One of the goals of the planner is to make sure no vessel is undocked without performing a task, so the controller should not be given a wait task without being at a dock. For both tasks, the vessel should stand still, so $\boldsymbol{\nu_p}$ is the zero vector. $\boldsymbol{\eta_p}$ is the given dock for the load task, and the vessel's state at the beginning of the task for the wait task. The load task is done after the time given in the task by the plan executor. Since the wait task is always the last, it is not done until the planner replans.

3.6 Optimal Trajectory Planning

This section presents how optimization is used to generate an optimal trajectory for a milliAmpere 1 vessel. First, the problem is defined and then the cost function is described. Then the constraints given by the model of the vessel, max thrust for the thrusters, bounds for the states, and the area the vessel operates within are described. The trajectory planning module is based on Autonomous docking using direct optimal control [43], and Trajectory Planning and Control for Automatic Docking of ASVs with Full-Scale Experiments [44], where the latter also uses the milliAmpere 1. To solve the OCP we use direct collocation, as described in Section 2.3, using the open-source tool CasAdi.

As long as there are no unforeseen events making it impossible, we want the vessel to follow the same trajectory every time it goes between two ports. Therefore, the trajectory planning presented in this section is done once between each port and recorded. This means that the trajectory planner block in Figure 3.2 usually works as a ”database” giving recorded states and inputs.

3.6.1 Problem Definition

What we want from the trajectory planner is a sequence of optimal states with corresponding inputs that take the vessel from its current position towards a desired state $\boldsymbol{x}_d$. The trajectory is discretized, meaning that each discrete state is a variable in the optimization. This is also the case for the input. The timestep in the discretization is a tunable variable. A longer timestep increases speed, while a shorter one increases accuracy. By some trial and error, a timestep of $2s$ seemed to give the best results. The time horizon is also a tunable variable where a longer horizon gives better accuracy, and a shorter decrease runtime. Just like in Trajectory Planning and Control for Automatic Docking of ASVs with Full-Scale Experiments [44] the trajectory planner calculates a trajectory every $10s$. A time horizon of $20s$ then seemed to both give good accuracy and reasonable runtime. This gives the variable vector given in Equation (3.2), where $N = 10$ and the decomposed input vector is explained in Section 2.2.3.

$$\boldsymbol{w} = \begin{bmatrix} \boldsymbol{x}_0^\top & \boldsymbol{x}_1^\top & \dots & \boldsymbol{x}_N^\top & \boldsymbol{f}_0^\top & \boldsymbol{f}_1^\top & \dots & \boldsymbol{f}_{N-1}^\top \end{bmatrix}^\top \tag{3.2}$$

3.6.2 Cost Function

The loss function we use in the cost function in Equation (2.4) is shown in Equation (3.3). This is based on the loss function in Autonomous docking using direct optimal control [43], but with some modifications. The cost function from Trajectory Planning and Control for Automatic Docking of ASVs with Full-Scale Experiments [44], which also uses milliAmpere 1, has also been tested, with similar results. However, the chosen loss function is used because the tuning is slightly more intuitive.

$$L(\boldsymbol{x}, \boldsymbol{f}) = ||\boldsymbol{\eta} - \boldsymbol{\eta}_d||_{\boldsymbol{Q}_{\boldsymbol{\eta}}} + ||\boldsymbol{\nu} - \boldsymbol{\nu}_d||_{\boldsymbol{Q}_{\boldsymbol{\nu}}} + ||\boldsymbol{f}||_{\boldsymbol{R}_{\boldsymbol{f}}} \tag{3.3}$$

One modification is that the loss function originally had a configuration loss to ensure an even distribution of thrust for the different thrusters. Because one thruster brakes and one thruster drives forward in milliAmpere 1, as described in Section 2.2.3, this is not as relevant for this case. Another modification is the addition of $\boldsymbol{\nu}_d$. This is done because the trajectory planner is both used for docking and transit between docks. For the docking cases $\boldsymbol{\nu}_d$ is simply equal to zero, but in transit $\boldsymbol{\nu}_d = \begin{bmatrix} 3 & 0 & 0 \end{bmatrix}^\top$ to ensure that the vessel is not braking too much when it is about to enter a new area. The last modification is not shown in Equation (3.3) for simplification of the expression. When calculating $\psi - \psi_d$ we take the smallest signed angle operation of the result. The smallest sign angle operation is shown in Equation (3.4). This is to ensure that the vessel turns in the right direction to get to the desired angle. However, the modulo operation is not a differentiable function meaning that the solver we use has to be able to handle MIP. The *ssa* function is also used every time two states are compared in the direct collocation method.

$$ssa(x) = modulo(x + \pi, 2\pi) - \pi \tag{3.4}$$

The weight matrices $\boldsymbol{Q_\eta}$, $\boldsymbol{Q_\nu}$ and $\boldsymbol{R_f}$ are diagonal matrices, and is presented in Equation (3.5).

$$\boldsymbol{Q_\eta} = diag\left\{10^4, 10^4, 10^6\right\} \tag{3.5a}$$

$$\boldsymbol{Q_\nu} = diag\left\{10^2, 10^5, 10^7\right\} \tag{3.5b}$$

$$\boldsymbol{R_f} = diag\left\{10^{-2}, 10^{-2}, 10^{-2}, 10^{-2}\right\} \tag{3.5c}$$

3.6.3 Vessel Model Constraints

The model we use for the vessel in the trajectory planner is slightly different from the milliAmpere 1 model presented in Section 2.2.3, and is taken directly from Trajectory Planning and Control for Automatic Docking of ASVs with Full-Scale Experiments [44]. The full model is presented in Appendix B, but the kinematic equations are also given in Equation (3.6). This model is a simpler model than the actual vessel model. Another change is the S matrix, which is added to make the trajectory a little slower and easier to track.

$$\dot{\boldsymbol{\eta}}_p = \boldsymbol{R}(\psi_p)\,\boldsymbol{\nu}_p \tag{3.6a}$$

$$\boldsymbol{S}\boldsymbol{M}_p\dot{\boldsymbol{\nu}}_p + \boldsymbol{C}_p(\boldsymbol{\nu}_p)\,\boldsymbol{\nu}_p + \boldsymbol{D}_p(\boldsymbol{\nu}_p)\,\boldsymbol{\nu}_p = \boldsymbol{\tau}_p \tag{3.6b}$$

This gives us the state dynamics given in Equation (3.7).

$$\dot{\boldsymbol{x}}_p = \boldsymbol{g}\left(\boldsymbol{x}_p, \boldsymbol{f}_p\right) = \begin{bmatrix} \boldsymbol{R}\left(\psi_p\right)\boldsymbol{\nu}_p \\ \left(\boldsymbol{S}\boldsymbol{M}_p\right)^{-1}\left(-\boldsymbol{C}_p\left(\boldsymbol{\nu}_p\right)\boldsymbol{\nu}_p - \boldsymbol{D}_p\left(\boldsymbol{\nu}_p\right)\boldsymbol{\nu}_p + \boldsymbol{B}\boldsymbol{f}_p\right) \end{bmatrix} \tag{3.7}$$

Since the thrusters can only produce up to T_{max} thrust, this could be added as a constraint as shown in Equation (3.8). However, adding this constraint sometimes results in the solver having problems with finding a feasible solution. The constraint is therefore omitted. Because of the bounds on the variables, which will be described shortly, this constraint tends to not be violated and when it is violated, it is not severely violated.

$$F_x^2 + F_y^2 \leq T_{max}^2 \tag{3.8}$$

All variables have to be given a lower and upper bound. For $\boldsymbol{\eta}$ the position will already be bounded by a convex area as can be seen in Section 3.6.4, and ψ is bounded by the *ssa* function described in Section 3.6.2. Therefore, $\boldsymbol{\eta}$ can simply be bounded by $\pm\infty$ for every timestep. $\boldsymbol{\nu}$ will be bounded by the maximum velocity the vessel can achieve. However, to force the trajectory to be smoother $\boldsymbol{\nu}$ is bounded between $-\begin{bmatrix}1 & 1 & 0.2\end{bmatrix}^\top$ and $\begin{bmatrix}6 & 1 & 0.2\end{bmatrix}^\top$ for every timestep. To include the drive/brake configuration explained in Section 2.2.3, $F_{1,x}$ is bounded to be negative and $F_{2,x}$ is bounded to be positive. The bounds on $F_{1,y}$ and $F_{2,y}$ are set to $\pm\frac{3T_{max}}{5}$ to limit powerful sideways movements.

3.6.4 Harbor Layout Constraints

To make sure the vessel avoids collisions with static obstacles, convex sets are made. As long as the whole vessel is within the convex set there are guaranteed to be no collisions. An illustration of a convex set can be seen in Figure 3.6. Expressed as constraints this gives us Equation (3.9) [43]. $\mathbb{S}_b^{NED}$ is the set of the NED positions of all corners of the vessel. Since the positions of the corners are given by the position of the vessel, Equation (3.9) is rewritten as Equation (3.10). Here $\boldsymbol{\eta} = \begin{bmatrix}x & y & \psi\end{bmatrix}^\top$ is part of the vessel's state vector, $\boldsymbol{R_2}$ is the two-dimensional rotation matrix and $\mathbb{S}_b^b$ is the positions of the vessel's corners from the center of the vessel. For both sets of $\mathbb{S}_b$ there are added some margins such that the vessel appears 10% bigger. This is illustrated as dashed lines in Figure 3.6.

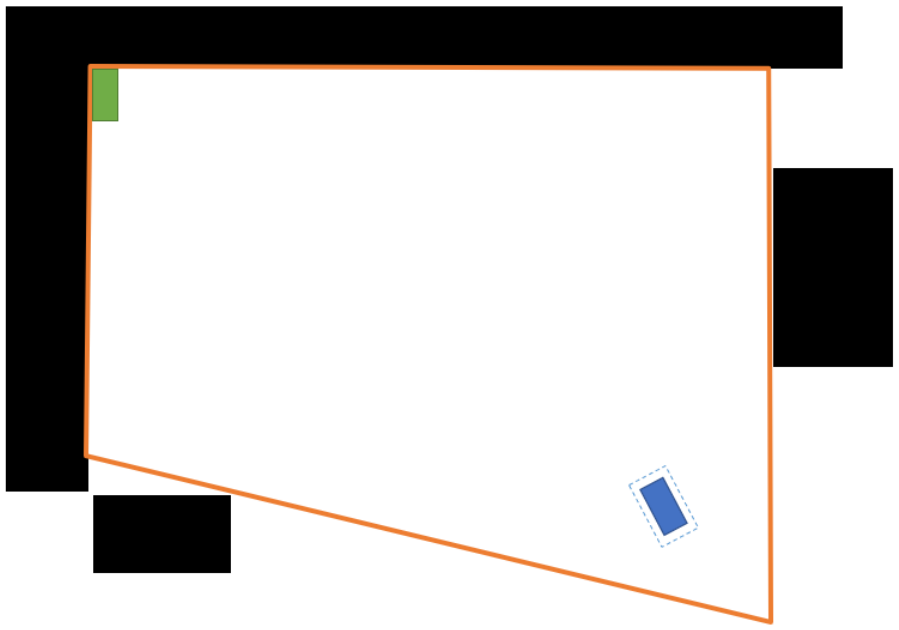


Figure 3.6: An illustration of a convex area. The black boxes are static obstacles. The orange polygon is the convex area, where it is safe to be within. The blue rectangle is the vessel, and the dashed line around it is the vessel with some margin. The green rectangle is the desired pose of the vessel.

$$\boldsymbol{A_s}\boldsymbol{x_i}^{NED} \leq \boldsymbol{b_s} \quad \forall \boldsymbol{x_i}^{NED} \in \mathbb{S}_b^{NED} \tag{3.9}$$

$$\boldsymbol{A_s}\left(\boldsymbol{R_2}\left(\psi\right)\boldsymbol{x_i}^b + \begin{bmatrix} x \\ y \end{bmatrix}\right) \leq \boldsymbol{b_s} \quad \forall \boldsymbol{x_i}^b \in \mathbb{S}_b^b \tag{3.10}$$

3.7 Trajectory-Tracking DP Controller

To follow the trajectory we get from Section 3.6, a trajectory-tracking DP controller is used. The controller is based on Trajectory Planning and Control for Automatic Docking of ASVs with Full-Scale Experiments [44], but with some modifications. The controller gives us an input vector $\boldsymbol{\tau}$, which is the combination of a feed-forward term and a feedback term, as seen in Equation (3.11).

$$\boldsymbol{\tau}\left(t\right) = \boldsymbol{\tau}_{ff}\left(t\right) + \boldsymbol{\tau}_{fb}\left(t\right) \tag{3.11}$$

The feed-forward term is taken directly from the optimal trajectory found in Section 3.6. We use the decomposed input vector $\boldsymbol{f}_t$, and get the feed-forward term by

Equation (3.12). The timestep used in the trajectory planner is $2s$, so every $\boldsymbol{f}_t$ is used for $2s$ before being updated.

$$\boldsymbol{\tau}_{ff}(t) = \boldsymbol{B}\boldsymbol{f}_t \tag{3.12}$$

In the feedback term we use a PID, where the difference between the actual state, $\boldsymbol{x}$, and the optimal state, $\boldsymbol{x}_p$, is used. However, since the state is updated at a rate of 10Hz in the simulation, while the optimal state is found with a timestep of $2s$, there are significant "jumps" between each optimal state. This leads to the actual state chasing the optimal state. To smooth the optimal trajectory we use Euler integration as shown in Equation (3.13).

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + h \cdot \boldsymbol{g}(\boldsymbol{x}_{t-1}, \boldsymbol{f}) \tag{3.13}$$

Every $2s$ we use the optimal state as a $\boldsymbol{x}_0$, and Euler integrate with $h = 0.1$, such that we get a unique $\boldsymbol{x}_p(t)$ for each timestep. We can then define $\tilde{\boldsymbol{\eta}}(t) = \boldsymbol{\eta}(t) - \boldsymbol{\eta}_p(t)$. The feedback term is given by Equation (3.14), and the gains are given in Equation (3.15). Every time the trajectory planner replans (every $10s$), the integral term is reset by setting $t_{pt} = t$.

$$\boldsymbol{\tau}_{fb}(t) = -\boldsymbol{R}(\psi(t))^\top \left(\boldsymbol{K}_p \tilde{\boldsymbol{\eta}}(t) + \int_{t_{pt}}^{t} \boldsymbol{K}_i \tilde{\boldsymbol{\eta}}(\tau)\, d\tau + \boldsymbol{K}_d \dot{\tilde{\boldsymbol{\eta}}}(t) \right) \tag{3.14}$$

$$\boldsymbol{K}_p = diag\{1000, 1000, 10000\} \tag{3.15a}$$

$$\boldsymbol{K}_i = diag\{10, 10, 20\} \tag{3.15b}$$

$$\boldsymbol{K}_d = diag\{1000, 1000, 15000\} \tag{3.15c}$$

3.8 Thrust Allocation

The two azimuth thrusters have two degrees of freedom each, angle and thrust. In contrast, the input force vector $\boldsymbol{\tau}$ has only three degrees of freedom, $\begin{bmatrix} X & Y & N \end{bmatrix}^\top$. This means that the vessel is overactuated, and we will use optimization to solve it. Control allocation for double-ended ferries with full-scale experimental results

[63] explain how this is already done for milliAmpere 1, but this approach assumes that the input vector is saturated. This means that the input vector is achievable given the thrusters' limitations. While the feed-forward term can be saturated by the trajectory planner, we cannot guarantee that the input vector is saturated when adding the feedback term. Therefore, we will do some modifications.

The variables we use in the optimization are all four decomposed forces, and a scale variable, s, that is used to scale the input vector to an achievable size. Our main objective is to make sure the scale is close to 1, meaning no scaling. The resulting constraints are presented in Equation (3.16). We also have to make sure the decomposed forces do not exceed T_{max}, so Equation (3.8) is also implemented for both thrusters.

$$F_{1,x} + F_{2,x} = s \cdot X \tag{3.16a}$$

$$F_{1,y} + F_{2,y} = s \cdot Y \tag{3.16b}$$

$$F_{1,y} - F_{2,y} = s \cdot \frac{N}{L_x} \tag{3.16c}$$

The bounds on the decomposed forces are almost the same as in Section 3.6, but F_y for both thrusters are now allowed to be $\pm T_{max}$. This is done because the thrust allocation is trying to follow an input, $\boldsymbol{\tau}$, and having smoother behavior is up to the trajectory planner and trajectory tracker. The scale is bounded by zero and one.

The cost function is presented in Equation (3.17), where thrust, T, and the angle, α, is found from Equation (3.18). ΔT and $\Delta\alpha$ are the deviation from last timstep, and $\delta\alpha$ are the deviation from the mean angle. In this case this is π for α_1 and 0 for α_2. All weights are given in Appendix C.

$$C_s = w_s \left(s - 1\right)^2 \tag{3.17a}$$

$$C_T = w_T T_1^2 + w_T T_2^2 + w_{\Delta T} \Delta T_1^2 + w_{\Delta T} \Delta T_2^2 \tag{3.17b}$$

$$C_\alpha = w_{\Delta\alpha} ssa\left(\Delta\alpha_1\right)^2 + w_{\Delta\alpha} ssa\left(\Delta\alpha_2\right)^2 + w_{\delta\alpha} ssa\left(\delta\alpha_1\right)^2 + w_{\delta\alpha} ssa\left(\delta\alpha_2\right)^2 \tag{3.17c}$$

$$C = C_s + C_T + C_\alpha \tag{3.17d}$$

$$T = \sqrt{F_x^2 + F_y^2} \tag{3.18a}$$

$$\alpha = \arctan \frac{F_y}{F_x} \tag{3.18b}$$

Chapter 4

Results and Discussions

In this chapter we will look at some different scenarios for the ferries. The scenarios are presented in Table 4.1. First, in Section 4.1, we look at the movement of passengers, average waiting time and vessel distribution throughout the day for all scenarios. Next, in Section 4.2, we look at how the system works in the evening rush, for the first two scenarios. This is done in order to look closer at how the system works. An important detail is the definition of waiting time, which includes the travel time. This means that the waiting time for a passenger is defined as the time from arrival at a port to the passenger reaches its destination port. Lastly, we see how the control system is performing on a single trip between Nyhavna and Trondheim S in Section 4.3, and some limitations we have encountered using AI-planning in Section 4.4.

4.1 Passenger Handling in Different Scenarios Throughout a Day

In this section we look at the different scenarios presented in Table 4.1. For all scenarios, except the last, we assume a ferry can fit 10 passengers. The actual maximum number of passengers the milliAmpere 1 can carry is 6 [45], while the ferry that is about to be launched in Stockholm, introduced in the introduction, can carry 25 passengers. Why a capacity of 10 passengers was chosen is explained shortly in Section 4.1.1. For the last scenario we look at a capacity of 25 passengers. For all scenarios, except one, the planner is given the same expected number of passengers, P, as the passenger generator, described in Section 3.1.2.

Table 4.1: The different scenarios we look at. AtB expects around 700 passengers a day.

Scenario	**Expected passengers**	**Number of ferries**	**Comment**
Minimum number of ferries	700	2	To follow the timetable 2 ferries are needed.
Adding another ferry	700	3	An extra ferry give more freedom.
The planner is given a higher number of passengers	700	3	The planner expect $1.7 \cdot 700$ passengers instead of the actual 700.
Adding a simple recharging scheme	700	3	Ferries have to recharge. Only one recharging at a time.
More capacity on the ferries	1400	2	Change capacity on the ferries to 25 passengers.

To look at some variance we run the simulation 5 times for every scenario, and show the exact number of passengers and average waiting time. The controllable variance for passengers is the standard deviation of the normal distribution introduced in Section 3.1.2. For the first run of each scenario we set the standard deviation to $\sigma = 0$, to look at a run with little variance. The next four runs have a standard deviation of $\sigma = 7$. For the hour with the most passengers there is an expected value of about 70 passengers, which results in the probability distribution shown in Figure 4.1. To be able to run multiple simulations of a day in a reasonable time we use records of the trips between the ports, such that the control inputs do not have to be found for each timestep.

