Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Jan-Marius Vatle

Adaptive Stress Testing for Safety Validation of Maritime Autonomous Collision Avoidance Systems

Master's thesis in Computer Science
Supervisor: Ole Jakob Mengshoel
September 2022

NTNU
Norwegian University of
Science and Technology

Jan-Marius Vatle

Adaptive Stress Testing for Safety Validation of Maritime Autonomous Collision Avoidance Systems

Master's thesis in Computer Science
Supervisor: Ole Jakob Mengshoel
September 2022

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

Abstract

The research in the field of autonomous marine vessels has increased during the last decade. In context with this, a company called Zeabuz is developing an autonomous ferry system and is seeking to perform state-of-the-art safety validation of their system. Such autonomous vessels operate in complex stochastic environments. As a consequence of this complexity, failure cannot be eliminated entirely. Moreover, real-world testing can be too dangerous to perform during development, and the use of formal verification is often precluded. Simulation-based techniques resorting to statistical considerations can be the solution to work around this issue. Moreover, work done with airborne collision avoidance systems and autonomous vehicles shows successful safety validation by applying a stress testing framework called Adaptive Stress Testing (AST). The framework is based on Reinforcement Learning (RL) techniques and adaptively finds the most likely path to a failure event for the system under test in a simulated environment.

This thesis proposes an architecture called Maritime Adaptive Stress Testing (MAST), which extends the existing AST architecture to be able to test a collision avoidance system in the maritime domain, particularly the Single Path Velocity Planner (SP-VP) used by Zeabuz's passenger ferries. Two variants of the architecture are proposed, namely the Gray-Box Maritime Adaptive Stress Testing (G-MAST) and the Black-Box Maritime Adaptive Stress Testing (B-MAST), which are used for Gray-Box and Black-Box simulators, respectively. The performance of these two architectures is found to be similar in this work. However, testing with a Gray-Box simulator with G-MAST may require more domain knowledge of the system than testing with a Black-Box simulator with B-MAST.

Furthermore, MAST found several interesting failure events in the SP-VP system, which might be useful for the safety validation of the collision avoidance system. It is, however, debatable how realistic the failure events are in a real-world setting due to the irrational behavior of the other vessels, referred to as obstacle vessels in this thesis. However, the MAST architecture can be configured to test other scenarios, which can potentially find more realistic failure events.

Sammendrag

Forskningen innen maritime autonome fartøy har økt i løpet av det siste tiåret. I sammenheng med dette, utvikler et selskap kalt Zeabuz et autonomt fergesystem og ønsker å utføre “state-of-the-art” sikkerhetsvalidering av systemet deres. Slike autonome fartøy opererer i komplekse stokastiske miljøer. Som en konsekvens av denne kompleksiteten kan ikke feil elimineres helt. Dessuten kan testing i den virkelige verden være for farlig å utføre under utvikling, og bruk av formell verifisering (engelsk: formal verification) er ofte utelukket. Simuleringsbaserte teknikker som tyr til statistikk kan være løsningen for å omgå dette problemet. Arbeid utført på antikollisjonssystemer (engelsk: collision avoidance systems) av luftfartøy og autonome kjøretøy viser vellykket sikkerhetsvalidering ved å bruke et rammeverk for stresstesting kalt adaptiv stresstesting (engelsk: Adaptive Stress Testing / AST). Rammeverket er basert på teknikker med forsterkende læring (engelsk: Reinforcement Learning / RL) og finner adaptivt den mest sannsynlige veien til en feil for systemet under test i et simulert miljø.

Denne oppgaven foreslår en arkitektur kalt maritim adaptiv stresstesting (engelsk: Maritime Adaptive Stress Testing / MAST), som utvider den eksisterende AST-arkitekturen for å kunne teste et antikollisjonssystem i det maritime domenet, kalt SP-VP (engelsk: Single Path Velocity Planner / SP-VP) brukt av Zeabuz sine autonome passasjerferger. To varianter av arkitekturen er foreslått, Gray-Box maritim adaptiv stresstesting (engelsk: Gray-Box Maritime Adaptive Stress Testing / G-MAST) og Black-Box maritim adaptiv stresstesting (engelsk: Black-Box Maritime Adaptive Stress Testing / B-MAST), som brukes for Gray-Box og Black-Box-simulatorer, henholdsvis. Ytelsen til disse to arkitekturene er funnet å være lik i dette arbeidet. Derimot, kan testing med en Gray-Box-simulator med G-MAST kreve mer domenekunnskap om systemet enn å teste med en Black-Box-simulator med B-MAST.

MAST fant flere interessante feil i SP-VP-systemet, som kan være nyttige for sikkerhetsvalideringen av antikollisjonssystemet. Det er diskutabelt hvor realistiske feilene er i virkelige omgivelser på grunn av den irrasjonelle oppførselen til de andre fartøyene, som oppfører seg som hindringer i denne oppgaven. MAST-arkitekturen kan imidlertid konfigureres til å teste andre scenarier, som potensielt finner mer realistiske feil.

Acknowledgements

First and foremost, I wish to express my deepest gratitude to my supervisor, professor Ole Jakob Mengshoel, for his invaluable guidance and continuous support throughout this thesis. I am also appreciative of his patience, motivation, enthusiasm, and immense knowledge. I could not have imagined having a better supervisor during the time of writing this thesis.

Besides my supervisor, I am enormously thankful to the company Zeabuz for their collaboration, especially Øyvind Smogeli, who led the project on their side. I would also like to thank Bjørn-Olav H. Eriksen, Hanna W. Hjelmeland, and Halvor Platou for their support and guidance with the collision avoidance simulator.