4.1.1 Scenario 1: Minimum Number of Ferries

To follow the timetable given in Appendix D, two active ferries are needed for most of the day. From AtB we have been given 700 passengers as an expected number of passengers per day. If we had used a capacity of 25 passengers this would be quite easy to handle. Some initial testing has shown that a capacity of 10 passengers is just enough to handle this amount of passengers. We will therefore use this capacity as a basis to look at the added value of another ferry.

The results from the 5 runs are shown in Table 4.2. Looking at the average waiting time, it is around ten and a half minutes, which includes the travel time. From the distribution between the trips in Table 3.1 we see that it is most popular to travel between Nyhavna and Trondheim S, where the trip takes about eight to nine minutes. This means that the resulting average waiting times are quite good. In

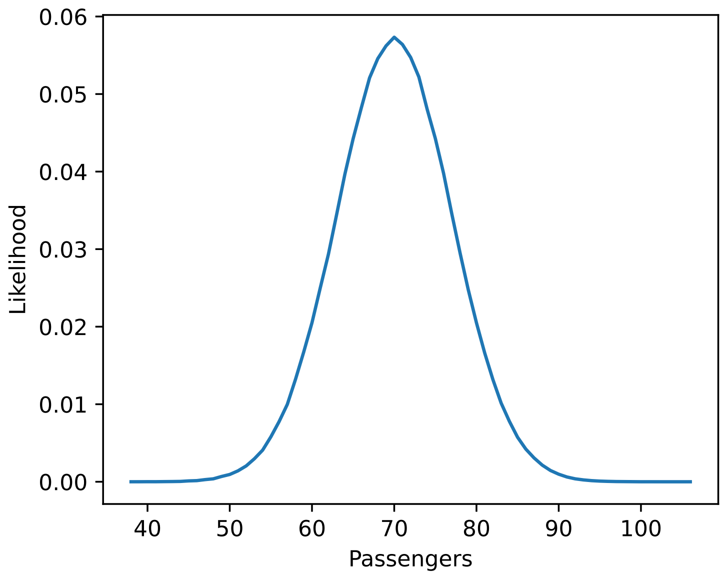


Figure 4.1: The likelihood of different numbers of passengers for an hour with an average expected number of passengers of 70, and a variance of $\sigma^2 = 7^2$.

Table 4.2: The number of passengers, average waiting time and vessel usage for the first scenario in Table 4.1.

Number of passengers	704	735	688	676	666
Average waiting time	10 : 23	10 : 35	10 : 27	10 : 32	10 : 12
Usage vessel 1	$15h$	$15h$	$16h$	$15h$	$15h$
Usage vessel 2	$20h$	$20h$	$19h$	$20h$	$20h$

the third and fourth run we see that there are fewer passengers, but the average waiting time is not reduced. As shown in Figure 4.5, the ferries are seldom loaded with max capacity, so to decrease waiting time more ferries are needed. The last run has a shorter waiting time than the others, but not by a lot. This is probably mostly caused by randomness where passengers arrive closer to departure. Lastly, we see that the ferries are used for 35 hours in total. This is the minimum required to follow the timetable. As we can see, the load is not shared equally. This will be discussed more in scenario 4.

Next, we look more closely at the first run. Looking at the passengers per hour in Figure 4.2, it looks similar to the average distribution shown in Figure 3.4. This run is without any randomness in the expected number of passengers per hour, but since the number of passengers per min is decided with a Poisson distribution there

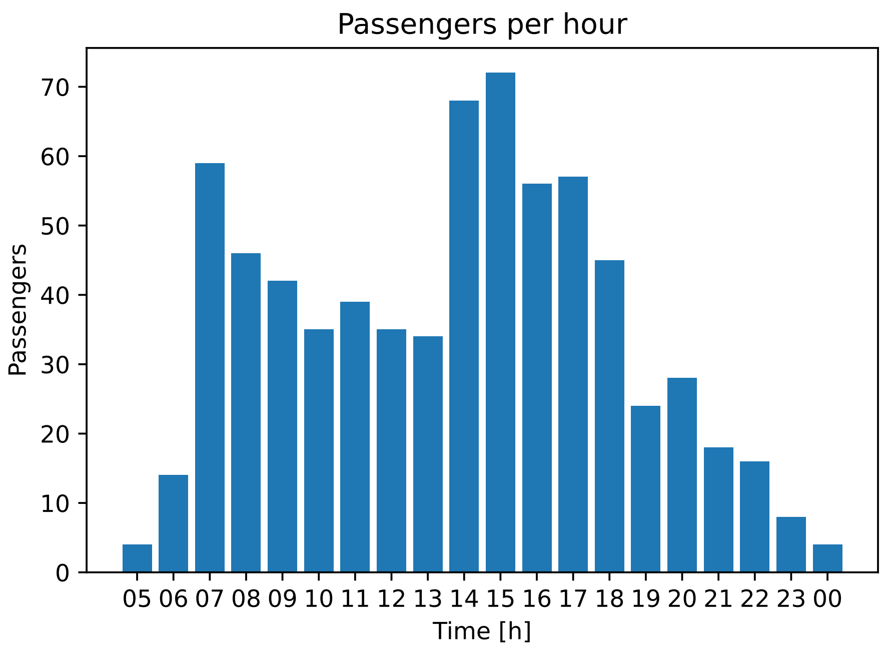


Figure 4.2: Number of passengers per hour for the first run of scenario 1 in Table 4.1.

is still some randomness.

From the average waiting time per hour, shown in Figure 4.3, we see that the waiting time is distributed mostly equally throughout most of the day. However, at five and nine the waiting time is longer. At five a passenger arrived at the port long before the ferry was supposed to leave. A situation like this is unlikely to happen in real life, but could be fixed with an on demand function. Why it is so large at nine is a little difficult to say, but two things might have happened. Firstly, the waiting time per hour is the number of minutes waited by passengers divided by the number of passengers that arrive at the given hour. Therefore, some of the passengers that arrived before nine might have been traveling after nine and thereby increased the average waiting time. We also see that the waiting time generally increases towards the end of the day, as the ferries start to leave every 20 minutes instead of every 10 minutes. Still, the average waiting time is usually below 15 minutes, which is reasonable outside rush hours.

In Figure 4.4 we see how many passengers are waiting at each port to take each trip for every minute. Here we can see that a single passenger showed up at Nyhavna to go to Trondheim S very early, and had to wait for a long time. Luckily the ferry was already there so the passenger could wait inside the ferry. Mostly we see spikes, which means that the passengers were picked up quite fast. The wider columns are mainly single passengers that came too early. At the busiest hours, we see that the number of waiting passengers does not reach zero every time a ferry comes to pick

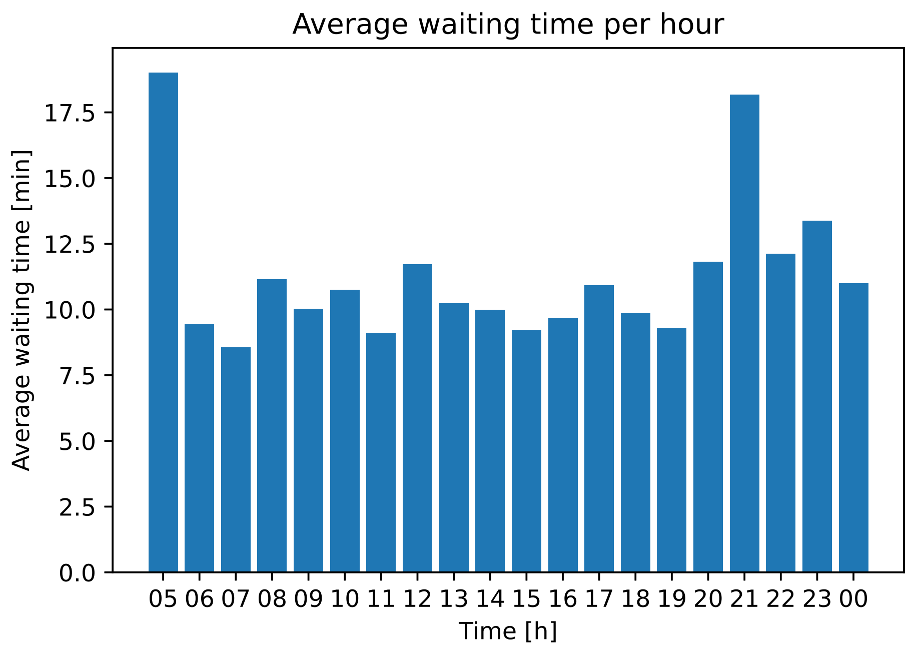


Figure 4.3: Average waiting time per hour for the first run of scenario 1 in Table 4.1. The waiting time includes the time spent on the ferry as well.

up passengers. This is either because the ferry is full, or that new passengers come right after the ferry leaves. While the latter is the passengers' fault, the former means that we need bigger ferries or more ferries for the busiest hours.

Figure 4.5 shows how many passengers each vessel has onboard. Here we also see that the second vessel is active for more hours than the first. We also see that the ferries are seldom filled to capacity, which means that it is unlikely that any passengers are left behind. To test this we ran another 5 simulations with $\sigma = 7$, and found that there are on average 2 passengers a day that have to wait on the next ferry. The highest value was 3 passengers a day, and it happened during rush hours every time. If the max capacity was increased slightly, leaving passengers behind would then be very unlikely. In the morning the second vessel is traveling without passengers for some time. This might indicate that we do not need two vessels that early.

The other runs for this scenario were mostly similar to the first in terms of performance, even though the passenger distribution was different. For the second run, the number of passengers traveling between 15:00 and 16:00 was above 90, while the average waiting time for that hour was still below 11 minutes. Run 3 also had more passengers between 15:00 and 17:00 than run 1, while the average waiting time was also under 11 minutes for those hours. These runs were filled up to max capacity 5 and 7 times respectively. This could indicate that a capacity of 10 passengers and

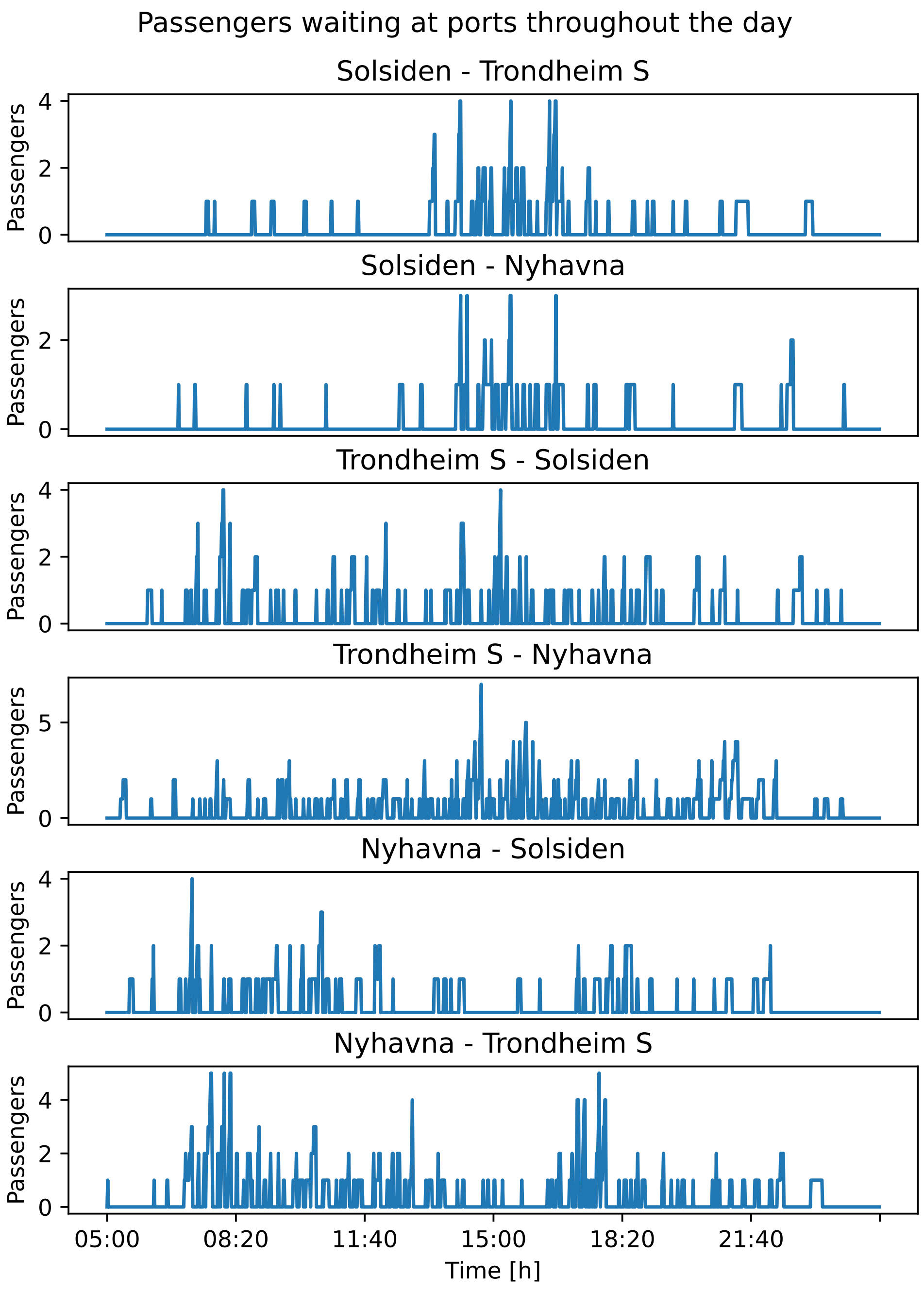


Figure 4.4: How many people that are waiting at each port to take each trip every minute for the first run of scenario 1 in Table 4.1.

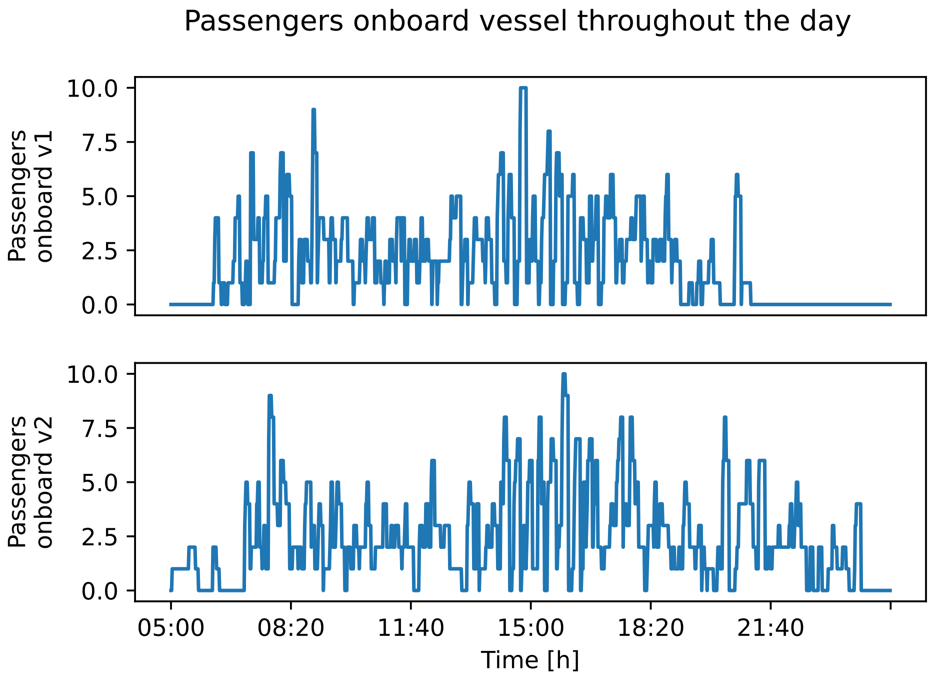


Figure 4.5: How many passengers each ferry has onboard for every minute for the first run of scenario 1 in Table 4.1.

2 ferries have too little capacity for an unusually busy rush hour.

4.1.2 Scenario 2: Adding Another Ferry

For the second scenario we added a third ferry. This means that in addition to following the timetable, the planner can use an extra ferry for hours with a larger expected number of passengers. From the results of the 5 runs, shown in Table 4.3, we see that the total usage of the vessels has gone up to 36 and 37 hours. This means that the system has only found the busiest hour to be worth using all three ferries. Since the planner plans for the same number of passengers for all the 5 runs, the one run with a usage of 37 hours is actually an issue with the planning problem. The planner found that it is most effective for a vessel to let off passengers by taking a route. This route is also taken by another vessel, which leads to the extra hour of vessel usage. The reason for this happening is that taking a route is an effect, and the action to start a route is quite short. Therefore, the planner thinks that passengers are let off instantly by taking a route. This problem has to be looked at in future work. We also see that the hours are not distributed equally between the ferries, which we will look at in scenario 4.

The effect of having an extra ferry is a slightly lower average waiting time. Simultaneously, the usage has not gone up dramatically. This is expected, since two

Table 4.3: The number of passengers, average waiting time and vessel usage for the second scenario in Table 4.1.

Number of passengers	701	697	673	727	674
Average waiting time	10 : 10	10 : 25	10 : 12	10 : 30	10 : 15
Usage vessel 1	$3h$	$3h$	$3h$	$3h$	$4h$
Usage vessel 2	$19h$	$20h$	$20h$	$15h$	$19h$
Usage vessel 3	$15h$	$13h$	$13h$	$18h$	$13h$

ferries were able to handle the same number of passengers. Decreasing the average waiting time by a lot is also difficult, as the most popular trip takes around eight to nine minutes. However, using three ferries at the busiest hour can make sure that no passengers are left behind. Just as for the last scenario, 5 simulations were run with $\sigma = 7$ to see how many that had to wait for the next ferry. The average was now 1.5 passengers, and just as in the last scenario this happened during rush hours with 3 passengers left behind as the largest value.

Looking at the passenger distribution through the day in Figure 4.6 we see that it is a little different from the one for the last scenario, shown in Figure 4.2. This is because the passenger distribution per minute is random even though the expected number per hour is not. However, it is still similar to the last scenario, so the distribution will not be included for the later scenarios.

In Figure 4.7 we see the average waiting time for the first run of scenario 2. Here we see that it is low throughout most of the day, with the expected busiest hour, 15:00 to 16:00, being among the lowest. From Figure 4.6 we know that the previous hour was actually busier, but it does not seem to increase the average waiting time by much. Just as for the last scenario, the waiting time increases when the ferry starts to leave every 20 minutes, instead of every 10 minutes. The effect where waiting time is being assigned to the next hour, with a lower amount of passengers might also have an effect here.

The plot of people waiting at each port is very similar to the last scenario, and it has therefore not been included. The only difference between the scenarios is that there is an extra ferry at one or two hours. Therefore, it is expected that we do not see much difference.