My sincere gratitude also goes to Ritchie Lee for taking the time to let me present my work and answer questions about the Adaptive Stress Testing framework. Also, for taking the time to update one of his old software packages for me to use in my thesis.

I want to extend my sincere gratitude to Adrian Agogino and Rory Lipkis at the Robust Engineering technical area, based in the Intelligent Systems Division at NASA's Ames Research Center, for letting me present my work and for the tremendous support with the software package AdaStress.

Finally, I am most thankful to my family and friends for all the support, encouragement, and motivation. Special thanks to my girlfriend, Kristin S. Ødegaard, for being patient and supportive during my master's thesis. While I cannot hope to comprehensively acknowledge all who have supported me through my master's thesis, I would like to express my appreciation from the bottom of my heart to all for their support through this incredible journey.

Preface

This master's thesis is conducted as the final work of the Master of Science degree in Computer Science with specialization in Artificial Intelligence (AI) at the Norwegian University of Science and Technology (NTNU), Department of Computer Science. The research and report are done in 2022.

The thesis is a collaborative initiative involving NTNU, Zeabuz, and NASA. The autonomous technology company Zeabuz, located in Trondheim, has provided the necessary information and explanation of the autonomous passenger ferry system and provides the simulator of the maritime autonomous collision avoidance system. NASA provided support on their software package AdaStress and valuable feedback on the work.

The thesis aims to find state-of-the-art safety validation methods suitable for stress testing maritime autonomous collision avoidance to help develop and deploy autonomous marine vessels. The Adaptive Stress Testing (AST) framework has been the most prioritized part of the research for finding suitable safety validation methods. The framework AST has shown promising results in similar autonomous systems from the aircraft and automotive industries. AST is based on Reinforcement Learning (RL), and it is performed in a simulated system consisting of a system under test and an environment in which it operates.

I was inspired to pursue this topic because of my interest in AI and RL. I thought it was interesting to see how RL could be applied in real-world tasks, such as safety validation of autonomous systems.

Jan-Marius Vatle
Trondheim, September 2022

Contents

List of Tables

List of Figures

Acronyms

3DOF Three Degrees of Freedom 13, 15, 29

6DOF Six Degrees of Freedom 9–11, 13

ACAS X The next-generation Airborne Collision Avoidance System 26, 27

AI Artificial Intelligence vii, 2, 5, 20–22

AST Adaptive Stress Testing i, iii, vii, 3–7, 9, 19, 23–27, 30, 33–37, 54, 58–60, 63–65

B-MAST Black-Box Maritime Adaptive Stress Testing i, iii, 6, 7, 20, 36, 38–40, 43, 44, 49, 55–57, 60, 61

COLREG International Regulations for Preventing Collisions at Sea 16, 33, 43, 45, 54, 63, 65

DAST Differential Adaptive Stress Testing 27, 65

DRL Deep Reinforcement Learning 27

DTW Dynamic Time Warping 58

ECEF Earth-Centered Earth-Fixed 11–14

ECI Earth-Centered Inertial 11–13

G-MAST Gray-Box Maritime Adaptive Stress Testing i, iii, 6, 7, 20, 36, 37, 39, 40, 43–51, 58, 60, 61

HPR High Penalty Region 16, 17, 31–33

LPR Low Penalty Region 16, 17, 31–33, 48, 54

MAST Maritime Adaptive Stress Testing i, iii, 4, 6, 7, 20, 35, 36, 58, 60, 63

MCTS Monte Carlo Tree Search 6, 7, 21–24, 26, 27, 38, 58

MCTS-PW Monte Carlo Tree Search with Progressive Widening 38, 44, 45, 54

MCTS-SA Monte Carlo Tree Search for Seed-Action Simulators 26, 38

MDP Markov Decision Process 21, 24, 27

ML Machine Learning 6, 9, 19, 20

NASA National Aeronautics and Space Administration vii, 6, 35, 36

NED North-East-Down 11–15, 17, 29, 31

NMA Norwegian Maritime Authority 2

NTNU Norwegian University of Science and Technology vii, 2

PID Proportional Integral Derivative 32

PPO Proximal Policy Optimization 27

RL Reinforcement Learning i, iii, vii, 3, 7, 9, 20–27, 29–31, 34, 36–38, 40, 44, 60, 65

ROC Region of Collision 16–18, 30–33, 40

RTCA Radio Technical Commission for Aeronautics 27

SNAME Society of Naval Architects and Marine Engineers 10, 11

Soft-DTW Soft-Dynamic Time Warping 31, 58, 59, 64

SP-VP Single Path Velocity Planner i, iii, 2, 4–6, 9, 16–18, 24, 29–33, 35–38, 40, 43–46, 48, 51, 52, 54–56, 60, 63–65