When looking at the passengers onboard the ferries in Figure 4.8, we see that the

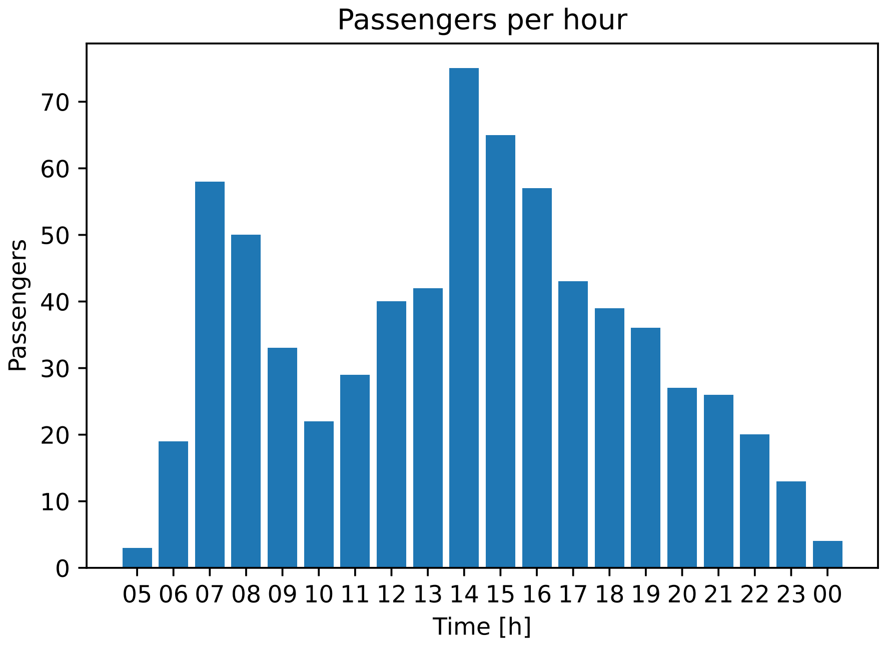


Figure 4.6: Number of passengers per hour for the first run of scenario 2 in Table 4.1.

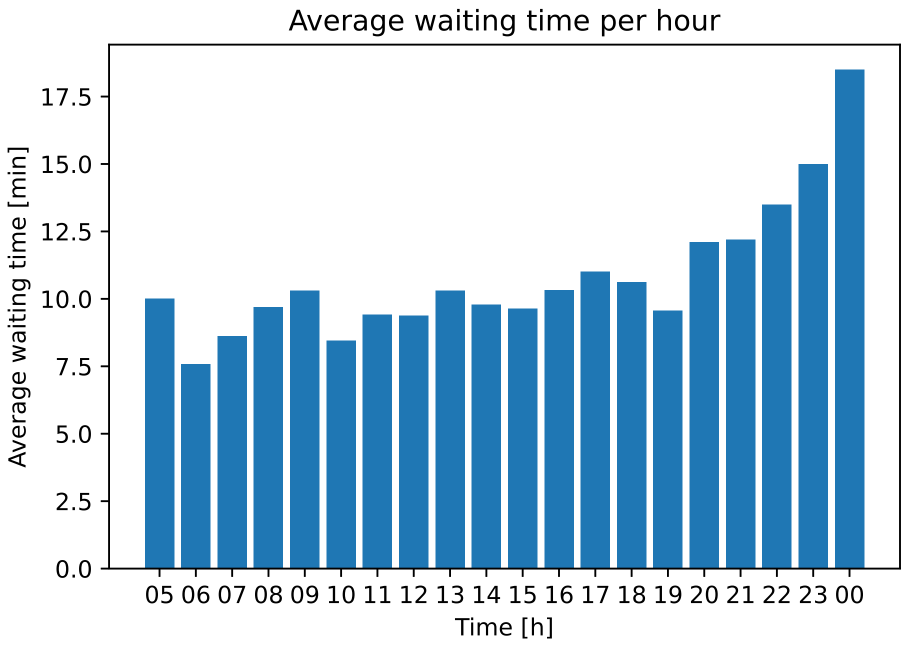


Figure 4.7: Average waiting time per hour for the first run of scenario 2 in Table 4.1. The waiting time includes the time spent on the ferry as well.

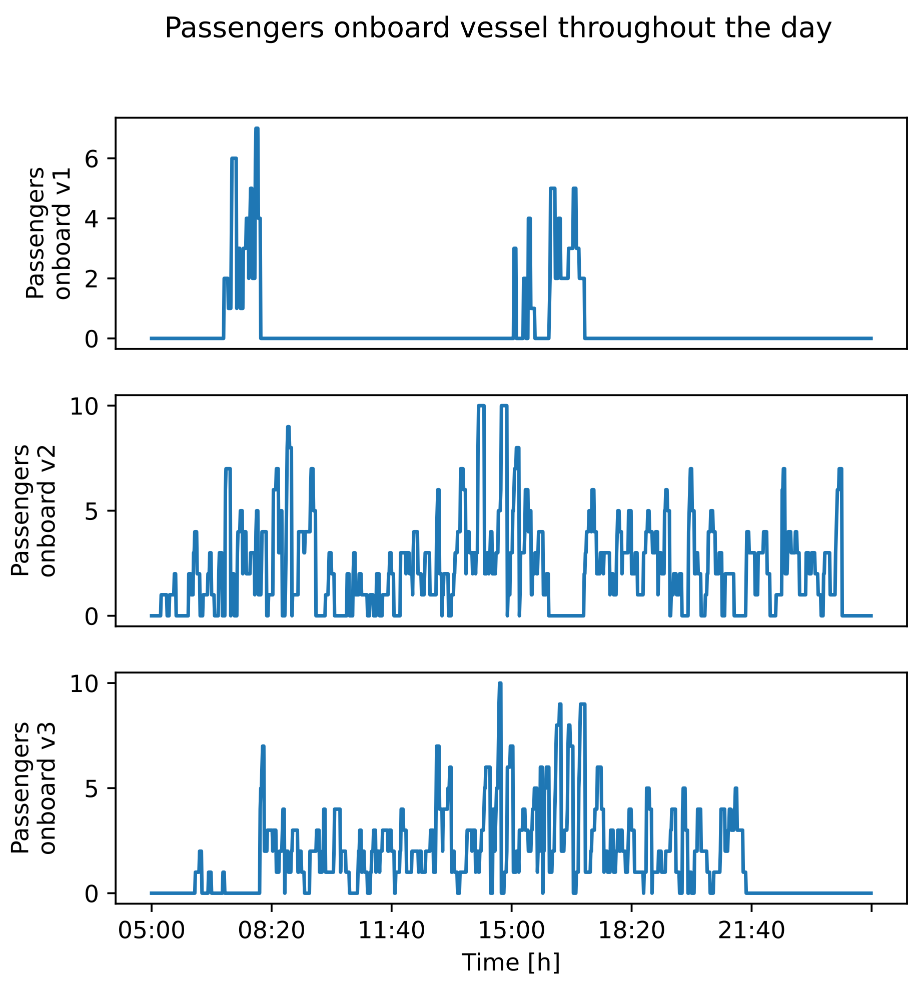


Figure 4.8: How many passengers each ferry has onboard for every minute for the first run of scenario 2 in Table 4.1.

first ferry does not move a lot of passengers between 15:00 and 16:00 when all three ferries are active. This means that passengers do not have to wait for the next ferry because of crowded ferries. However, since there were other hours that were just as busy, the extra ferry could be used more often to decrease the waiting time further.

4.1.3 Scenario 3: Planning for More Passengers Than the Actual Number

Since the actual number of passengers can vary a lot, it is sensible to plan for a higher number than the average. If we use the passenger distribution throughout the day provided by AtB, shown in Figure 3.4, we see that the busiest hour has 10.9% of the passengers, which is almost 1.7 times as much as other rush hours. This means that if we plan for 1.7 times as many passengers, the system should use all ferries at all rush hours.

In Table 4.4 we can see that the total usage time has gone up to 41 hours. This means that for 6 hours all three ferries are used at the same time. This results in an even lower average waiting time, but not by a lot. However, when testing how many passengers that have to wait for the next ferry, the average ended up being 0.8 passengers with 2 passengers as the highest. Another advantage of using 3 ferries is that the waiting time can be less than 10 minutes to the next ferry during rush hours. As the amount of passengers above capacity has not exceeded 3 for any of the scenarios, slightly larger ferries would fix this problem, but could be more expensive.

Table 4.4: The number of passengers, average waiting time and vessel usage for the third scenario in Table 4.1.

Number of passengers	703	721	690	685	724
Average waiting time	9 : 54	9 : 57	10 : 11	10 : 07	10 : 15
Usage vessel 1	$8h$	$6h$	$6h$	$8h$	$6h$
Usage vessel 2	$18h$	$20h$	$20h$	$18h$	$20h$
Usage vessel 3	$15h$	$15h$	$15h$	$15h$	$15h$

The average waiting time per hour, shown in Figure 4.9, is overall very low. Towards the end, it becomes a little higher, just as for the last scenarios. This is outside the rush hours where the system works the same as for the last scenarios. So it is expected that the results are similar. The passengers waiting at the ports plot is similar to the first scenario, but with slightly thinner spikes, so it is omitted here as well.

In Figure 4.10 we see that the ferries are filled to max capacity about as often as in the first scenario. This is a little surprising since we have more ferries that go more often. However, the passengers will always take the ferry that leaves first, so if there are 10 passengers ready they will all take the first ferry arriving.

4.1.4 Scenario 4: Adding a Simple Recharging Scheme

In the first three scenarios the workload has been distributed quite unevenly. This both means that the tear on the ferries is uneven, but also means that there is

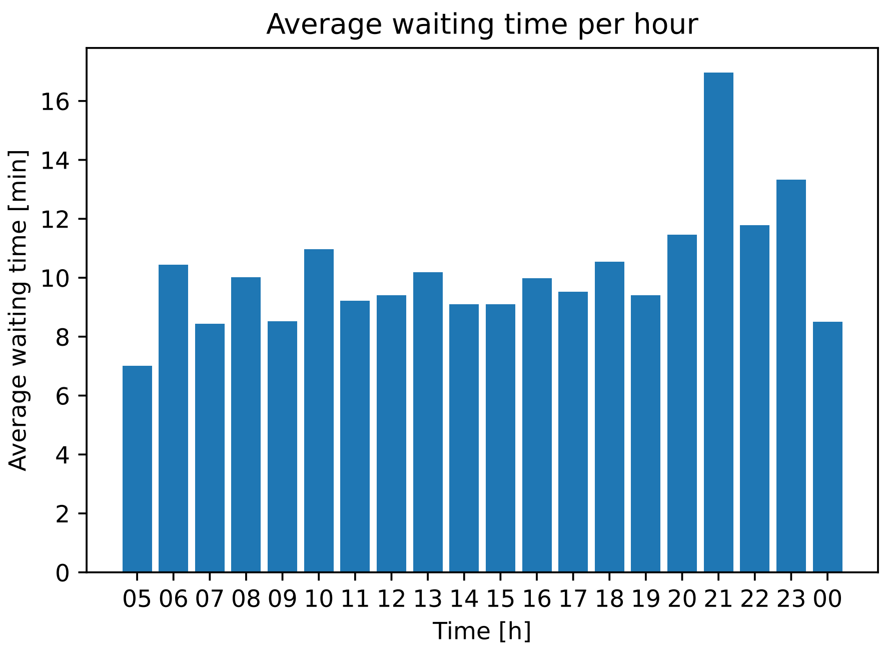


Figure 4.9: Average waiting time per hour for the first run of scenario 3 in Table 4.1. The waiting time includes the time spent on the ferry as well.

no room for recharging or maintenance for the ferries with the biggest workload. Therefore, we have added a very simple recharging scheme. If a ferry has been active for three hours straight it has to take a break for an hour. However, since two active ferries are needed to follow the timetable, only one ferry can have a break at a time. This scheme is very simplistic, but shows how the system works with unavailable ferries.

The results of the 5 runs for this scenario are shown in Table 4.5. Firstly, we can see that the 36 hours workload is distributed more evenly. Next, we see the largest variance in the number of passengers until now. This is a coincidence, as the passengers are added in the same way as in the first three scenarios. Except for the last run, the average waiting time is about the same as for the second run. The last run is not that much worse either, so trying to see what increased the waiting time for this particular run is difficult. Something that could have happened is that when a ferry is set to inactive, the ferry taking over its route might be at the other end of the channel. It then has to travel all the way to the start of the route, which will increase waiting time. This can be avoided if the recharging scheme was a part of the planning problem, and can be looked at in further work.

While the other plots are mostly similar to the first scenarios, the passengers onboard plot is a little different, as we can see in Figure 4.11. This shows how the workload is distributed much more evenly among the ferries. From 15:00 to 16:00 all ferries

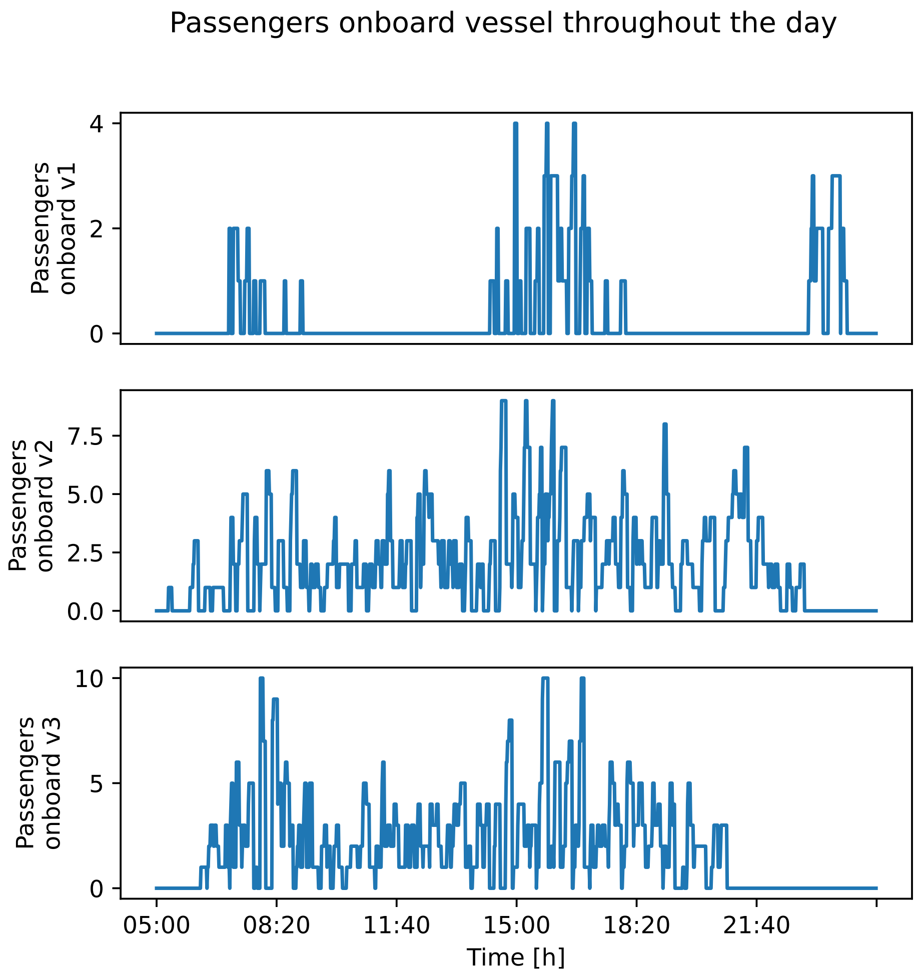


Figure 4.10: How many passengers each ferry has onboard for every minute for the first run of scenario 3 in Table 4.1.

Table 4.5: The number of passengers, average waiting time and vessel usage for the fourth scenario in Table 4.1.

Number of passengers	690	675	761	587	724
Average waiting time	10 : 20	10 : 32	10 : 14	10 : 11	10 : 52
Usage vessel 1	$12h$	$13h$	$12h$	$12h$	$12h$
Usage vessel 2	$14h$	$11h$	$14h$	$14h$	$12h$
Usage vessel 3	$10h$	$12h$	$10h$	$10h$	$12h$

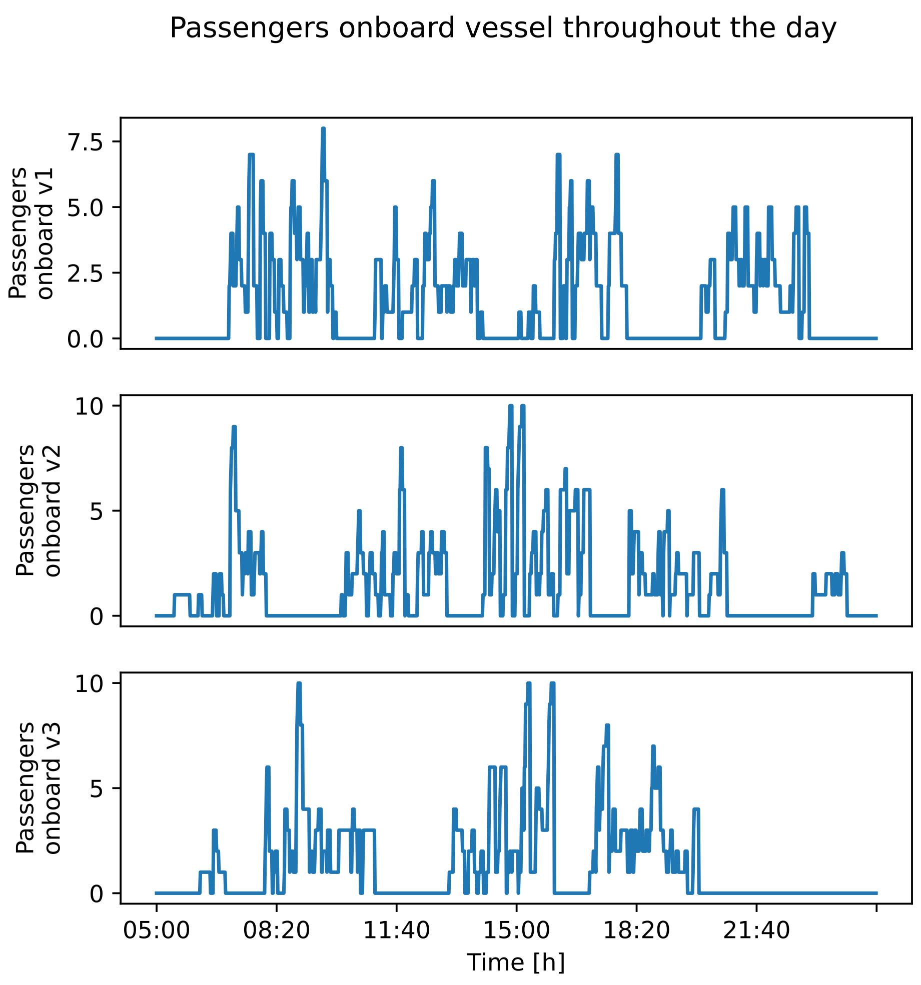


Figure 4.11: How many passengers each ferry has onboard for every minute for the first run of scenario 4 in Table 4.1.

are active, but we see that the first ferry carries much fewer passengers. The other two ferries are probably the ones that follow the predefined routes, meaning that the passengers know the departure times. In contrast, the ferry that is set to take extra capacity just goes as fast as it can. This effect is present for all scenarios with 3 ferries, but it seems like the passengers were better at following the timetable in this run.

4.1.5 Scenario 5: More Capacity on the Ferries

It would be sensible for AtB to use already existing ferries like the one about to be launched in Stockholm, with a capacity of 25 passengers. Since the increased capacity would simply mean that there would always be enough space in the ferries, we also look at how the system would handle twice as many passengers.

The results are shown in Table 4.6, where we can see that the average waiting time is about the same as for the first scenario. However, there is a little less variance. This might simply be because we have more passengers. Another observation is that the bug we encountered in the second scenario, where a ferry takes a whole route to let off passengers, is more prominent here. This is no surprise as the increase in passengers also increases the likelihood of having passengers onboard. Then the situation where this mistake is done becomes more likely.

Table 4.6: The number of passengers, average waiting time and vessel usage for the fifth scenario in Table 4.1.

Number of passengers	1441	1451	1455	1353	1341
Average waiting time	10 : 25	10 : 23	10 : 27	10 : 25	10 : 21
Usage vessel 1	$15h$	$19h$	$15h$	$19h$	$15h$
Usage vessel 2	$20h$	$17h$	$20h$	$17h$	$20h$

Most of the plots are similar to the last scenarios, but with more passengers. In Figure 4.12 we see the passengers onboard the ferries throughout the day. As opposed to before, the ferries are never filled to capacity. Through the five runs, a ferry was only full one time, in the third run, but it did not result in any passengers being left behind. This means that with a capacity of 25 passengers, the system is able to handle twice as many passengers as expected by AtB.