TCAS Traffic Alert and Collision Avoidance 27

UCT Upper Confidence Bound for Trees 22, 44

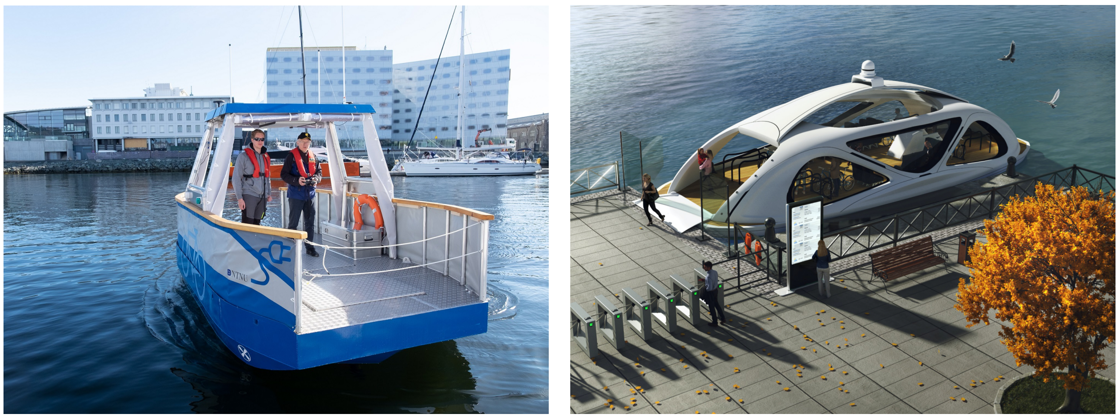
Chapter 1

Introduction

In this chapter, the background and motivation for this thesis are introduced. Further, the goal and research questions are stated and described. Then the research methodology applied will be described and justified. Followed by a summary of the structured literature review protocol and this thesis' contributions. Finally, an overview of the thesis and the remaining chapters are described.

1.1 Background and Motivation

Autonomous marine vessels have received much attention in recent years, and research in the field has increased significantly in the last decade [30, 38, 43]. The research on urban ferry transportation systems is one area in the field that is undergoing a revival. The need for transportation in cities and urban areas increases as the population grows. In many areas, ferry transportation is a suitable solution, as many cities are built by historical waterways, which often remain largely unused [30]. A good way of exploiting ferry transportation is by using autonomous marine vessels. Not only might autonomous vessels have an obvious economic impact resulting from increased efficiency and the replacement of human labor, but there is also a potential to eliminate injuries and material damage caused by collisions. Almost half of the casualties at sea from 2014 to 2019 were caused by "navigation in nature, including contact, collision, and grounding or stranding" [1]. In order to solve this issue, autonomous vessels must have a maritime autonomous collision avoidance system that provides the necessary safety to operate in their environment. However, such autonomous vessels operate in complex stochastic environments. As a consequence of this complexity, failure cannot be eliminated completely [25]. Moreover, real-world testing can be too dangerous to perform during development, and the use of formal verification is often precluded. Formal verification is the procedure of proving that a system is safe from all possible failure events [8]. In order to work around this issue, simulation-based techniques resorting to statistical considerations can be the answer [9].



(a) milliAmpere. Image credit: Kai Dragland. **(b)** The Zeabuz design. Image credit: Zeabuz.[2]

Figure 1.1: Photograph of milliAmpere and Render of the Zeabuz Passenger Ferry. Figure 1.1a shows a photograph of milliAmpere, the world's first autonomous passenger ferry prototype. In Figure 1.1b, a render of the Zeabuz ferry system design is shown.

One Norwegian company that has decided to investigate this area is Zeabuz. The company is a spinoff from the progressive research center for Autonomous Marine Operations and Systems at the Norwegian University of Science and Technology (NTNU), who are also a collaborating partner in the research done in this thesis. The company is developing a network of electric, autonomous passenger ferries enabled by Artificial Intelligence (AI). Figure 1.1 shows different versions of the passenger ferry system; where Figure 1.1a shows a photograph of milliAmpere. The milliAmpere vessel is famous for being the world's first autonomous passenger ferry prototype[1] [4]. In Figure 1.1b, the Zeabuz design of the ferry system is shown. There are also ongoing trial operations of their world's first full-scale autonomous ferry, milliAmpere 2, shown in Figure 1.2, which started in September 2022. The Zeabuz passenger ferry is the system under test in this thesis and will be referred to as the milliAmpere vessel without any version number in the rest of this thesis. The vessel is equipped with the maritime autonomous collision avoidance system Single Path Velocity Planner (SP-VP), which is the specific part of milliAmpere that is tested in this work.

The milliAmpere vessel will be used for transporting passengers in environments with unpredictable maritime traffic. Therefore, the safety requirements and the reliability of the collision avoidance system must be high [4]. This is important in order to build trust in these complex, autonomous mobility systems. In context with this, Zeabuz wants to perform state-of-the-art safety validation on their system. This is important in context with the regulations by the Norwegian Maritime Authority (NMA), which includes the

[1]Prototype promo video of milliAmpere can be found on: www.youtube.com/watch?v=z11Bk02OyXE

[2]The rendered image has been retrieved from the homepage of Zeabuz, where more renders and information about their system can be found: www.zeabuz.com

Figure 1.2: The milliAmpere 2 Passenger Ferry. This figure shows milliAmpere 2, the world's first full-scale autonomous passenger ferry. Image credit: Zeabuz.

need for conducting risk analysis for approval of the autonomous ferry. Furthermore, since milliAmpere 2 is the only operating autonomous ferry in Norway while writing this thesis, there is limited guidance on how to perform such risk analysis. There are many challenges when working with safety-critical systems operating in complex stochastic environments. Failures occur as a result of sequential interactions between the system and its environment; because of this, the analysis must be done in a system that combines both the system under test and the environment in which it operates. Scalability is also a huge issue, and brute force methods considering all possible paths are often unsuitable. Finding failure states in some safety-critical systems can also be challenging, as they are built to avoid failure, especially systems with a large search space [25]. This brings us to the next section, which defines this project's goal and research questions.

1.2 Goals and Research Questions

The goal of the research in this thesis is to find a suitable safety validation method for testing maritime autonomous collision avoidance systems. Our assumption is that Reinforcement Learning (RL) based methods are suitable for finding failure events in such systems. In particular, the framework Adaptive Stress Testing (AST), which has promising results in the safety validation of aircraft collision avoidance systems [25], will further be investigated. The goal can thereby be described as:

Goal *Investigate if AST is a suitable safety validation method for testing maritime autonomous collision avoidance.*

Further, four research questions were formulated to help reach this goal:

Research question 1 *How should we set up the scenarios for testing the collision avoidance system with AST in the maritime domain?*

Research question 2 *Is Black-Box or Gray-Box simulators best suited when testing maritime autonomous collision avoidance systems with AST?*

Research question 3 *How useful are the failure events found with AST for safety validation of the maritime autonomous collision avoidance system SP-VP?*

Research question 4 *How to interpret the failure events in the maritime autonomous collision avoidance system found with AST?*

The next section will describe the research method that this thesis uses to answer the research questions and investigate the goal presented here.