4.2 System Performance During the Expected Busiest Hours

In this section we look at how the ferries handle the passengers between 15:00 and 17:00 for the first two scenarios in Table 4.1. For both scenarios we start by looking at the plans from the planner. Next, the ferries' movements are presented, before the movement of passengers is shown. For both scenarios the simulation is run one time with $\sigma = 0$.

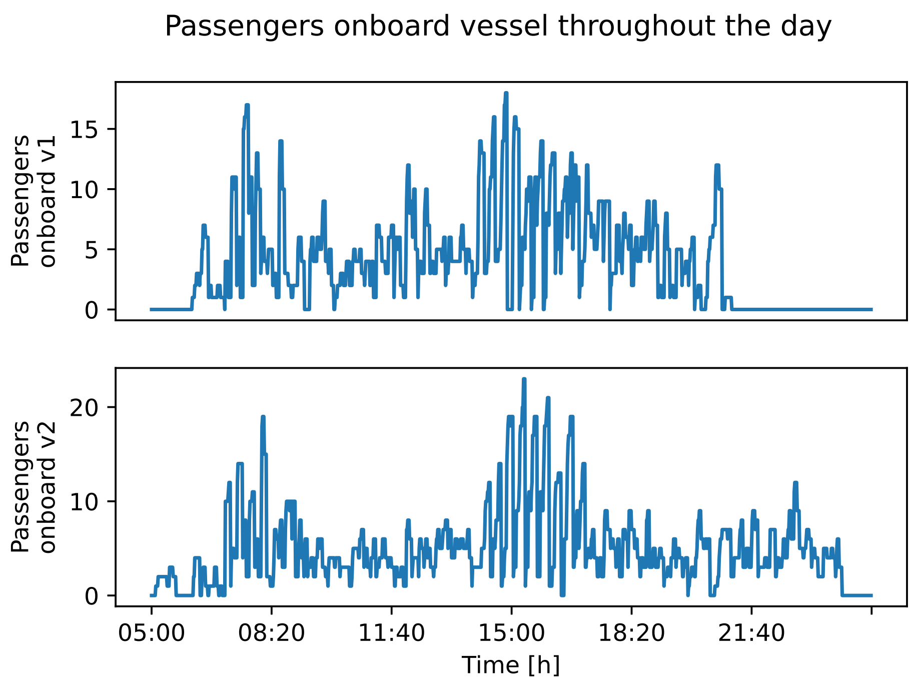


Figure 4.12: How many passengers each ferry has onboard for every minute for the first run of scenario 5 in Table 4.1.

4.2.1 Scenario 1: Minimum Number of Ferries

The plan made at 15:00 is shown in Listing 4.1, where the initial state was that ferry 1 was on its way to Trondheim S (*portd*) with passengers onboard, and ferry 2 was on its way to Nyhavna (*portb*), also with passengers onboard. Both vessels are coming from Solsiden (*portc*). There are two routes that have to be taken, namely route 1 and 2. Since both ferries are between two ports, they begin with completing the trip they have started. Next, they let off current passengers, before starting the routes. Since the ferries are going towards the end ports, no passengers are going further. Therefore, the last parameters in *take_planned_route*, which are the next ports, have no effects and are therefore set by the planner to two identical ports.

```
0.000: (move_to_port v1 portd portc)  [147.500]
0.000: (move_to_port v2 portb portc)  [127.000]
127.001: (let_off_passengers v2 pa2 portb)  [15.000]
147.501: (let_off_passengers v1 pa1 portd)  [15.000]
162.501: (take_planned_route v1 route1 portd portb portb)  [1.000]
163.502: (take_planned_route v2 route2 portb portc portc)  [1.000]
```

Listing 4.1: Resulting plan, made at 15:00, for scenario 1 in Table 4.1. v1 and v2 are the two ferries and pa1 and pa2 are groups of passengers onboard. portb is Nyhavna, portc is Solsiden and portd is Trondheim S.

This is a simple plan, that was found in 0.01s, and it is basically saying that the ferry should keep following the timetable. With 2 ferries and 2 routes that have to be taken, there is not much room for variance. So, unsurprisingly, the plan is the exact same when planning at 16:00. With plans as simple as these, the planner is not adding much value. However, with more ferries than routes this can change.

In Figure 4.13 the movements of the two ferries are shown. The possible locations are the three ports and the trip between them. In a real system this would not be as symmetrical, but since we use the recorded trips and do not add random delays this is bound to happen in the simulation. To take the possibilities of delays into account it does not take a full 10 minutes from one end port to another, where 10 minutes is the shortest time between departures from the same port in the timetable. This leads to a long loading time at the end ports, where it is around 140s at Trondheim S and around 100s at Nyhavna. From the perspective of the passengers this is not a problem, as the ferries leave at the given time. Being able to go onboard in good time before the ferry leaves is probably also a positive thing. There is also added some room for delay for Solsiden, where the ferry is loading for around 30s when going westwards and 80s when going eastwards. While 30s might be a sensible loading time, 80s is too long. However, moving the departure time with a minute would make the loading time go below 20s, which might be too short.

Because Solsiden is almost exactly in the middle between the other ports, the two ferries are loading there with some overlap. This could be fixed with a change in the timetable. However, with more ferries and possible delays, handling multiple cooperative ferries using the same dock is a task that has to be looked at in future work anyway. At the same time, assuming that there is room for at least two ferries at a time is not that far-fetched. So the time used will probably not change drastically.

Figure 4.14 shows the passengers waiting at each port, just as in Section 4.1. In addition, the departure times of the ferries are shown as crosses. As the number of passengers waiting is updated every minute, it looks like passengers are being picked up after the ferry has left sometimes. Another observation is that some of the columns are quite wide, often starting right after a ferry has left. This is because we made it as likely to come just a little too late as 5 minutes before departure, as can be seen in Figure 3.5. If this is an unrealistic assumption and passengers usually come on time, the average waiting time will be even better than the results in Section 4.1. Lastly, we also see that the ferries pick up passengers almost every time it loads.

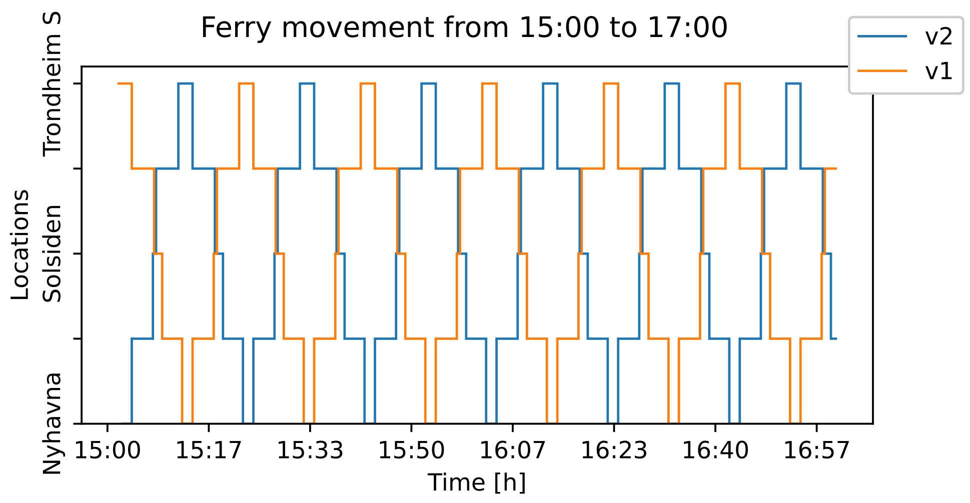


Figure 4.13: Movement of the two ferries in scenario 1 in Table 4.1 between 15:00 and 17:00. Ferry 1 is $v1$ and ferry 2 is $v2$. The locations between the ports indicate that the ferries are traveling between the ports.

In Figure 4.15 the amount of passengers onboard the ferries is shown, together with the arrival time at the different ports. At the beginning of the plot for ferry 2, there is a double arrival at Nyhavna. The same also happens right after 16:00. This is because the $let\_off\_passengers$ action is translated to a load task and the $take\_planned\_route$ starts with a loading task. In Figure 4.14 this is not visible because the first load task has no direction, which means that the ferry does not indicate to waiting passengers if it is still active. For the passengers this might be a little confusing, but with the timetable and the fact that the second load task indicates that the ferry is active, it should not be a huge problem. Since ferry 1 also has the same action sequence, the same thing happens. However, in the plot, the time is rounded down to a whole minute, resulting in the same timestamp for both load actions. Given the slack in loading time, there is still more than enough time to perform a normal loading sequence. At the same time, it adds a potentially unwanted extra loading time of $15s$, which is not a lot, but it might be unnecessary. The problem comes from the planner, which has actions with different levels of abstraction. While the move and let off passengers actions are translated to just a few tasks, the taking a route actions are translated to many tasks. A plan compression scheme could, however, fix some of these problems by detecting the double loading task.

We can see from Figure 4.15 that the ferries are filled to max capacity three times. Every time has been after visiting Solsiden. If we look at Figure 4.14 as well we see clearly that there are none left waiting at the port after visiting. This indicates that a capacity of 10 was just enough. However, we saw from Section 4.1 that passengers

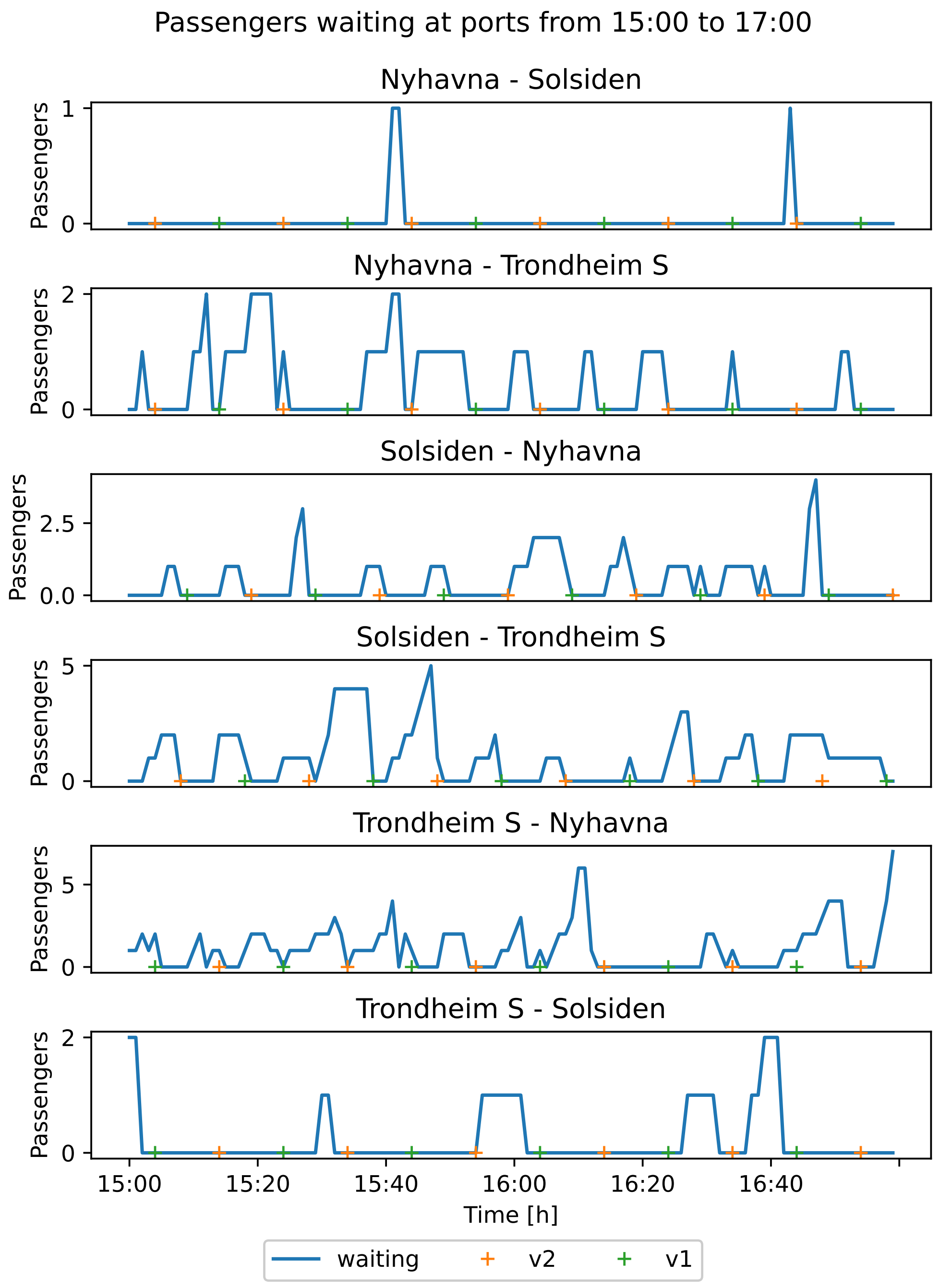


Figure 4.14: Passengers waiting at the different ports divided by their wanted destination between 15:00 and 17:00 for scenario 1 in Table 4.1.

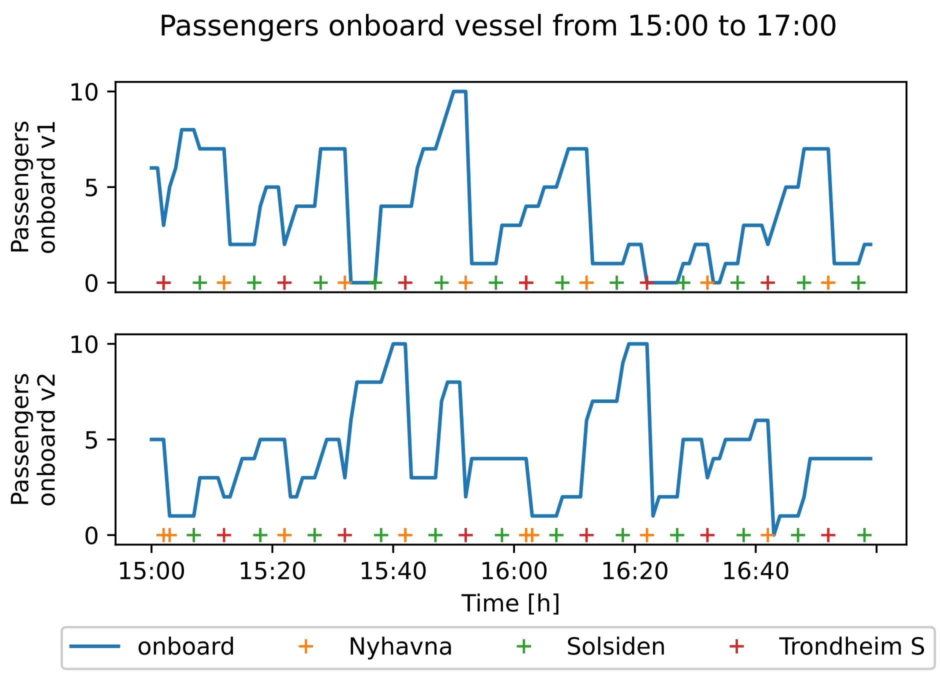


Figure 4.15: Passengers onboard each ferry between 15:00 and 17:00 for scenario 1 in Table 4.1. Ferry 1 is $v1$ and ferry 2 is $v2$.

are left behind at times, using the same capacity. Another observation is that the ferries are almost never empty. There are only two times that ferry 1 is traveling without passengers. The ferries are at around half the capacity for most of the time in this scenario. It could be interesting to see if a different planning scheme would work better in future work. As an example, a pure on demand scheme with cameras counting the amount of waiting passengers could be interesting. As discussed in Section 4.4, this would require a different planning algorithm.

4.2.2 Scenario 2: Adding Another Ferry

The plan made at 15:00 is shown in Listing 4.2, and the plan made at 16:00 is shown in Listing 4.3. At 15:00 the initial state is similar to the last scenario. Ferry 2 and 3 are on their way to Nyhavna and Trondheim S with passengers onboard. The difference is, of course, that there is a third ferry already waiting at Trondheim S. This results in almost the same plan, with the difference being that the third ferry is covering the expected capacity between Solsiden and Trondheim S. Although the planner only used $0.28s$ to solve this, it is a huge increase compared to the last scenario. This shows how the solving time can increase rapidly with a small increase in plan length.

The plan in Listing 4.2 could actually be a second shorter. To make sure that the route starting first is taken first we added a turn variable to the planning problem. This does not make as much sense when the two routes start at the same time, or at different ports. However, the plan is translated to tasks individually by the different ferries. Therefore, this does not result in any actual time delay. In the planning problem, it does have an effect. As we can see from the last two actions, *route*2 is taken after *route*1. However, if ferry 1 took *route*1, *route*2 could be taken right after ferry 2 was done letting passengers off.

```
0.000: (move_to_port v3 portd portc)  [147.500]
0.000: (move_to_port v2 portb portc)  [127.000]
0.000: (route_single_trip v1 tripcd portd)  [1.000]
127.001: (let_off_passengers v2 pa1 portb)  [15.000]
147.501: (let_off_passengers v3 pa2 portd)  [15.000]
162.501: (take_planned_route v3 route1 portd portb portb)  [1.000]
163.502: (take_planned_route v2 route2 portb portc portc)  [1.000]
```

Listing 4.2: Resulting plan, made at 15:00, for scenario 2 in Table 4.1. v1, v2 and v3 are the three ferries and pa1 and pa2 are groups of passengers onboard. portb is Nyhavna, portc is Solsiden and portd is Trondheim S.

It is a little difficult saying exactly what went wrong, but upon further inspection we discovered that removing the metric resulted in the optimal plan. Since travel time dominates the plan length, it will be minimized anyway, so removing the metric would probably be a safe thing to do. Since the cost of the plan was unchanged, it is unclear why the metric had an effect. However, by taking a route ferry 1 would not be able to use a move action afterward, stopping that opportunity to increase the cost.

```
0.000: (move_to_port v3 portd portc)  [147.500]
0.000: (move_to_port v1 portd portc)  [147.500]
0.000: (move_to_port v2 portb portc)  [127.000]
127.001: (let_off_passengers v2 pa1 portb)  [15.000]
147.501: (let_off_passengers v3 pa2 portd)  [15.000]
147.501: (take_planned_route v1 route1 portd portb portb)  [1.000]
148.502: (take_planned_route v2 route2 portb portc portc)  [1.000]
```

Listing 4.3: Resulting plan, made at 16:00, for scenario 2 in Table 4.1. v1, v2 and v3 are the three ferries and pa1 and pa2 are groups of passengers onboard. portb is Nyhavna, portc is Solsiden and portd is Trondheim S.

In the plan made at 16:00, given in Listing 4.3, the initial state is almost the same as for 15:00. The only difference is that ferry 1 is on its way to Trondheim S (portd), instead of being there. Just as before, the extra ferry is only needed between 15:00 and 16:00, so no extra routes are taken in this plan. Since ferry 3 has passengers, and ferry 1 is empty, ferry 1 is taking over the route. This could be a problem as we do not add how far each ferry has come between the ports. However, as we can see in Figure 4.16, ferry 1 is not delayed even though it was actually further behind. As described in Section 4.4, we were not able to encode the departure time in the planning problem, so the planner tries to find a ferry for the routes as fast as possible. Therefore, having a more exact estimation of how far away the ferries are would not necessarily give a better plan, as the ferry could have to wait for departure anyway. Compared to two ferries, this is still a simple plan, computed in $0.54s$. Although this is not a lot either, it is still a major increase compared to using two ferries.

With three ferries, the planner gives a bit more value by assigning ferries to routes. If ferry 1 had any passengers onboard it would also have made sure these reached their destination. At the same time, the planner does not need to come up with a particularly intelligent plan. Therefore, it might be possible to make a planning algorithm without the use of AI-planning, that does the same thing. Nevertheless, the planner does its job reasonably well, even though it is not perfect. In future work, it would be interesting to see if the planner can make the system more intelligent. Maybe with the use of another AI-planner.

The movements of all three ferries are shown in Figure 4.16. While ferry 2 and 3 do the same as for the last scenario between 15:00 and 16:00, ferry 1 is traveling between Trondheim S and Solsiden as frequently as possible. The lowest allowed loading time in the simulation is $15s$, which means that ferry 1 can take more trips than the other two. The minimum loading time of $15s$ is not based on how long it takes to load and unload passengers on the milliAmpere 1, but rather an estimate based on buses outside rush hours. Therefore, it might be that this estimate is incorrect, and should perhaps be simulated using a stochastic loading time. Since the frequency of ferry 1 is different than the others, there are two instances where all three ferries are loading at Solsiden around the same time. As mentioned before, having multiple ferries wanting to use a port at the same time is a problem that has to be looked at in future work. Lastly, we can see that ferry 1 was a little behind ferry 3 when replanning at 16:00. However, ferry 1 had still more than enough time to load at Trondheim S.

In Figure 4.17 we see the number of passengers waiting at the ports for this scenario. For the trip between Solsiden and Trondheim S, we can see that crosses appear quite

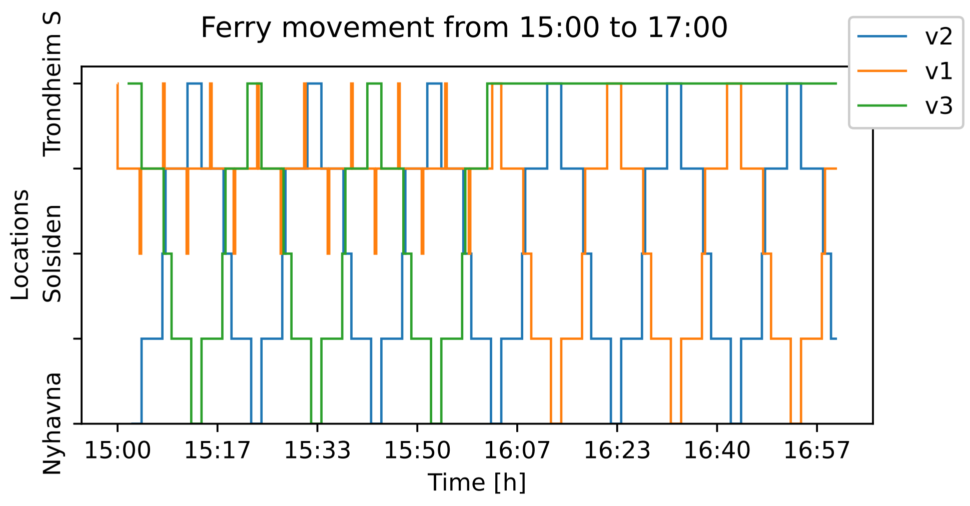


Figure 4.16: Movement of the three ferries in scenario 2 in Table 4.1 between 15:00 and 17:00. Ferry 1 is $v1$, ferry 2 is $v2$ and ferry 3 is $v3$. The locations between the ports indicate that the ferries are traveling between the ports.

often before 16:00, resulting in a shorter waiting time for these trips. However, there is often no one waiting when ferry 1 arrives. The passenger distribution is fitted to the timetable, so it is not a surprise that the extra ferry has a little less to do. It does not help that ferry 1 is sometimes arriving right after one of the other two ferries. To make the passengers aware of the extra ferry, there should be some screens at the ports with estimated departure times of all ferries, although passengers would probably always take the first ferry, even if they were only going to Solsiden.

Around 15:40 we can see that a passenger was left behind for the trip from Solsiden to Nyhavna. This begs the question if the trip between Solsiden and Trondheim S is the only one needing extra capacity. The estimated extra needed capacity is calculated by looking at the number of passengers wanting to travel on each stretch, meaning that if someone wants to take a trip from Trondheim S to Nyhavna it counts towards both stretches. Looking at Figure 4.17, we can see that the stretch between Solsiden and Trondheim S is a little busier. In the generated planning problem there is also just one unit capacity that has to be covered, meaning that the planner could also have chosen to take the full route. Looking at Figure 4.18 we see that a ferry is only full one time, so this kind of abnormality is hard to predict.

In Figure 4.18 we see the number of passengers onboard the three ferries. Here we see even more clearly that ferry 1 is not adding that much value before 16:00, as it mostly travels without any passengers. However, it did have a small effect on average waiting time, as seen Section 4.1.2, although it might have been used better. Having it be more reactive to the number of passengers waiting could be interesting to look at in the future.

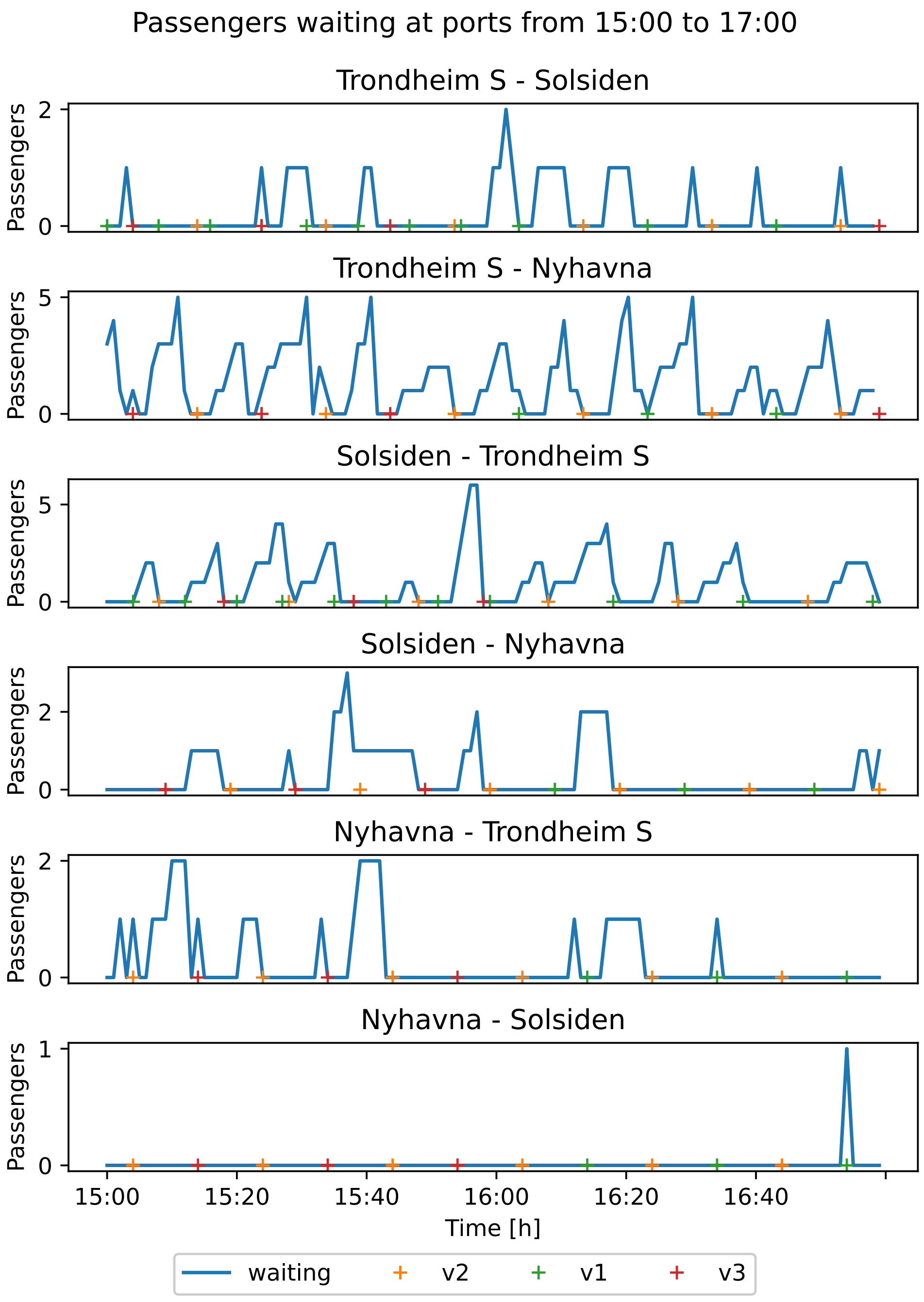


Figure 4.17: Passengers waiting at the different ports divided by their wanted destination between 15:00 and 17:00 for scenario 2 in Table 4.1.

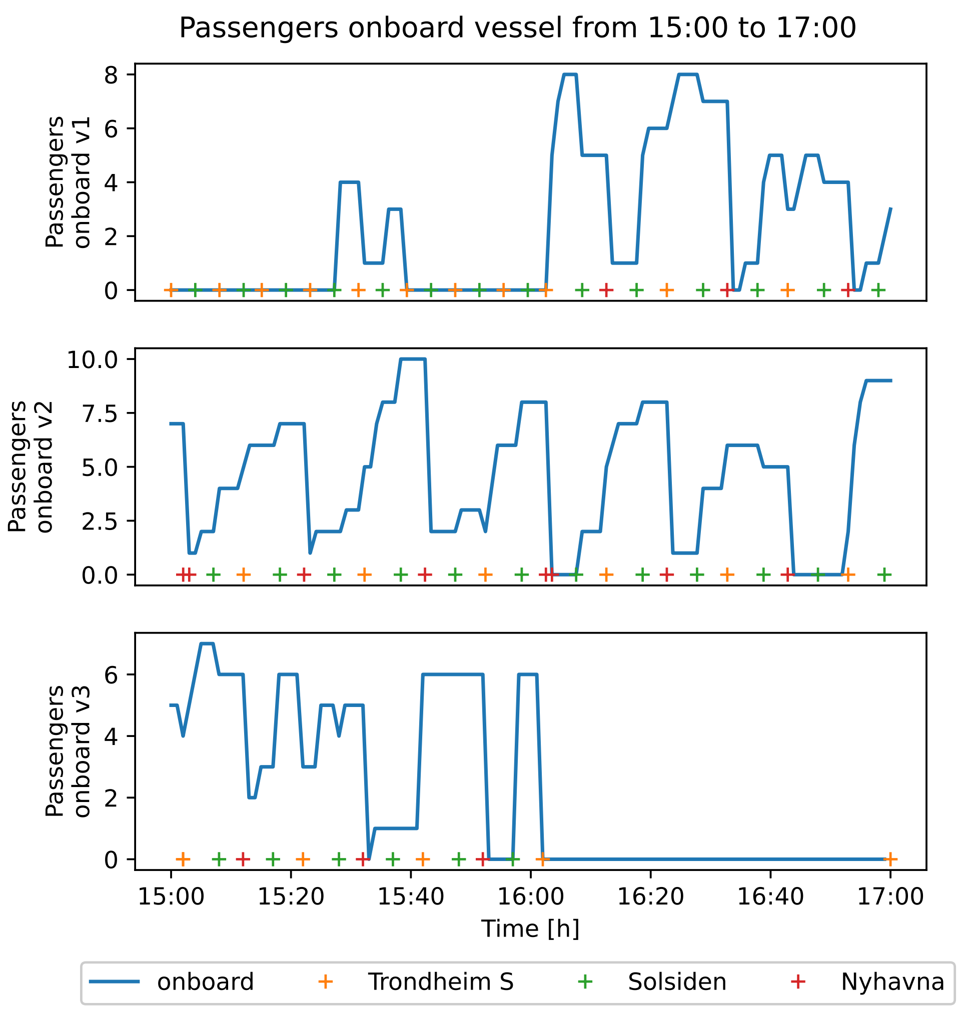


Figure 4.18: Passengers onboard each ferry between 15:00 and 17:00 for scenario 2 in Table 4.1. Ferry 1 is $v1$, ferry 2 is $v2$ and ferry 3 is $v3$.

4.3 Control System Performance During a Single Trip

In this section we look at the performance of the control system. Since the performance is similar for the different trips we only look at the trip from Nyhavna to Trondheim S. An overview of the vessel's path can be seen in Figure 4.19. The vessel's pose is plotted every 10 seconds as a rectangle such that we can see more clearly how the vessel moves. It is worth noting that the map is simply plotted as a picture behind the path. Together with the inaccuracies that come from the earth being round and the plot being flat, the plotted map becomes a little inaccurate. The control system operates with the actual coordinates of the convex sets, so even though it looks like the vessel crashed into the corner at the bottom, the path is

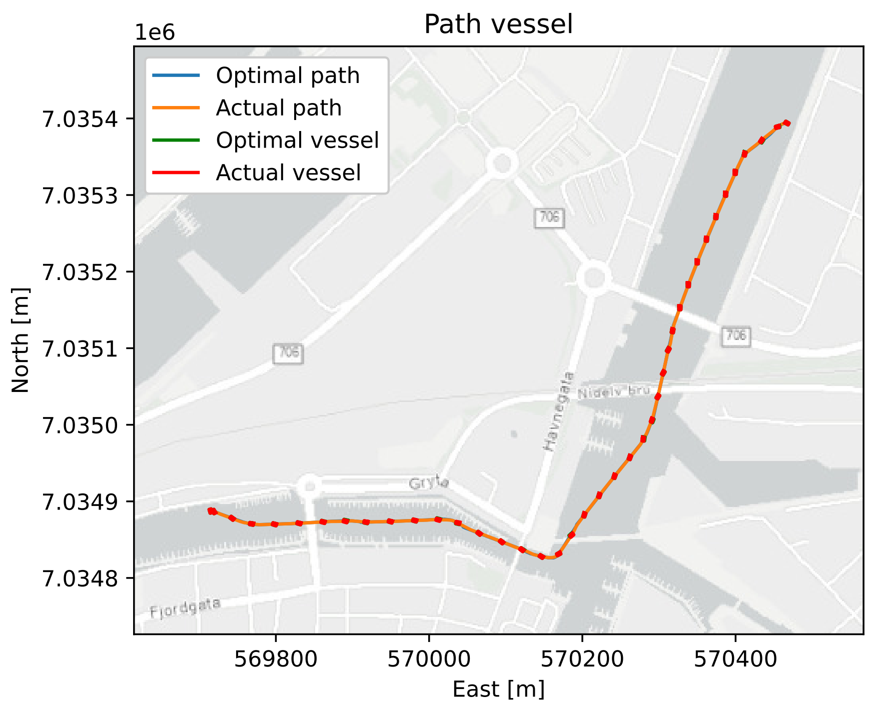


Figure 4.19: Overview of a trip between Nyhavna and Trondheim S. The map is slightly skewed in some places because of the earth's curvature. The vessel is plotted as a rectangle every 10s.

collision-free.

If we look at the plotted paths we can barely see the optimal path, since the actual path follows the optimal very well. The paths also look sensible, where the vessel travels in the middle of the channels. One observation we can make is that the vessel is not traveling with the front first, but rather with a front corner. This can be seen more clearly in Figure 4.20. This could be because the milliAmpere 1 vessel is almost squared, as seen in Figure 1.1, which means that going straight forward might not be the most efficient. Something else that might have an effect is the thrust configuration where one thruster drives and one brakes. The vessel might get more effect from the thrusters if the front thruster gives thrust in the sway direction, instead of braking or being inactive.

Looking at the surge and sway velocity in Figure 4.21 we see that also here the vessel is able to follow the optimal trajectory quite well. In the surge velocity plot we see some dips. This happens when the vessel reaches a connection point between two convex sets. Since the trajectory planner tries to get the vessel to the connection point, it slows down upon approaching. This effect could be avoided by increasing the circle of acceptance for reaching the connection point. However,

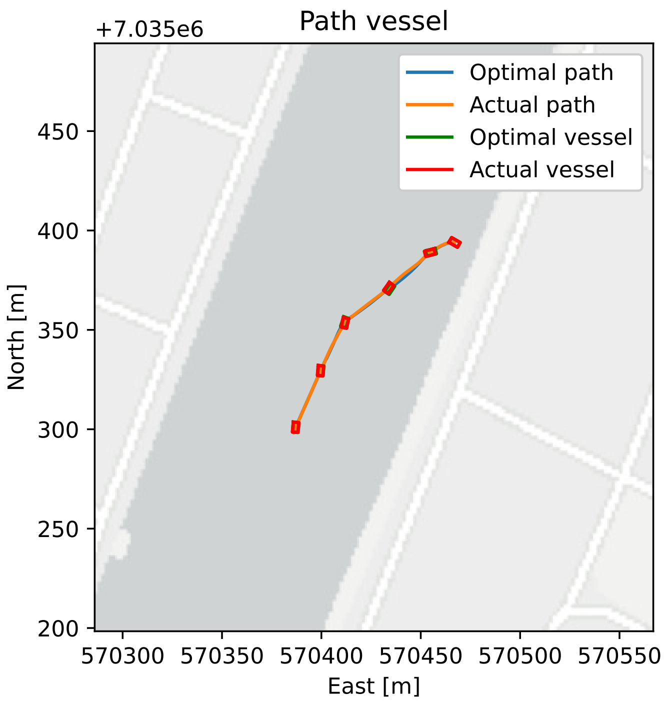


Figure 4.20: The first 51s of the trip from Nyhavna to Trondheim S, shown in Figure 4.19.

when the channel becomes narrow, a bigger circle of acceptance could mean that the vessel is outside the next convex set when reaching the connection point. A possible solution to this could be to try to increase the overlap between the areas, and combine it with a more complex acceptance criteria. As an example, an ellipse could be used instead of a simple circle.

We initially expected the sway velocity to be close to zero, when the vessel is not turning. However, since the vessel prefers to travel with a corner of the vessel pointing forward, the sway velocity is almost always saturated to $\pm 1m/s$. As previously discussed this might not be very bad, if it is more effective or faster. It might also be interesting to test this with passengers to see if it feels uncomfortable. Either way, it is difficult to force the vessel to go straight forward. Having a sway velocity of $\pm 1m/s$ might be natural when turning. Limiting the state to have a smaller magnitude in the trajectory planner has resulted in infeasible solutions when testing. Limiting the thrusters' sway components even more is a more viable solution, but some simple tests have shown that in order to get an effect it has to be limited to $\pm 100N$. This might limit the thrusters too much when the vessel is turning. The last possible solution is to allow both thrusters to drive when in transit. However, if the vessel has to brake suddenly, the front thruster might use too much time to turn around for the solution to be safe. One last observation about the sway velocity is

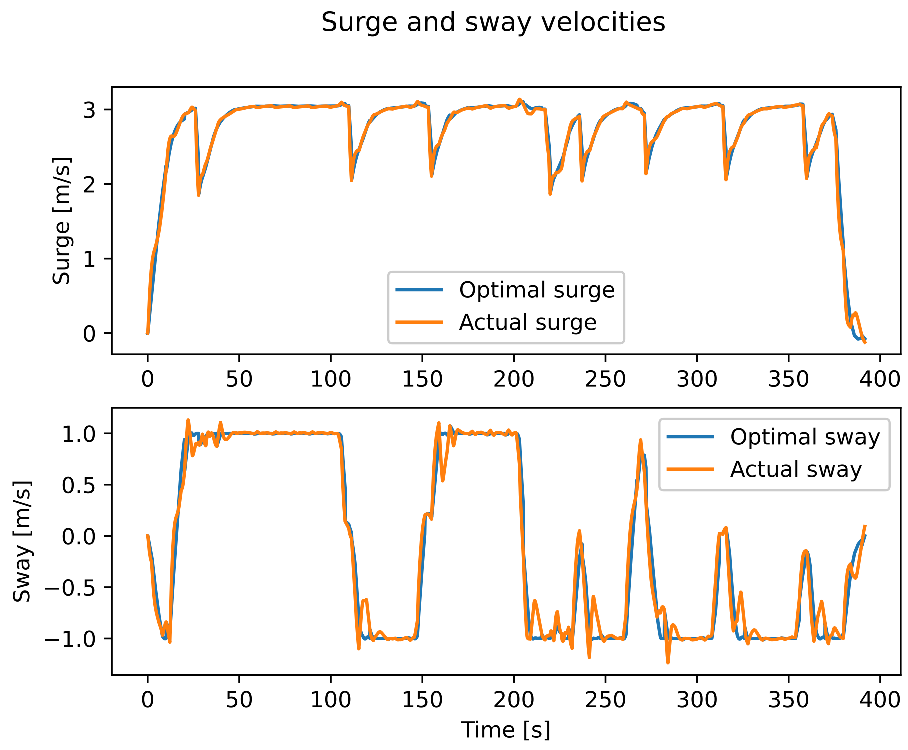


Figure 4.21: The surge and sway velocity of the vessel for the trip from Nyhavna to Trondheim S, shown in Figure 4.19.

that it is a little oscillatory. We have omitted the thruster dynamics, making them unrealistically responsive. This might be the cause of the oscillations, and will be discussed more when looking at the thruster plots.