1.3 Research Method

The research method in this thesis is comprised of three phases. In the first phase, a literature review was performed. The existing literature was sorted and reviewed, and will be discussed in more detail in Section 1.4. This was done to collect the necessary knowledge about related work and existing solutions. A big part of the first phase was done during the specialization project in the fall of 2021, during the course *TDT4501 - Computer Science, Specialization Project*. In the second phase, the chosen framework AST was set up to fit the maritime domain. The necessary interfaces between the simulator and AST, and between the different programming languages were implemented. This phase led to the proposed architecture in this thesis, namely the Maritime Adaptive Stress Testing (MAST), presented in more detail in Section 3.3. In the final phase, the experiments were conducted using a simulator that consists of the system under test and the environment in which it operates. The system under test is the maritime autonomous collision avoidance system SP-VP, which milliAmpere use. Since real-world testing can be too dangerous during development, a simulation-based method is preferred. In addition, real-world testing can be insufficient due to the complexity and the number of possible failure events in the system. Using a simulator will allow the opportunity to run many scenarios and iterations to test the system in an environment close to the real-world setting. However, the simulator is an abstraction that simplifies the real-world environment. Thereby, environmental disturbances, such as weather and sea states, are absent. This should be taken into account when reading this thesis. The results from the conducted experiments are presented and discussed in Chapter 4. Finally, the results from MAST were interpreted and analyzed in the end. The results are presented in Section 4.7.

1.3.1 Non-disclosure Agreement

Since this thesis has been performed with Zeabuz as a collaborating partner, a non-disclosure agreement is signed to ensure that confidential information about their system is protected. Therefore, the simulator details can not be fully disclosed. As a consequence, the code base cannot be disclosed. However, in Section 3.1, the most fundamental details of the simulator are explained to ensure that enough knowledge of the system is presented to understand the problem. Furthermore, the confidentiality requirements will not have a huge impact on the thesis itself, but it might, however, be more difficult to reproduce the results due to the simulator not being revealed.

1.4 Structured Literature Review Protocol

The structured literature review protocol this thesis uses is described here. This protocol is used to search for and collect relevant literature necessary to accomplish the thesis goal and answer the research questions in Section 1.2. The literature in this thesis has been retrieved mainly from arXiv[3], Google Scholar[4], Oria[5], and ScienceDirect[6]. The AI-powered research tool Semantic Scholar[7] is used to track cited articles. The most important search words were: *adaptive stress testing*, *autonomous*, *interpret time series data*, *marine vessel dynamics*, *maritime collision avoidance*, *milliAmpere*, *monte carlo tree search/mcts*, *reinforcement learning*, *safety validation*, *single path velocity planner/sp-vp*. Questions related to the literature research were:

Literature research question 1 *What is the state-of-the-art of AST?*

Literature research question 2 *What is necessary domain knowledge when simulating marine vessels?*

Literature research question 3 *How is the maritime collision avoidance system SP-VP working?*

Literature research question 4 *How to interpret time series data?*

The main literature chosen in this thesis fulfills one of the inclusion criteria and all of the quality criteria presented below:

Quality criteria 1 *The literature has an explanation of their code if present.*

Quality criteria 2 *The literature has context with other studies.*

Quality criteria 3 *The literature has intuitive figures and explanations of their design and architectures.*

[3]arXiv: www.arxiv.org
[4]Google Scholar: www.scholar.google.com
[5]Oria: www.oria.no
[6]ScienceDirect: www.sciencedirect.com
[7]Semantic Scholar: www.semanticscholar.org

Inclusion criteria 1 *The study's focus on AST.*

Inclusion criteria 2 *The study's focus is on interpreting time series data.*

Inclusion criteria 3 *The study's focus on maritime collision avoidance.*

Inclusion criteria 4 *The study's focus on Monte Carlo Tree Search (MCTS).*

It is also worth mentioning that the structured literature review protocol was only used to collect the main relevant literature used in this thesis. Literature for citing relevant theories and assertions was retrieved continuously throughout the writing.

1.5 Contributions

The contributions of the work conducted in this thesis are summarized in the list below:

- The framework AST has been tested on a maritime collision avoidance system, in particular, the SP-VP method that the milliAmpere vessel use.
- A simulator interface has been developed to make AST work in the maritime collision avoidance domain. The interface is developed to support communication between the AST software package AdaStress developed in Julia, and the Zeabuz simulator, developed in Python.
- The MAST architecture for testing the maritime collision avoidance system, SP-VP, is proposed. Here, two variants of the architecture are proposed, namely the Gray-Box Maritime Adaptive Stress Testing (G-MAST) and the Black-Box Adaptive Stress Testing (B-MAST), used for Gray-Box and Black-Box simulators, respectively.
- Failure events in the collision avoidance system SP-VP have been found, which can be investigated further by the Zeabuz team.
- The AST software package AdaStress developed in Julia has been tested. It has been tested together with a simulator developed in Python. Feedback was given to the maintainers of the software package at NASA.
- The extra functionalities of the Zeabuz simulator developed in this thesis will be used by Zeabuz to continue the work on AST.