Both the yaw and yaw rate track the optimal trajectory quite well as we see in Figure 4.22. Since the vessel can go both ways, the definition of which way is forward is simply changed at the beginning of the trip, as seen by the sudden change in yaw. This is done because the front should be towards land when docking, but towards the channel when undocking. Just like the sway velocity, the yaw rate is slightly oscillatory, which might also stem from the thruster dynamic being a little unrealistic.

In Figure 4.23 we can see the motor thrust from both thrusters, and in Figure 4.24 we see the angles of the thrusters. Furthermore, we see the decomposed forces from the thrusters in Figures 4.25 and 4.26. In all these figures we see that there is no time delay for changing thrust or the thruster angles. This is a simplification we did because we believed it would not have much effect. Furthermore, we use more powerful thrusters than the milliAmpere 1, such that the thruster dynamics would have to be estimated from the current thrusters [77]. When looking at the rapid changes, the thruster dynamic might have a significant effect, and it would be

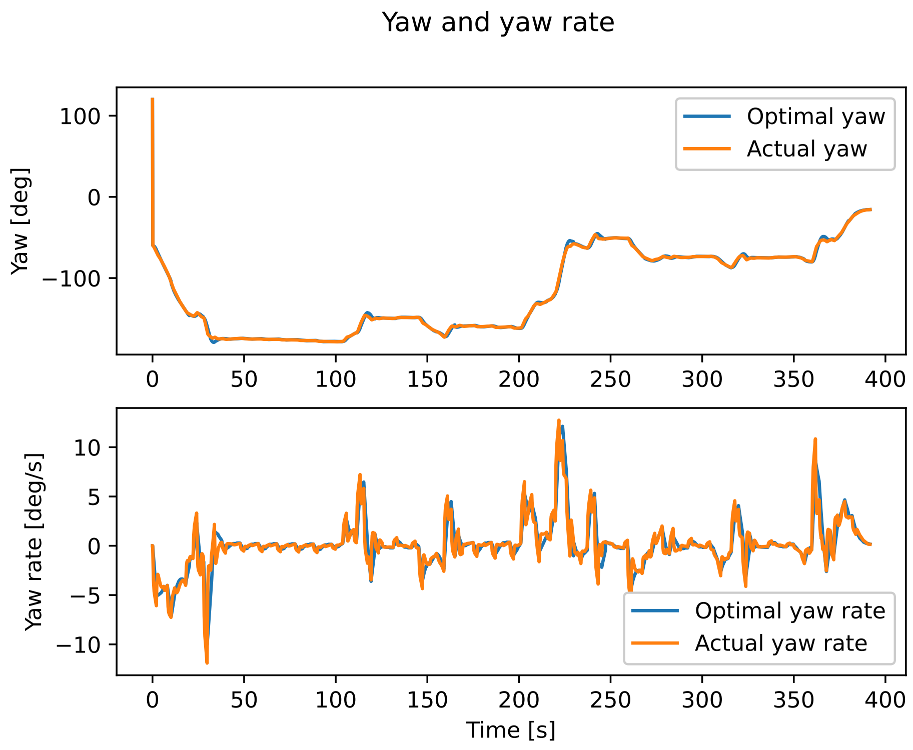


Figure 4.22: The yaw and yaw rate of the vessel for the trip from Nyhavna to Trondheim S, shown in Figure 4.19.

interesting to look at this in future work.

Looking at Figure 4.23 we see that the back thruster is often at maximum thrust of $1500N$. This is expected for the driving thruster since it is the fastest way to get to the connection points or docks. There are some negative spikes, which come when the vessel reaches a connection point as previously discussed. It might be hard to see from the plots, but there are spikes for both the optimal and actual thrust. The spikes are rather short, which means that the acceptance radius, for reaching a connection point, is almost big enough.

Another observation is that the optimal back thrust is above the maximum thrust quite often. This is because the decomposed forces from the thrusters are not connected in the trajectory planner. As we have seen from the vessel response, this simplification did not affect the vessel's ability to track the optimal states. Perhaps more strange is the actual thrust, which is not necessarily close to being saturated when the optimal thrust is above the maximum. It is hard to say why this is, but the optimal trajectory is made a little slower than it could be. This might be the reason the vessel is still able to track the optimal states. At the same time, the lower control parts try to track the optimal states and not thrust. Therefore another thrust configuration has then been deemed to be more optimal.

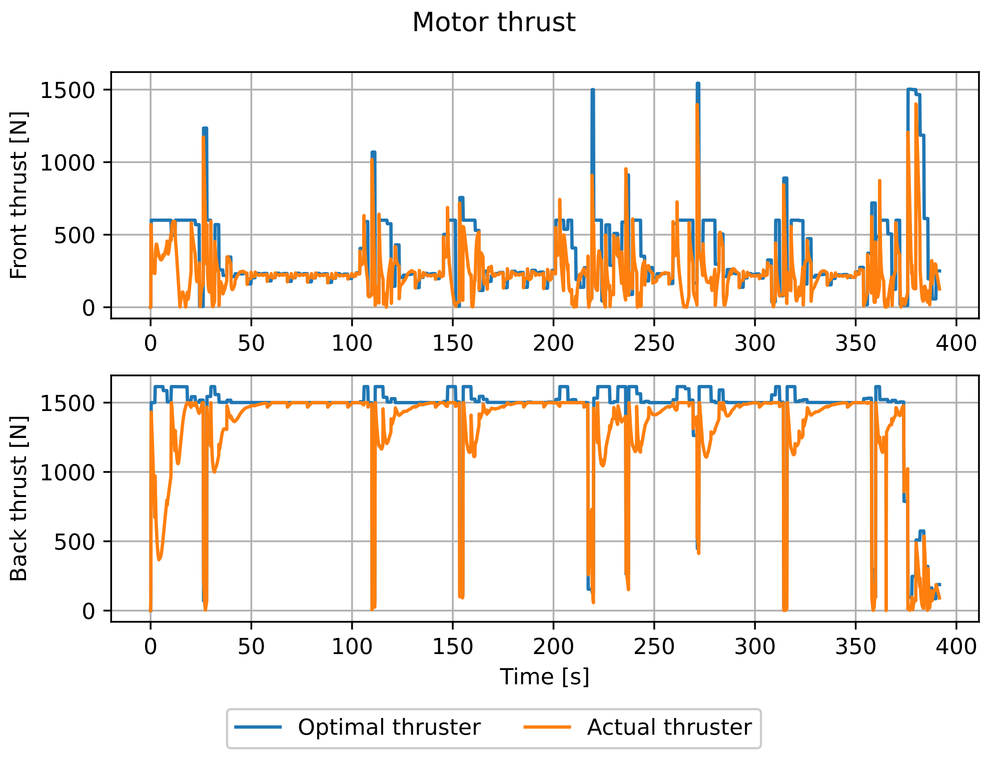


Figure 4.23: The thrust from each motor during the trip from Nyhavna to Trondheim S, shown in Figure 4.19.

Having the thruster be at their maximum throughout large parts of the trip is not necessarily the best way to use the thrusters with respect to tear and efficiency. One solution could be to penalize the use of thrust a lot harder. However, to be able to keep up with the routes we do not have that much to go on with respect to velocity. Therefore, the thrusters have to be able to produce around $1500N$ in normal operations. The results in Section 4.1 showed us that larger ferries are needed, so this has to be looked further at in future work anyway. However, the vessels have to be able to operate at around $3m/s$ in normal operations.

If we look at the front thruster in Figure 4.23 we see that it is quite active. Our initial thought was that this thruster would only be active when turning or while braking when approaching the dock. However, as discussed previously, there has been some unforeseen behavior. The first is that the vessel is traveling with a saturated sway velocity, which is produced using the front thruster. Next, is the braking upon reaching connection points, where the back thruster is turned down and the front thruster is turned up. The front thruster does not follow the optimal thrust closely either. The reason for this is probably the same as for the back thruster.

The angle of the back thruster, shown in Figure 4.24, is about what is expected. It is mostly around 0°, meaning all thrust is going forward. When the angle is spiking,

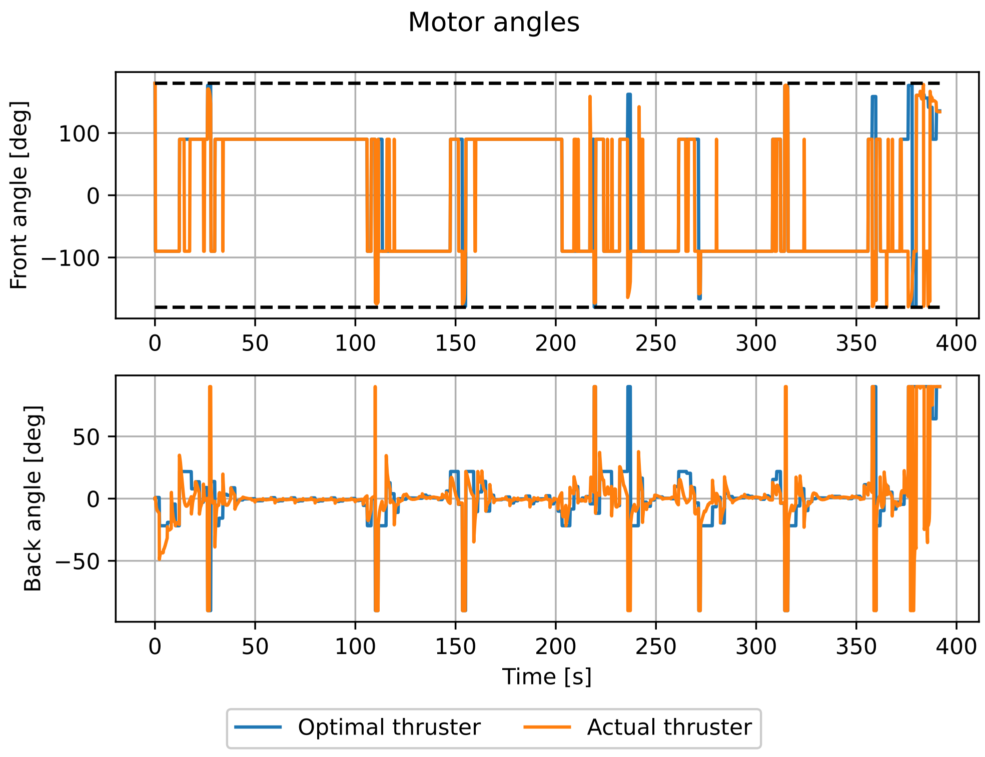


Figure 4.24: The angles of the azimuth thrusters, as shown in Figure 2.3, during the trip from Nyhavna to Trondheim S, shown in Figure 4.19.

it corresponds to the dips in thrust. This means that the thruster is used in the sway direction instead. The spikes often go between ±90° in rapid succession, which is not very realistic.

The angle of the front thruster is going between ±90° quite often. This is not very realistic behavior, and would result in a lot of tear on the thrusters. Since the thruster is active throughout the whole trip it is not a surprise that the angle is either ±90°, since this means that the thrust is only in the sway direction. There are some spikes where the angle goes to ±180°, and these correspond with reaching a connection point.

Looking at the decomposed forces in Figure 4.25, we see the same effects as previously discussed more clearly. The front thruster brakes when approaching connection points and at the end. Meanwhile, the sway component is rather noisy, and in some ways similar to the sway velocity plot. Where the sway velocity does not change much, the sway component is rather stable. However, where the sway component is noisy, the sway velocity is oscillatory. This is, of course, not a surprise since sway velocity is an effect of the thrust in this direction. However, it would probably mean that by using the actual thruster dynamics the thrust would be less noisy, such that the sway velocity would be less oscillatory. Using the thruster dynamics could also

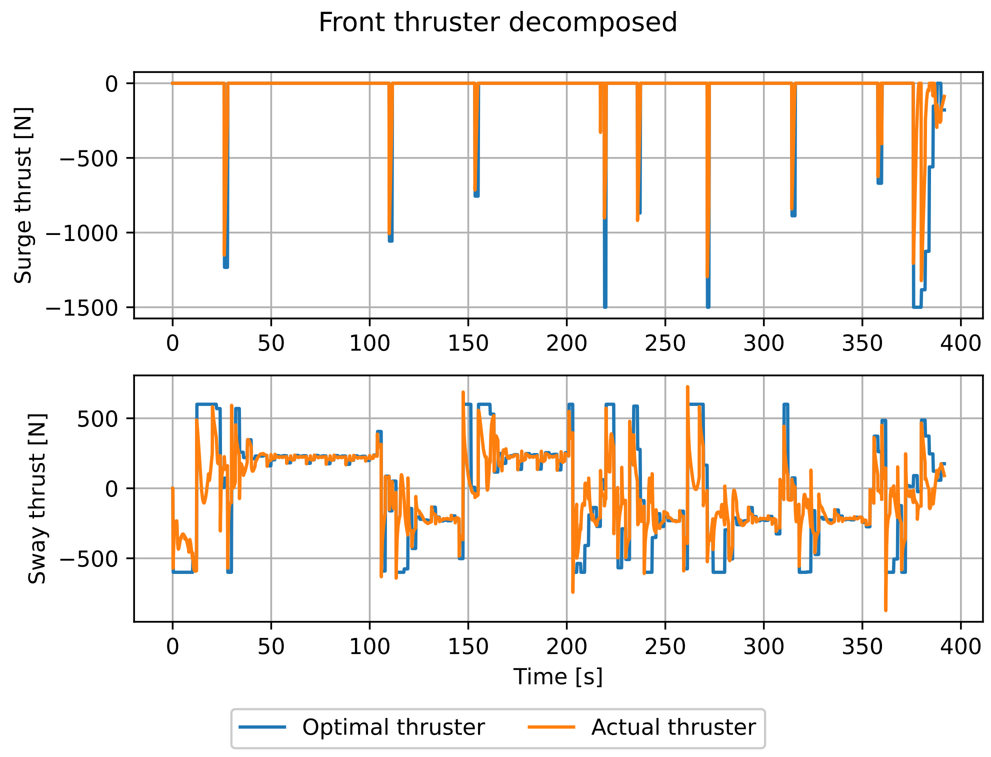


Figure 4.25: The decomposed forces from the front thruster during the trip from Nyhavna to Trondheim S, shown in Figure 4.19.

mean that the dips in the surge component would be smaller, such that the dips in surge velocity would be smaller.

The surge component of the back thruster, shown in Figure 4.26, is almost identical to the overall thrust in Figure 4.23. This is not very surprising as the back thruster's main task is to drive the vessel forward. The sway component is often around $0N$, and is mostly active around the connection points.

4.4 Limitations of AI-Planning

Since AI-planning is not as commonly used as tools such as optimization or machine learning, there are some major problems. This includes the availability of planners, bugs in implementation and performance of planners. In Figure 4.27 an illustration of limiting factors are shown. The figure is made in collaboration with Øystein Solbø, who has faced similar problems [81]. In this subsection the figure will be explained in more detail, before some problems specific to our planning problem are presented.

At the bottom of the pyramid in Figure 4.27 all planners are present. This layer

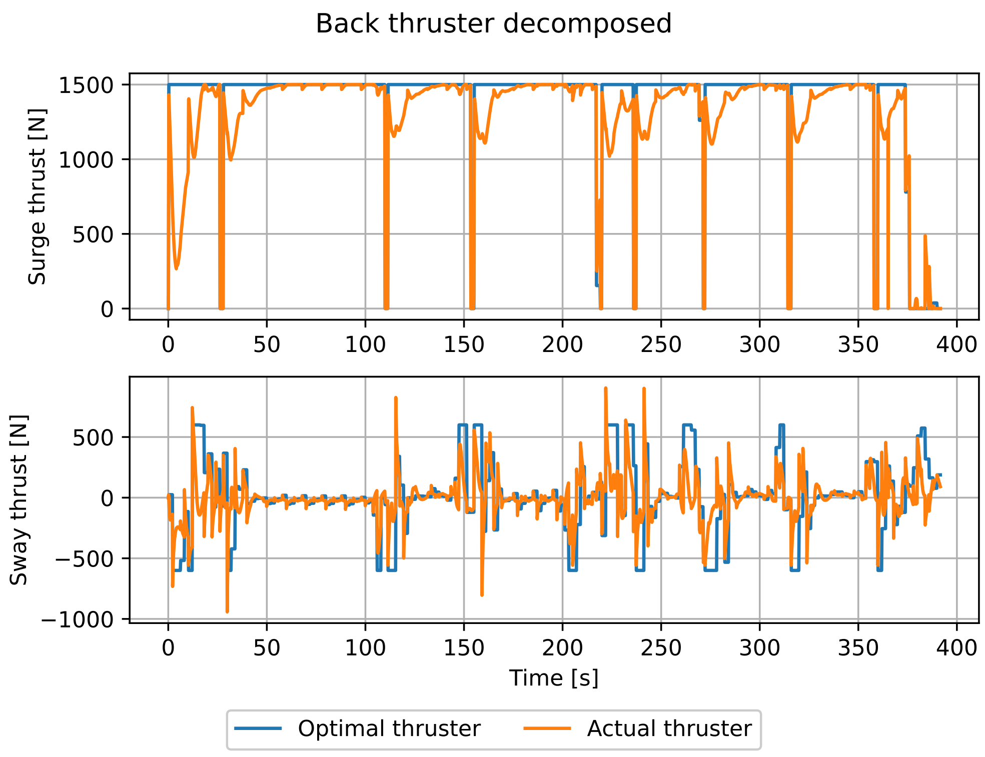
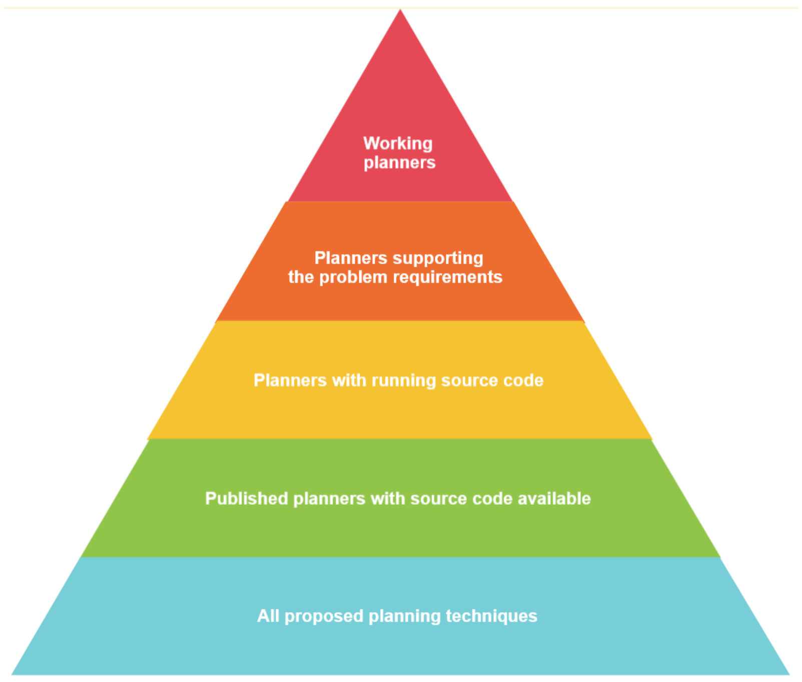


Figure 4.26: The decomposed forces from the back thruster during the trip from Nyhavna to Trondheim S, shown in Figure 4.19.



Working planners
Planners supporting the problem requirements
Planners with running source code
Published planners with source code available
All proposed planning techniques

Figure 4.27: An illustration of limiting factors when using AI-planning. The figure is made in collaboration with Øystein Solbø, who has faced similar problems [81].

is quite large, as there have been made a lot of different planners. The layer above is unfortunately significantly decreased, as finding a planner with published source code, or other ways to use the algorithm, is quite rare. Most available planners are published as a contestant in one of the planning competitions held by the International Conference on Automated Planning and Scheduling.

The next layer refers to the difficulties in getting the source code to run. Unfortunately, the source code is seldom documented very well, if it is documented at all. Therefore finding out how to run the code and what tools are needed is time-consuming, and requires in-depth knowledge about the tools. At times it might even result in conflicts with the rest of the system.

Almost at the top of the pyramid is the limitation of requirements. There are several possible requirements, so finding a planner that supports your needed requirements is difficult. Furthermore, the published source code of a planner sometimes does not support everything the proposed planner is said to support. At the top of the pyramid, there are the bugs in the code that make the code crash or terminate even though the planner is supposed to support the wanted functionalities. This layer also includes problems planners have with certain problems. An example of this is that planners usually have problems with concurrent actions.

While searching for planners, two planners reached the top layer of the pyramid. This was the POPF planner and the OPTIC planner, where the latter had fewer issues in the top layer. However, some limiting problems have still been encountered. One of the main problems is the performance in finding solutions. If the problem becomes too big, the planner can search for several minutes without finding a good enough solution, before suddenly crashing. This would mean that we have to plan for smaller time horizons. The problem becomes worse when there are concurrent actions, which we will have with multiple vessels. This means that the planning problem we make has to be rather simple for the planner to handle it.

A lot of planning problems revolve around moving goods [26]. However, when the goods are passengers there is an extra challenge in the sense that passengers care about how long they wait. Therefore, a fixed timetable makes it convenient for passengers to arrive just before departure. Adding a departure time to a planning problem is not an arbitrary task. Since the time an action start is not available to use in an action, we have to look at some tricks.

The simplest way to add a departure time would be to use time initialed predicates, which is a requirement the available OPTIC version [1] does not support. Time initial predicates can be substituted using actions that have to start at time zero in the plan [82], however this adds complexity such that the rest of the problem has to be even more simple. Another solution is to use a timer action that is forced to start at time zero, and continuously increase a timer function. However, continuously increasing a function is a functionality that OPTIC allows when parsing the problem, but when trying to solve the problem, a bug makes the planner crash.

An alternative approach to using a timetable is a pure on demand solution where the planner is given the full information about how many passengers that have arrived and where they want to go. When testing this solution we encountered problems using numerical variables. First, we discovered that the two different versions of POPF we tried did not manage to solve problems where a numeric variable had to obtain a certain value. In this case, a goal is that there are no more waiting passengers. OPTIC managed to decrease numeric variables, but was not able to solve a problem where a ferry had to leave passengers behind when it was full. It also crashed when we had two ferries, and it would be optimal to use both. The last test we did was to see if it managed to move two passengers from Nyhavna. One was going to Solsiden and one to Trondheim S. This also resulted in the planner crashing.

Chapter 5

Conclusion

In this thesis we reached our primary objective by showing how a small fleet of autonomous passenger ferries can be used as an alternative transportation method in Trondheim. In order to do this we have shown how a high-level mission plan can be split up into tasks such as docking, undocking, transit and loading, as well as how to make a control system that can translate these tasks to input for the thrusters. In addition, we showed how AI-planning could be used to make a high-level mission plan. To simulate the distribution of passengers we made a stochastic simulation that added a non-negative number of passengers at each port every minute.

To look at the different capabilities of the system we simulated five different scenarios. For all five scenarios the average waiting time was between ten and ten and a half minutes. This is quite good as the waiting time includes the time spent on the ferries, and it is most popular to take the longest trip which takes around eight to nine minutes. To follow the routes set by the predefined timetable two ferries are needed for most of the day. In the scenario with only two ferries, with a capacity of 10 people, the system is almost able to handle around 700 passengers a day. However, an average of about two passengers a day were left behind because the ferries were filled up to their max capacity. When having a capacity of 25 passengers, which is the same as a similar ferry system about to be launched in Stockholm [9], the ferries always had enough space. Even though that scenario also doubled the number of passengers, the capacity was more than doubled, so this was not very surprising. However, it showed that using already existing ferries makes it possible to handle more than enough passengers.

In the other three scenarios we had three ferries instead of two. In the scenario where we only added a third ferry, the system found that it only needed more capacity between Trondheim S and Solsiden between 15:00 and 16:00. This slightly decreased the average waiting time and the average passengers left behind throughout the day. As the total usage of the ferries only increased with an hour this is quite good. However, the ferry that was taking extra capacity routes was much less busy than the ones following the timetable. This indicates that it could be possible to utilize the third ferry better. The system calculates the extra needed capacity based on the expected number of passengers. Since we added some significant uncertainty, one scenario gave the planner a higher expected number of passengers. This resulted in using all three ferries for the six busiest hours of the day, and a further decrease in the average waiting time and number of passengers left behind. In the last scenario with three ferries, we added a simple recharging scheme where the most used ferry had to take an hour break if it had been active for three hours straight. Unsurprisingly, this resulted in a much more even distribution of workload between the ferries. It also showed the robustness of the system, as it was able to plan with an unavailable ferry and the average waiting time was mostly unaffected.

The control system was able to handle every task it was given. When we looked at one of the longest trips, we saw that the trajectory planner gave a sensible collision-free path. However, at the connection point between convex sets the vessel started to brake until it was close enough to the point. It then increased the velocity again and kept going. The trajectory tracker and thrust allocation were able to follow the path very closely. As a simplification, the thruster dynamics were not included in the vessel model or the control system. This meant both the trajectory planner and thrust allocator were allowed to change the vessel input infinitely fast. A little surprisingly, this was something the control system exploited often, by frequently changing the thrust and angles of the azimuth thruster between their minimum and maximum limits.

With regards to our secondary objective, the AI-planner makes sure that the ferries are distributed among the needed routes and that onboard passengers reach their destination after a ferry become inactive. It is also compatible with the rest of the system by translating the actions in the plan into tasks for the individual ferries. Some bugs made some plans suboptimal, and one compatibility issue with double loading tasks was encountered. These would have to be looked at in future work. However, it might be best to look at the whole planning module as well. While the planner does its job reasonably well, similar plans can be achieved by predefining what route each ferry should take. By ending every route at either Nyhavna or Trondheim S, the passengers will also reach their destination. In Section 4.4 we discussed alternative formulations of the planning problem, and why it failed. To get a more intelligent system, these approaches may be tried again with a different planner. It might also be an idea to use other tools for mission planning, such as optimization or more classical scheduling algorithms. The reason why the planner is activated every hour is because the passenger distribution changes every hour and because it was perceived to be a reasonable time horizon for planning. However, one could also make a planning problem for the whole day, as long as the possible actions are so high-level that the planner can handle the number of actions. Another approach could be to have predefined routes, and activate the planner if some anomaly is detected. This could be a delay for a ferry, or a spike in waiting passengers at a port.

To make sure the vessels were able to follow the timetable we had to use a different maximum thrust than the thrusters on milliAmpere 1 actually give. A higher maximum passenger capacity is also required. Therefore, another vessel has to be used, with a capacity of at least 10 passengers and an operational velocity of around $3m/s$. This would mean that the control system has to be modified to work for the new vessel. In addition, the connection point problem has to be looked at, and the thruster dynamics should be added to the model and the control system. Another control problem that has been omitted for simplicity is collision avoidance. This has to be added between the ferries, to make sure a possible collision is never encountered. The channel is also used by other vessels, so a full collision avoidance scheme has to be implemented. How much the forces from the river affect the vessels has to be investigated as well. If the system is ever launched in real life, it would also be a good idea to fetch data for the actual passenger distribution. This could be used for making even more intelligent decisions.

Bibliography

[1] Dongbo Xie. *optic-clp-release.* https://github.com/Dongbox/optic-clp-release. 2022.

[2] Elvira Ismagilova et al. 'Smart cities: Advances in research—An information systems perspective'. In: *International journal of information management* 47 (2019), pp. 88–100.

[3] M. S. Tannum and J. H. Ulvensøen. 'Urban mobility at sea and on waterways in Norway'. In: *Journal of Physics: Conference Series* 1357.1 (Oct. 2019), p. 012018. DOI: 10.1088/1742-6596/1357/1/012018. URL: https://dx.doi.org/10.1088/1742-6596/1357/1/012018.

[4] Namireddy Praveen Reddy et al. 'Zero-emission autonomous ferries for urban water transport: Cheaper, cleaner alternative to bridges and manned vessels'. In: *IEEE Electrification Magazine* 7.4 (2019), pp. 32–45.

[5] Rolls-Royce marine. *Rolls-Royce and Finferries demonstrate world's first Fully Autonomous Ferry.* https://www.rolls-royce.com/media/press-releases/2018/03-12-2018-rr-and-finferries-demonstrate-worlds-first-fully-autonomous-ferry.aspx. [accessed 21-September-2022]. 2018.

[6] ABB. *ABB enables groundbreaking trial of remotely operated passenger ferry.* https://new.abb.com/news/detail/11632/abb-enables-groundbreaking-trial-of-remotely-operated-passenger-ferry. [accessed 14-April-2023]. 2018.

[7] Konsberg-Maritime. *Automatic ferry enters regular service following world-first crossing with passengers onboard.* https://www.kongsberg.com/maritime/about-us/news-and-media/news-archive/2020/first-adaptive-transit-on-bastofosen-vi/. [accessed 11-December-2022]. 2020.

[8] Norwegian SciTech News. *NTNU trials world's first urban autonomous passenger ferry.* https://norwegianscitechnews.com/2022/09/ntnu-trials-worlds-first-urban-autonomous-passenger-ferry/. [accessed 11-December-2022]. 2022.

[9] Brødrene Aa. *Brødrene Aa is building autonomous ferry.* https://www.braa.no/news/brodrene-aa-is-building-the-worlds-first-driverless-ferry. [accessed 14-April-2023]. 2022.

[10] Helge Martin Markussen. *Ship of The Year 2023 - Her er de nominerte.* https://www.skipsrevyen.no/bluewild-brodrene-aa-ecofibras/ship-of-the-year-2023-her-er-de-nominerte/1511018. [accessed 28-April-2023]. 2023.

[11] The Guardian. *Roboats: Amsterdam to trial self-driving electric boats.* https://www.theguardian.com/world/2021/jun/03/roboats-amsterdam-to-trial-self-driving-electric-boats. [accessed 14-April-2023]. 2021.

[12] Trondheim kommune. *Etablerer felles eiendomsselskap for å utvikle Nyhavna til ny bydel.* https://www.trondheim.kommune.no/aktuelt/nyhetssaker/2021/etablerer-felles-eiendomsselskap--for-a-utvikle-nyhavna-til-ny-bydel/. [accessed 14-April-2023]. 2023.

[13] MA Hinostroza and Anastasios M Lekkas. 'A Rudimentary Mission Planning System for Marine Autonomous Surface Ships'. In: *IFAC-PapersOnLine* 55.31 (2022), pp. 196–203.

[14] Malik Ghallab, Dana Nau and Paolo Traverso. *Automated planning and acting.* Cambridge University Press, 2016.

[15] Richard E Fikes and Nils J Nilsson. 'STRIPS: A new approach to the application of theorem proving to problem solving'. In: *Artificial intelligence* 2.3-4 (1971), pp. 189–208.

[16] Gregg Rabideau et al. 'Iterative repair planning for spacecraft operations using the ASPEN system'. In: *Artificial Intelligence, Robotics and Automation in Space.* Vol. 440. 1999, p. 99.

[17] Douglas Bernard et al. 'Spacecraft autonomy flight experience: The DS1 Remote Agent experiment'. In: (1999).

[18] Amanda Coles et al. 'Forward-chaining partial-order planning'. In: *Proceedings of the International Conference on Automated Planning and Scheduling.* Vol. 20. 2010, pp. 42–49.

[19] J Benton, Amanda Coles and Andrew Coles. 'Temporal planning with preferences and time-dependent continuous costs'. In: *Proceedings of the International Conference on Automated Planning and Scheduling.* Vol. 22. 2012, pp. 2–10.

[20] Michael Cashmore et al. 'AUV mission control via temporal planning'. In: *2014 IEEE international conference on robotics and automation (ICRA).* IEEE. 2014, pp. 6535–6541.

[21] Michael Cashmore et al. ‘Rosplan: Planning in the robot operating system’. In: *Proceedings of the international conference on automated planning and scheduling*. Vol. 25. 2015, pp. 333–341.

[22] Sara Bernardini, Maria Fox and Derek Long. ‘Planning the behaviour of low-cost quadcopters for surveillance missions’. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 24. 2014, pp. 445–453.

[23] Sara Bernardini, Maria Fox and Derek Long. ‘Combining temporal planning with probabilistic reasoning for autonomous surveillance missions’. In: *Autonomous Robots* 41 (2017), pp. 181–203.

[24] Isabel Cenamor et al. ‘Temporal: Temporal portfolio algorithm’. In: *Proceedings of ICAPS International Planning Competition* (2018).

[25] D Furelos-Blanco and A Jonsson. *CP4TP: A Classical Planning for Temporal Planning Portfolio*. 2018.

[26] Alejandro Torreno et al. ‘Cooperative multi-agent planning: A survey’. In: *ACM Computing Surveys (CSUR)* 50.6 (2017), pp. 1–32.

[27] Matthew Crosby and Ronald PA Petrick. ‘Temporal multiagent planning with concurrent action constraints’. In: *Proc 2nd ICAPS Workshop on Distributed and Multi-Agent Planning. ICAPS*. 2014, pp. 16–24.

[28] Jonas Kvarnström. ‘Planning for loosely coupled agents using partial order forward-chaining’. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 21. 2011, pp. 138–145.

[29] Fletcher Thompson and Damien Guihen. ‘Review of mission planning for autonomous marine vehicle fleets’. In: *Journal of Field Robotics* 36.2 (2019), pp. 333–354.

[30] Martin Ludvigsen et al. ‘Network of heterogeneous autonomous vehicles for marine research and management’. In: *OCEANS 2016 MTS/IEEE Monterey*. IEEE. 2016, pp. 1–7.

[31] Trygve Olav Fossum et al. ‘Information-driven robotic sampling in the coastal ocean’. In: *Journal of Field Robotics* 35.7 (2018), pp. 1101–1121.

[32] Yaniel Carreno, Yvan Petillot and Ronald PA Petrick. ‘Multi-Vehicle Temporal Planning for Underwater Applications’. In: *Proceedings of ICAPS Workshop on Planning and Robotics*. 2019.

[33] Yaniel Carreno et al. ‘A decentralised strategy for heterogeneous auv missions via goal distribution and temporal planning’. In: *Proceedings of the international conference on automated planning and scheduling.* Vol. 30. 2020, pp. 431–439.

[34] Lanah Evers et al. ‘Robust UAV mission planning’. In: *Annals of Operations Research* 222 (2014), pp. 293–315.

[35] Bariş Başpinar, Hamsa Balakrishnan and Emre Koyuncu. ‘Mission planning and control of multi-aircraft systems with signal temporal logic specifications’. In: *IEEE Access* 7 (2019), pp. 155941–155950.

[36] Felix Heilemann and Axel Schulte. ‘Interaction concept for mixed-initiative mission planning on multiple delegation levels in multi-UCAV fighter missions’. In: *Intelligent Human Systems Integration 2019: Proceedings of the 2nd International Conference on Intelligent Human Systems Integration (IHSI 2019): Integrating People and Intelligent Systems, February 7-10, 2019, San Diego, California, USA*. Springer. 2019, pp. 699–705.

[37] Tolga Bektas. ‘The multiple traveling salesman problem: an overview of formulations and solution procedures’. In: *omega* 34.3 (2006), pp. 209–219.

[38] Morten Breivik and Jon-Erik Loberg. ‘A virtual target-based underway docking procedure for unmanned surface vehicles’. In: *IFAC Proceedings Volumes* 44.1 (2011), pp. 13630–13635.

[39] Joohyun Woo and Nakwan Kim. ‘Vector field based guidance method for docking of an unmanned surface vehicle’. In: *The Twelfth ISOPE Pacific/Asia Offshore Mechanics Symposium.* OnePetro. 2016.

[40] Namkyun Im et al. ‘A study on ship automatic berthing with assistance of auxiliary devices’. In: *International Journal of Naval Architecture and Ocean Engineering* 4.3 (2012), pp. 199–210.

[41] Yonghui Shuai et al. ‘An efficient neural-network based approach to automatic ship docking’. In: *Ocean Engineering* 191 (2019), p. 106514.

[42] Naoki Mizuno, Yosuke Uchida and Tadatsugi Okazaki. ‘Quasi real-time optimal control scheme for automatic berthing’. In: *IFAC-PapersOnLine* 48.16 (2015), pp. 305–312.

[43] Andreas B Martinsen, Anastasios M Lekkas and Sebastien Gros. ‘Autonomous docking using direct optimal control’. In: *IFAC-PapersOnLine* 52.21 (2019), pp. 97–102.

[44] Glenn Bitar et al. ‘Trajectory planning and control for automatic docking of ASVs with full-scale experiments’. In: *IFAC-PapersOnLine* 53.2 (2020), pp. 14488–14494.

[45] Edmund F Brekke et al. ‘milliampere: An autonomous ferry prototype’. In: *Journal of Physics: Conference Series*. Vol. 2311. 1. IOP Publishing. 2022, p. 012029.

[46] Petter K Ødven, Andreas B Martinsen and Anastasios M Lekkas. ‘Static and dynamic multi-obstacle avoidance and docking of ASVs using computational geometry and numerical optimal control’. In: *IFAC-PapersOnLine* 55.31 (2022), pp. 50–57.

[47] Yoshiki Miyauchi et al. ‘Optimization on planning of trajectory and control of autonomous berthing and unberthing for the realistic port geometry’. In: *Ocean Engineering* 245 (2022), p. 110390.

[48] Glenn Bitar et al. ‘Three-phase automatic crossing for a passenger ferry with field trials’. In: *2021 European Control Conference (ECC)*. IEEE. 2021, pp. 2271–2277.

[49] Glenn Bitar et al. ‘Warm-started optimized trajectory planning for ASVs’. In: *IFAC-PapersOnLine* 52.21 (2019), pp. 308–314.

[50] Andreas B Martinsen, Anastasios M Lekkas and Sébastien Gros. ‘Optimal model-based trajectory planning with static polygonal constraints’. In: *IEEE Transactions on Control Systems Technology* 30.3 (2021), pp. 1159–1170.

[51] Anete Vagale et al. ‘Path planning and collision avoidance for autonomous surface vehicles I: a review’. In: *Journal of Marine Science and Technology* (2021), pp. 1–15.

[52] Mauro Candeloro, Anastasios M Lekkas and Asgeir J Sørensen. ‘A Voronoi-diagram-based dynamic path-planning system for underactuated marine vessels’. In: *Control Engineering Practice* 61 (2017), pp. 41–54.

[53] Hanlin Niu et al. ‘An energy-efficient path planning algorithm for unmanned surface vehicles’. In: *Ocean Engineering* 161 (2018), pp. 308–321.

[54] Yin Cheng and Weidong Zhang. ‘Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels’. In: *Neurocomputing* 272 (2018), pp. 63–73.

[55] Asgeir J Sørensen, Svein I Sagatun and Thor I Fossen. ‘Design of a dynamic positioning system using model-based control’. In: *Control Engineering Practice* 4.3 (1996), pp. 359–368.