1.6 Thesis Structure

The layout of this thesis is as follows:

Chapter 2 - Background Theory: Describes the necessary domain knowledge about the marine vessel dynamics and the collision avoidance system Single Path Velocity Planner (SP-VP). Further, safety validation, Machine Learning (ML), Reinforcement

Learning (RL), Monte Carlo Tree Search (MCTS) are described. Finally, the Adaptive Stress Testing (AST) framework and its related work will be described.

Chapter 3 - Methodology: Describes the simulator used for testing in this thesis. Further, it describes the proposed architecture in this work, Maritime Adaptive Stress Test (MAST), and its two variants Black-Box Maritime Adaptive Stress Testing (B-MAST) and Gray-Box Maritime Adaptive Stress Testing G-MAST.

Chapter 4 - Results and Discussion: Presents the results from the conducted experiments in this thesis together with a discussion.

Chapter 5 - Conclusions and Future Work: Contains the conclusions and future work related to this thesis.

Chapter 2

Background Theory

This chapter presents the necessary background theory for this research. The theory includes important information on marine vessel dynamics, collision avoidance with SP-VP, and safety validation. In addition to briefly describing Black-Box and Gray-Box simulators. Next, Machine Learning (ML) and RL are described. Finally, the framework AST will be introduced, and related work on this will be mentioned.

2.1 Marine Vessel Dynamics

This section will briefly introduce some concepts to better understand the dynamics of marine vessels operating in environments with different complexity. Some simplifying assumptions and how they are used in this thesis will also be introduced. These topics are essential for simulating autonomous marine vessels.

2.1.1 Motion at Sea in Six Degrees of Freedom

Foremost, an introduction to marine vessels operating in more complex environments will be introduced to provide a more complete understanding of the domain of this thesis. In context with this, the dynamics of a marine vessel are often described using Six Degrees of Freedom (6DOF), which are the set of independent displacements and rotations that define the displaced position and orientation of the vessel. The 6DOF consists of six components, where three are translational and three are rotational. This is illustrated in Figure 2.1. The translational components are *surge*, *sway*, and *heave*, which is longitudinal motion, sideways motion and vertical motion respectively. The rotational components are *roll*, *pitch*, and *yaw*, which is the rotation about the longitudinal axis, the transverse axis and the vertical axis, respectively. Equation 2.1 shows the mathematical notation for the positions and orientations of a marine vessel in 6DOF in vector form, represented by $\boldsymbol{\eta}$:

$$\boldsymbol{\eta} = [x, y, z, \phi, \theta, \psi]^T, \tag{2.1}$$

where the first three components of the vector, (x, y, z), represents the position in the surge, sway, and heave, respectively. The last three components, (ϕ, θ, ψ), represent the orientation in roll, pitch, and yaw, respectively. Marine vessels operating in 6DOF also have six velocity components, which can be divided into linear and angular velocities, represented by $\boldsymbol{\nu}$. These are illustrated in Figure 2.1. Equation 2.2 shows their mathematical notation:

$$\boldsymbol{\nu} = [u, v, w, p, q, r]^T, \tag{2.2}$$

where the first three components (u, v, w) are the linear velocities in the surge, sway and heave, respectively. The last components (p, q, r) are the angular velocities in roll, pitch, and yaw, respectively [13].

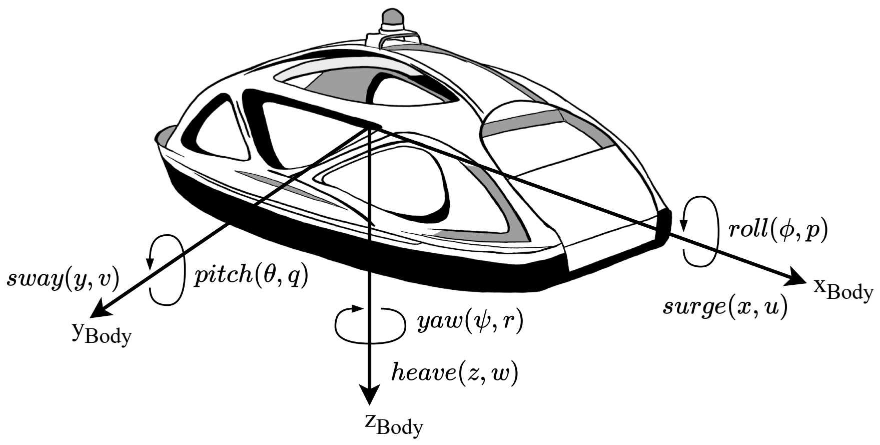


Figure 2.1: Six Degrees of Freedom (6DOF). The x, y and z are the displaced positions in surge, sway and heave, respectively. The ϕ, θ and ψ are the displaced orientations in roll, pitch and yaw, respectively. The u, v and w are the linear velocities in surge, sway and heave, respectively. The p, q and r are the angular velocities in roll, pitch, and yaw, respectively. The figure is inspired by [13].

The mathematical notation for marine vessels in this 6DOF is The Society of Naval Architects and Marine Engineers (SNAME) [36]. An overview of this notation can be found in Table 2.1. This notation will be used throughout this thesis, but with the adjustment of replacing x and y, with N and E, respectively, for readability. More about this in Section 2.1.3.

Table 2.1: SNAME notation for marine vessels. The table is based on [14].