[56] Aleksander Veksler et al. ‘Dynamic positioning with model predictive control’. In: *IEEE Transactions on Control Systems Technology* 24.4 (2016), pp. 1340–1353.

[57] Margarita V Sotnikova and Evgeny I Veremey. ‘Dynamic positioning based on nonlinear MPC’. In: *IFAC Proceedings Volumes* 46.33 (2013), pp. 37–42.

[58] Emil H Thyri, Glenn Bitar and Morten Breivik. ‘A 3DOF path-following controller for a non-directionally stable vessel with slow thruster dynamics’. In: *IFAC-PapersOnLine* 54.16 (2021), pp. 288–294.

[59] Thor I Fossen and Tor A Johansen. ‘A survey of control allocation methods for ships and underwater vehicles’. In: *2006 14th Mediterranean Conference on Control and Automation*. IEEE. 2006, pp. 1–6.

[60] Tor A Johansen and Thor I Fossen. ‘Control allocation—A survey’. In: *Automatica* 49.5 (2013), pp. 1087–1103.

[61] Ole J Sørdalen. ‘Optimal thrust allocation for marine vessels’. In: *Control Engineering Practice* 5.9 (1997), pp. 1223–1231.

[62] Eivind Ruth. ‘Propulsion control and thrust allocation on marine vessels’. In: (2008).

[63] Tobias R Torben, Astrid H Brodtkorb and Asgeir J Sørensen. ‘Control allocation for double-ended ferries with full-scale experimental results’. In: *IFAC-PapersOnLine* 52.21 (2019), pp. 45–50.

[64] Inger Berge Hagen. *Christening of the ferry*. https://www.ntnu.edu/web/autoferry/news/-/blogs/christenning-of-the-ferry?_com_liferay_blogs_web_portlet_BlogsPortlet_redirect=https%3A%2F%2Fwww.ntnu.edu%2Fweb%2Fautoferry%2Fnews%3Fp_p_id%3Dcom_liferay_blogs_web_portlet_BlogsPortlet%26p_p_lifecycle%3D0%26p_p_state%3Dnormal%26p_p_mode%3Dview%26_com_liferay_blogs_web_portlet_BlogsPortlet_cur%3D1%26_com_liferay_blogs_web_portlet_BlogsPortlet_delta%3D20. [accessed 27-April-2023]. 2021.

[65] Peter E Hart, Nils J Nilsson and Bertram Raphael. ‘A formal basis for the heuristic determination of minimum cost paths’. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[66] Avrim L Blum and Merrick L Furst. ‘Fast planning through planning graph analysis’. In: *Artificial intelligence* 90.1-2 (1997), pp. 281–300.

[67] Maria Fox and Derek Long. ‘PDDL2. 1: An extension to PDDL for expressing temporal planning domains’. In: *Journal of artificial intelligence research* 20 (2003), pp. 61–124.

[68] Amanda Jane Coles et al. ‘Temporal Planning in Domains with Linear Processes.’ In: *IJCAI*. 2009, pp. 1671–1676.

[69] Andrew Coles et al. ‘Planning with Problems Requiring Temporal Coordination.’ In: *AAAI*. 2008, pp. 892–897.

[70] Keith Halsey, Derek Long and Maria Fox. ‘CRIKEY-a temporal planner looking at the integration of scheduling and planning’. In: *Workshop on Integrating Planning into Scheduling, ICAPS*. Citeseer. 2004, pp. 46–52.

[71] Andrew Coles et al. ‘Cost-sensitive concurrent planning under duration uncertainty for service-level agreements’. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 21. 2011, pp. 34–41.

[72] Amanda Coles et al. ‘POPF2: A forward-chaining partial order planner’. In: *The 2011 International Planning Competition* 65 (2011).

[73] Jörg Hoffmann and Bernhard Nebel. ‘The FF planning system: Fast plan generation through heuristic search’. In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 253–302.

[74] Jörg Hoffmann. ‘Extending FF to numerical state variables’. In: *ECAI*. Vol. 2. Citeseer. 2002, pp. 571–575.

[75] Society of Naval Architects, Marine Engineers (U.S.). Technical and Research Committee. Hydrodynamics Subcommittee. *Nomenclature for Treating the Motion of a Submerged Body Through a Fluid: Report of the American Towing Tank Conference*. Technical and research bulletin. Society of Naval Architects and Marine Engineers, 1950.

[76] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2021.

[77] Anders Aglen Pedersen. ‘Optimization based system identification for the milliAmpere ferry’. MA thesis. NTNU, 2019.

[78] Christian Kirches et al. ‘Efficient direct multiple shooting for nonlinear model predictive control on long horizons’. In: *Journal of Process Control* 22.3 (2012), pp. 540–550.

[79] Petter Knutsen Ødven. ‘Static and dynamic multi-obstacle avoidance for docking of ASVs using computational geometry and numerical optimal control’. MA thesis. NTNU, 2022.

[80] Moritz Diehl and Sébastien Gros. ‘Numerical optimal control’. In: *Optimization in Engineering Center (OPTEC)* (2011).

[81] Øystein Solbø. ‘Towards mission planning for search and rescue at sea’. MA thesis. NTNU, 2023.

[82] Maria Fox, Derek Long and Keith Halsey. ‘An investigation into the expressive power of PDDL2. 1’. In: *ECAI*. Vol. 16. 2004, p. 338.

Appendix A

milliAmpere 1 Characteristics and Parameters

The characteristics of milliAmpere 1 are given in Table A.1. Parameters for the 3-DOF model [77] are given in Table A.2. The elements of the dampening matrix $\boldsymbol{D}$, introduced in Section 2.2.3, are presented in Equation (A.1).

$$d_{11}(\boldsymbol{\nu}) = -X_u - X_{|u|u}|u| - X_{uuu}u^2 \tag{A.1a}$$

$$d_{22}(\boldsymbol{\nu}) = -Y_v - Y_{|v|v}|v| - Y_{|r|v}|r| - Y_{vvv}v^2 \tag{A.1b}$$

$$d_{23}(\boldsymbol{\nu}) = -Y_r - Y_{|v|r}|v| - Y_{|r|r}|r| \tag{A.1c}$$

$$d_{32}(\boldsymbol{\nu}) = -N_v - N_{|v|v}|v| - N_{|r|v}|r| \tag{A.1d}$$

$$d_{33}(\boldsymbol{\nu}) = -N_r - N_{|v|r}|v| - N_{|r|r}|r| - N_{rrr}r^2 \tag{A.1e}$$

Table A.1: Characteristics of milliAmpere 1 [63, 77]. In this thesis we have used $T_{max} = 1500N$.

Description	Symbol	Value
Hull length	LOA	$5m$
Hull width	B	$2.8m$
Max thrust	T_{max}	$500N$
Thruster distance from center	L_x	$1.8m$

Table A.2: Parameters for the 3-DOF model of milliAmpere 1 introduced in Section 2.2.3 [77].

Parameter	**Value**	**Unit**
m_{11}	2389.657	kg
m_{22}	2533.911	kg
m_{23}	62.386	kg
m_{32}	28.141	kg
m_{33}	5068.910	kgm^2
X_u	-27.632	kg/s
$X_{u\|u}$	-110.064	kg/s
X_{uuu}	-13.965	kg/s
Y_v	-52.947	kg/s
$Y_{\|v\|v}$	-116.486	kg/s
Y_{vvv}	-24.313	kg/s
$Y_{\|r\|v}$	-1540.383	kg/s
Y_r	24.732	kg/s
$Y_{\|v\|r}$	572.141	kg/s
$Y_{\|r\|r}$	-115.457	kg/s
N_v	3.524	kg/s
$N_{\|v\|v}$	-0.832	kg/s
$N_{\|r\|v}$	336.827	kg/s
N_r	-122.860	kg/s
$N_{\|r\|r}$	-874.428	kg/s
N_{rrr}	0.000	kg/s
$N_{\|v\|r}$	-121.957	kg/s

Appendix B

Model Used in Optimal Control

The optimal model is presented again in Equation (B.1) [44]. The differences from the vessel model presented in Section 2.2.3, are the matrices $\boldsymbol{S}$, $\boldsymbol{M}_p$, $\boldsymbol{C}_p$ and $\boldsymbol{D}_p$, which are presented in Equation (B.2). The elements of $\boldsymbol{D}_p$ are presented in Equation (B.3), and all parameters are found in Appendix A.

$$\dot{\boldsymbol{\eta}}_p = \boldsymbol{R}(\psi_p)\,\boldsymbol{\nu}_p \tag{B.1a}$$

$$\boldsymbol{S}\boldsymbol{M}_p\dot{\boldsymbol{\nu}}_p + \boldsymbol{C}_p(\boldsymbol{\nu}_p)\,\boldsymbol{\nu}_p + \boldsymbol{D}_p(\boldsymbol{\nu}_p)\,\boldsymbol{\nu}_p = \boldsymbol{\tau}_p \tag{B.1b}$$

$$\boldsymbol{S} = diag\{2.5, 2.5, 5.0\} \tag{B.2a}$$

$$\boldsymbol{M}_p = diag\{m_{11}, m_{22}, m_{33}\} \tag{B.2b}$$

$$\boldsymbol{C}_p = \begin{bmatrix} 0 & 0 & -m_{22}v_p \\ 0 & 0 & m_{11}u_p \\ m_{22}v_p & -m_{11}u_p & 0 \end{bmatrix} \tag{B.2c}$$

$$\boldsymbol{D}_p = diag\{d_{11}(\boldsymbol{\nu}_p), d_{22}(\boldsymbol{\nu}_p), d_{33}(\boldsymbol{\nu}_p)\} \tag{B.2d}$$

$$d_{11}(\boldsymbol{\nu}_p) = -X_u - X_{|u|u}|u_p| - X_{uuu}u_p^2 \tag{B.3a}$$

$$d_{22}(\boldsymbol{\nu}_p) = -Y_v - Y_{|v|v}|v_p| - Y_{vvv}v_p^2 \tag{B.3b}$$

$$d_{33}(\boldsymbol{\nu}_p) = -N_r - N_{|r|r}|r_p| \tag{B.3c}$$

Appendix C

Weights Used in Thrust Allocation

The weights used in the thrust allocation are given in Table C.1. Some values are taken directly from Control allocation for double-ended ferries with full-scale experimental results [63], as they seem to work fine in this case too.

Table C.1: Weights used in thrust allocation.

Description	**Symbol**	**Value**
Weight on scaling down	w_s	10^7
Weight on thrust usage	w_T	0.1
Weight on thrust change	$w_{\Delta T}$	0.1
Weight on angle change	$w_{\Delta\alpha}$	10
Weight on deviation from mean angle	$w_{\delta\alpha}$	3

Appendix D

Timetable

The timetable for when a ferry should leave from each port is shown in Tables D.1 to D.3. This table is corresponding with the number 2 bus leaving from Trondheim S, going south.

Table D.1: Ferry timetable from 05:22 to 07:39.

Nyhavna	**Solsiden westward**	**Solsiden eastward**	**Trondheim S**
5:22	5:26	5:37	5:32
5:42	5:46	5:57	5:52
6:02	6:06	6:17	6:12
6:16	6:20	6:31	6:26
6:26	6:30	6:41	6:36
6:36	6:40	6:51	6:46
6:44	6:48	6:59	6:54
6:54	6:58	7:09	7:04
7:04	7:08	7:19	7:14
7:14	7:18	7:29	7:24
7:24	7:28	7:39	7:34

Table D.2: Ferry timetable from 07:34 to 14:59.

Nyhavna	**Solsiden westward**	**Solsiden eastward**	**Trondheim S**
7:34	7:38	7:49	7:44
7:44	7:48	7:59	7:54
7:54	7:58	8:09	8:04
8:04	8:08	8:19	8:14
8:14	8:18	8:29	8:24
8:24	8:28	8:39	8:34
8:34	8:38	8:49	8:44
8:46	8:50	9:01	8:56
8:56	9:00	9:11	9:06
9:06	9:10	9:21	9:16
9:16	9:20	9:31	9:26
9:26	9:30	9:41	9:36
9:36	9:40	9:51	9:46
9:46	9:50	10:01	9:56
9:56	10:00	10:11	10:06
10:06	10:10	10:21	10:16
10:16	10:20	10:31	10:26
10:26	10:30	10:41	10:36
10:36	10:40	10:51	10:46
10:46	10:50	11:01	10:56
10:56	11:00	11:11	11:06
11:06	11:10	11:21	11:16
11:16	11:20	11:31	11:26
11:26	11:30	11:41	11:36
11:36	11:40	11:51	11:46
11:46	11:50	12:01	11:56
11:56	12:00	12:11	12:06
12:06	12:10	12:21	12:16
12:16	12:20	12:31	12:26
12:26	12:30	12:41	12:36
12:36	12:40	12:51	12:46
12:46	12:50	13:01	12:56
12:56	13:00	13:11	13:06
13:06	13:10	13:21	13:16
13:16	13:20	13:31	13:26
13:26	13:30	13:41	13:36
13:36	13:40	13:51	13:46
13:46	13:50	14:01	13:56
13:56	14:00	14:11	14:06
14:06	14:10	14:21	14:16
14:14	14:18	14:29	14:24
14:24	14:28	14:39	14:34
14:34	14:38	14:49	14:44
14:44	14:48	14:59	14:54

Table D.3: Ferry timetable from 14:54 to 00:07.

Nyhavna	**Solsiden westward**	**Solsiden eastward**	**Trondheim S**
14:54	14:58	15:09	15:04
15:04	15:08	15:19	15:14
15:14	15:18	15:29	15:24
15:24	15:28	15:39	15:34
15:34	15:38	15:49	15:44
15:44	15:48	15:59	15:54
15:54	15:58	16:09	16:04
16:04	16:08	16:19	16:14
16:14	16:18	16:29	16:24
16:24	16:28	16:39	16:34
16:34	16:38	16:49	16:44
16:44	16:48	16:59	16:54
16:54	16:58	17:09	17:04
17:04	17:08	17:19	17:14
17:14	17:18	17:29	17:24
17:26	17:30	17:41	17:36
17:36	17:40	17:51	17:46
17:46	17:50	18:01	17:56
17:56	18:00	18:11	18:06
18:06	18:10	18:21	18:16
18:16	18:20	18:31	18:26
18:26	18:30	18:41	18:36
18:36	18:40	18:51	18:46
18:46	18:50	19:01	18:56
18:56	19:00	19:11	19:06
19:06	19:10	19:21	19:16
19:16	19:20	19:31	19:26
19:26	19:30	19:41	19:36
19:36	19:40	19:51	19:46
19:46	19:50	20:01	19:56
19:57	20:01	20:12	20:07
20:12	20:16	20:27	20:22
20:32	20:36	20:47	20:42
20:52	20:56	21:07	21:02
21:12	21:16	21:27	21:22
21:32	21:36	21:47	21:42
21:52	21:56	22:07	22:02
22:12	22:16	22:27	22:22
22:32	22:36	22:47	22:42
22:52	22:56	23:07	23:02
23:12	23:16	23:27	23:22
23:32	23:36	23:47	23:42
23:52	23:56	00:07	00:02

Appendix E

Domain File

The domain file for the AI-planning used in this thesis is presented in listings E.1 to E.5.

Listing E.1: The domain name, requirements and types.

```
(define (domain manageRoutes)

(:requirements :strips :durative-actions :typing :numeric-fluents)

(:types
    vessel
    plannedRoute
    trip
    port
    passengers
)
```

Listing E.2: The predicates and functions.

```
(:predicates
    (at ?v - vessel ?p - port)
    (betweenPorts ?v - vessel ?p1 - port ?p2 - port)
    (goingTo ?v - vessel ?p - port)
    (docked ?v - vessel)
    (available ?v - vessel)
    (notRecharge ?v - vessel)

    (routeStarting ?r - plannedRoute ?p - port)
    (nextStop ?r - plannedRoute ?p - port)
    (coveredPlannedRoute ?r - plannedRoute)

    (endPoint ?t - trip ?p - port)
    (differentTrips ?t1 - trip ?t2 - trip)

    (onboard ?pa - passengers ?v - vessel)
    (destination ?p - port ?pa - passengers)
    (reachedDestination ?pa - passengers)
)

(:functions
    (movetime ?port_from - port ?port_to - port)

    (extraNeededVessels ?t - trip)

    (turn ?r - plannedRoute)
    (totalTurn)

    (totalDelays)
)
```

Listing E.3: The two different move actions.

```
(:durative-action move_between_ports
    :parameters (?v - vessel ?port_from - port ?port_to - port)
    :duration (= ?duration (movetime ?port_from ?port_to))
    :condition (and
        (at start (at ?v ?port_from))
        (over all (available ?v))
    )
    :effect (and
        (at start (increase (totalDelays)
        (movetime ?port_from ?port_to))) ;line split
        (at start (not (at ?v ?port_from)))
        (at end (at ?v ?port_to))
    )
)

(:durative-action move_to_port
    :parameters (?v - vessel ?port_to - port ?port_from - port)
    :duration (= ?duration (/ (movetime ?port_from ?port_to) 2))
    :condition (and
        (at start (betweenPorts ?v ?port_from ?port_to))
        (over all (available ?v))
        (at start (goingTo ?v ?port_to))
    )
    :effect (and
        (at start (increase (totalDelays)
        (/ (movetime ?port_from ?port_to) 2))) ;line split
        (at start (not (betweenPorts ?v ?port_from ?port_to)))
        (at end (at ?v ?port_to))
        (at end (not (goingTo ?v ?port_to)))
        (at end (docked ?v))
    )
)
```

Listing E.4: The action for beginning a planned route, and the action for letting of passengers.

```
(:durative-action take_planned_route
    :parameters (?v - vessel ?r - plannedRoute ?p - port
        ?port_next - port ?port_next_next - port)
    :duration (= ?duration 1)
    :condition (and
        (at start (at ?v ?p))
        (at start (available ?v))
        (over all (routeStarting ?r ?p))
        (at start (= (totalTurn) (turn ?r)))
        (over all (nextStop ?r ?port_next))
        (over all (nextStop ?r ?port_next_next))
        (over all (notRecharge ?v))
    )
    :effect (and
        (at start (not (available ?v)))
        (at end (coveredPlannedRoute ?r))
        (at end (increase (totalTurn) 1))
        (at start (not (at ?v ?p)))
        (at end (at ?v ?port_next))
        (at end (at ?v ?port_next_next))
    )
)

(:durative-action let_off_passengers
    :parameters (?v - vessel ?pa - passengers ?p - port)
    :duration (= ?duration 15)
    :condition (and
        (at start (onboard ?pa ?v))
        (over all (at ?v ?p))
        (over all (destination ?p ?pa))
    )
    :effect (and
        (at start (not (onboard ?pa ?v)))
        (at end (reachedDestination ?pa))
    )
)
```

Listing E.5: The two actions for beginning a route for taking extra predicted needed capacity.

```
(:durative-action route_single_trip
    :parameters (?v - vessel ?t - trip ?p - port)
    :duration (= ?duration 1)
    :condition (and
        (at start (at ?v ?p))
        (at start (available ?v))
        (over all (endPoint ?t ?p))
        (at start (> (extraNeededVessels ?t) 0))
        (over all (notRecharge ?v))
    )
    :effect (and
        (at start (not (available ?v)))
        (at end (decrease (extraNeededVessels ?t) 2))
    )
)

(:durative-action route_both_trips
    :parameters (?v - vessel ?t1 - trip ?t2 - trip ?p - port)
    :duration (= ?duration 1)
    :condition (and
        (at start (at ?v ?p))
        (at start (available ?v))
        (over all (endPoint ?t1 ?p))
        (over all (differentTrips ?t1 ?t2))
        (at start (> (extraNeededVessels ?t1) 0))
        (at start (> (extraNeededVessels ?t2) 0))
        (over all (notRecharge ?v))
    )
    :effect (and
        (at start (not (available ?v)))
        (at start (decrease (extraNeededVessels ?t1) 1))
        (at start (decrease (extraNeededVessels ?t2) 1))
    )
)
```