Description	**Positions and Euler angles**	**Linear and angular velocities**
Motions in the x direction (surge)	x	u
Motions in the y direction (sway)	y	v
Motions in the z direction (heave)	z	w
Rotation about the x axis (roll)	ϕ	p
Rotation about the y axis (pitch)	θ	q
Rotation about the z axis (yaw)	ψ	r

2.1.2 Reference Frames

More complex systems also use a variety of reference frames, most of these are not used in the work of this thesis, but are introduced to provide a better understanding of the domain. Reference frames and the coordinate systems that are associated with them are useful in the analysis of marine vessels in 6DOF. The types of reference frames can be divided into two groups, the Earth-centered reference frames, and the geographic reference frames. Of this, two types of Earth-centered reference frames are used, the Earth-Centered Inertial (ECI) frame and the Earth-Centered Earth-Fixed (ECEF) frame. The ECI-frame is an inertial frame with the origin located at the Earth's center of mass. The x-axis is permanently fixed in a direction relative to the celestial sphere, which does not rotate as Earth does. The x-y plane coincides with the equatorial plane of Earth, with the z-axis extending through the North Pole [31]. In this non-accelerating reference frame, Newton's laws of motion apply. The ECEF-frame is fixed relative to the Earth, not fixed to the celestial sphere, and is moving relative to the inertial frame with an angular rate ω_{ECEF} with rotation about the z-axis [13]. Both of the Earth-centered reference frames are depicted in Figure 2.2, where the figure also shows how they behave relative to each other.

The geographic reference frames are the North-East-Down (NED) frame and body-fixed frame. The NED-frame has its origin defined relative to the Earth's reference ellipsoid, and is defined as the tangent plane on the surface of the Earth moving with the vessel. The x-axis points towards true *North*, the y-axis points towards *East*, and the z-axis points *downwards* normal to the Earth's surface. The NED-frame is located relative to the ECEF-frame using *longitude* and *latitude*. The body-fixed reference frame has the coordinate system fixed to the vessel, where the origin coincides with a point midship in the water line. For marine vessels, the body axes are chosen to coincide with the *principal axes of inertia*, where the longitudinal x-axis is directed from aft to fore, the transversal y-axis is directed to starboard, and the normal z-axis is directed from top to bottom. Both the NED-frame and the body-fixed-frame are depicted in Figure 2.2. The position and orientation ($\boldsymbol{\eta}$) of the vessel are described relative to the inertial reference frame, which can be the ECEF-frame or the NED-frame, depending on the problem. The

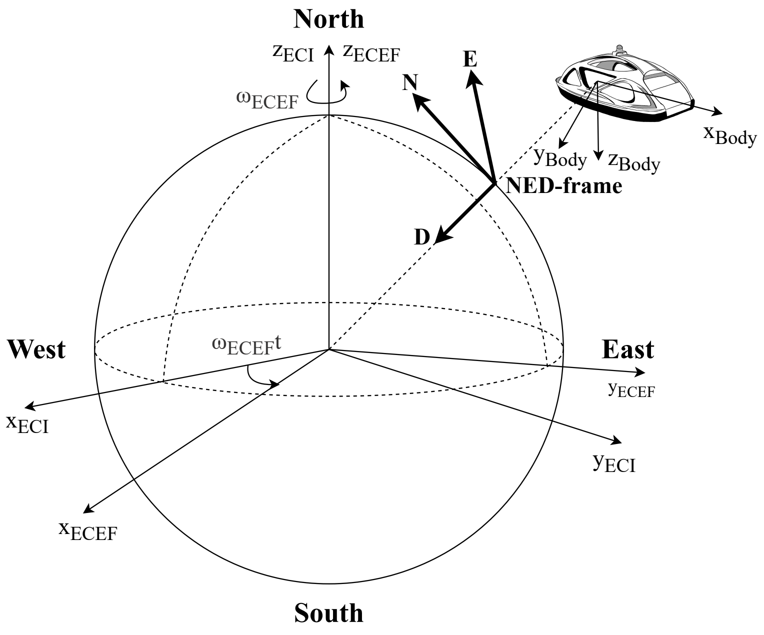


Figure 2.2: Reference Frames. This figure depicts the reference frames Earth-Centered Intertial (ECI), Earth-Centered Earth-Fixed (ECEF), North-East-Down (NED) and the body-fixed. The ECI-frame is an inertial frame that is fixed to the celestial sphere, while the ECEF-frame is fixed relative to the Earth. The ECEF-frame moves relative to the ECI-frame with an angular rate ω_{ECEF} with rotation about the z-axis. The NED-frame has its origin defined to the Earth's reference ellipsoid and it is defined as the tangent plane on the surface of the Earth moving with the marine vessel. The body-fixed reference frame has the coordinate system fixed to the marine vessel. The figure is based on [13].

linear and angular velocities ($\boldsymbol{\nu}$) are expressed in the body-fixed-frame. There are also some additional body-fixed coordinate systems and reference frames not presented here, for example the *seakeeping reference frame* and the additional body-fixed coordinate system when the system uses *flow axes* [13]. In this thesis, there are only two relevant reference frames from the ones mentioned above: the NED-frame and the body-fixed reference frame, the reason why is presented in the next section.

2.1.3 Flat Earth Navigation in Three Degrees of Freedom

For marine vessels that do not have actuation in all 6DOF and operate under certain conditions, it is possible to simplify the simulation and use reduced-order models. This is achieved by decoupling unnecessary motion components of the marine vessels and considering a different reference frame than the ECI-frame to be inertial. This can be achieved by a few simplifying assumptions presented in Fossen's work [13]. The simplifying assumptions are presented in Table 2.2, where (1) marine vessels operating at relatively low speeds can neglect the Earth's rotation, and thereby the ECEF-frame can be considered to be inertial; (2) for marine vessels operating in a local area with approximately constant longitude and latitude, an Earth-fixed tangent plane on the surface of the Earth is used for navigation. Due to this, the NED-frame can be assumed to be inertial. Finally, (3) for marine vessels that operate in the calm sea where one can assume that the displaced orientations in roll ϕ and pitch θ are to be arbitrarily small. Thereby, the components corresponding to heave, roll, and pitch can be neglected. The rest of this section will show how these assumptions and simplifications are used in this thesis.

Table 2.2: Simplifying assumptions for marine vessels [13].

No.	**Assumption**	**Simplification**
1	The marine vessel operates at relatively low speed.	The ECEF-frame can be considered to be inertial.
2	The marine vessel operates in a local area.	The NED-frame can be considered to be inertial.
3	The marine vessel operates in calm sea.	Components corresponding to heave, roll, and pitch can be neglected.

In Table 2.3, an overview of some reduced order models and their usage can be found. The interesting ones for this thesis are the horizontal plane models in Three Degrees of Freedom (3DOF). These can be achieved by using the simplifying assumption (3) from Table 2.2. This simplification allow us to neglect the components corresponding to heave, roll, and pitch, which makes it possible to rewrite the equations from Section 2.1.1 into Equation 2.3:

$$\boldsymbol{\eta} = [N, E, \psi]^T, \tag{2.3}$$

where we now have removed z, ϕ and θ. Also, note that x and y are replaced by N (North) and E (East) for readability. In Equation 2.4:

$$\boldsymbol{\nu} = [u, v, r]^T, \tag{2.4}$$

we have removed w, p and q.

Table 2.3: Reduced-order models and their usage [13].

DOFs	**Usage**	**State Components**
1DOF	Forward speed controllers	(*surge*)
	Heading autopilots	(*yaw*)
	Roll damping systems	(*roll*)
3DOF	Horizontal plane models	(*surge*, *sway*, *yaw*)
	Longitudinal models	(*surge*, *roll*, *yaw*)
	Lateral models	(*sway*, *roll*, *yaw*)
4DOF	Formed by adding the roll equation to the 3DOF horizontal plane model	(*surge*, *sway*, *roll*, *yaw*)
6DOF	Fully coupled equations of motion used for simulation and prediction of coupled vehicle motions	(*surge*, *sway*, *heave*, *roll*, *pitch*, *yaw*)

Using the simplifying assumption (2) from Table 2.2, the marine vessel operates in a local area, it is possible to achieve what Fossen refers to as flat earth navigation [13]. The NED-frame is now considered to be inertial. However, as the NED-frame is now considered to be inertial, we thereby do not have to consider assumption (1) from Table 2.2, that the ECEF-frame is considered to be inertial. In Figure 2.3, an illustration of how the marine vessel's positions, orientations, and velocities are represented in a NED-frame. Since the z-axis in the NED-frame points downwards normal to the Earth's surface, it looks like the y-axis shows the North N position, and the x-axis the East E position, but this is not the case. The North N is the true x-axis and East E is the true y-axis. The yaw angle ψ represents the heading of the vessel relative to the North. The vectors $\vec{u}$ and $\vec{v}$ represent surge velocity and sway velocity, respectively. The angular velocity in yaw is represented by $\vec{r}$.

For horizontal plane models, the kinematic equations can be expressed as Equation 2.5, when assuming calm sea and no weather such as wind:

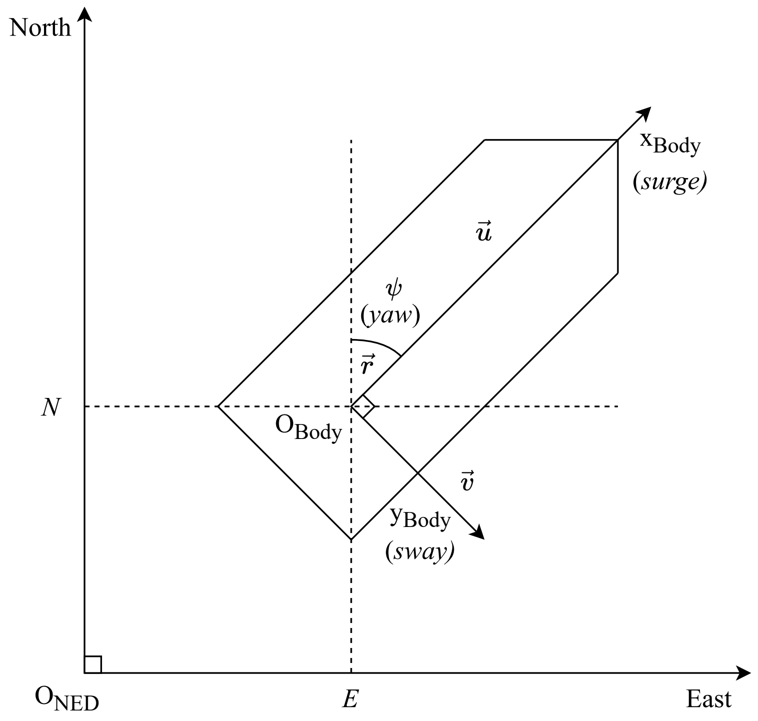


Figure 2.3: Flat Earth Navigation in 3DOF. The figure depicts flat earth navigation in 3DOF. This simplified version's only necessary reference frame is the NED-frame. N and E represent the marine vessel's displaced positions in the reference frame, and ψ represents the displaced orientation. $\vec{u}$ and $\vec{v}$ represents the linear velocities in *surge* and *sway*, respectively. $\vec{r}$ represents the angular velocity $\vec{r}$ in *yaw*. Notice that y_{Body} is in the starboard direction. This figure is based on [28].

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \boldsymbol{R}(\psi)\boldsymbol{\nu}, \\ \boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} &= \boldsymbol{\tau}, \end{aligned} \tag{2.5}$$

where $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{C}(\boldsymbol{\nu})$ is the centripetal and Coriolis matrix, and $\boldsymbol{D}(\boldsymbol{\nu})$ is the damping matrix. Since the only rotation is about the z-axis or yaw, we get $\boldsymbol{R}(\psi)$ in Equation 2.6 expressed as:

$$\boldsymbol{R}(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.6}$$

The marine vessels in the simulator provided by Zeabuz are based on these equations for their dynamics.

2.2 Collision Avoidance with Single Path Velocity Planner

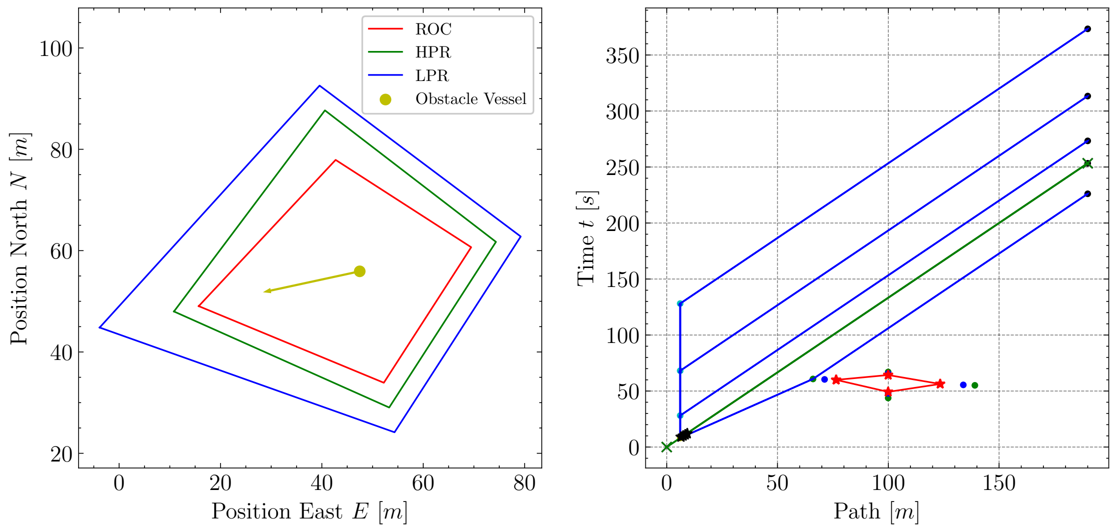
SP-VP[1] is a motion planning system that aims to ensure collision-free maneuvers from the start to its goal waypoint. The system behaves as a maritime autonomous collision avoidance system and was first introduced by Thyri et al. [40]. This section is also based on Thyri's master's thesis [41] and the continued work on the method by Berget in his master's thesis [3]. The method is developed for the specific case of autonomous passenger ferries operating in confined waters with high traffic, and was built in context with the milliAmpere project. Confined waters, in the context of navigation, is referred to as an area of the sea with a relatively small width of the waterway relative to the marine vessel's ability to maneuver. The collision avoidance system is provided with a predefined waypoint mission, which must be collision-free with respect to static obstacles, such as islets and breakwaters. The collision avoidance problem is a velocity planning problem, which means that it only plans a velocity profile and does not change the marine vessel's route. In this context, the route is defined as a straight line from the start to the goal of the waypoint mission. The SP-VP method tracks dynamic obstacles, which are moving objects such as other vessels. These obstacles are represented as shown in Figure 2.4a. More details on this representation is found in Section 2.2.1. These obstacles are transformed into a path-time space and are then constructed as a conditioned visibility graph and traversed with Dijkstra's algorithm [11], to find a collision-free velocity profile. This path-time space is illustrated in Figure 2.4b, to give a better understanding of how SP-VP work. However, this is outside the thesis's scope and will not be further discussed. A similar method of path-velocity decomposition has also been proposed by Kant et al. [17].

The SP-VP method does not comply well with the International Regulations for Preventing Collision at Sea (COLREG) when maneuvering in sight of other vessels, according to Brekke et al. [4]. The reason is that the rules in COLREG are weighted more on course change maneuvers over speed change maneuvers to avoid a collision. Furthermore, the fixed predefined waypoint mission makes compliance with the COLREG rules difficult to adhere to.

2.2.1 Obstacle Representation

In SP-VP, a simplified obstacle representation is used for robustness and ease of computation. Three regions surround the point considered to be the obstacle represented in a northeast frame. The regions are shown in Figure 2.4a, where the Region of Collision (ROC) is shown in red, the High Penalty Region (HPR) is shown in green and the Low Penalty Region (LPR) is shown in blue. In yellow, we see the point that is considered the obstacle together with its heading. All of the regions are diamond-shaped convex polygons. The ROC is designed to include both the obstacle vessel's dimensions and the dimensions of the vessel using the system, which in our case is milliAmpere. By do-

[1]A video showing the SP-VP system as part of milliAmpere, retrieved from Brekke et al. [4]: www.youtube.com/watch?v=Ry3-yxVaDuE



(a) Obstacle Representation **(b)** SP-VP Path-time Space

Figure 2.4: SP-VP Obstacle Representation and Path-time Space. In Figure 2.4a the obstacle representation of SP-VP is shown. In red is the Region of Collision (ROC), in green is the High Penalty Region (HPR), and in blue is the Low Penalty Region (LPR). We can see the obstacle as a point together with its heading in yellow. Figure 2.4b shows an obstacle represented in a path-time graph together with its visibility graph.

ing this, the milliAmpere can be considered as a point when constructing the visibility graph. The ROC can be calculated by the method described by Lozano-Perez in [27]. The regions for an obstacle can be calculated by Equation 2.7:

$$\begin{aligned}
c_f &= [N_o + l_f \cos(\psi_o), E_o + l_f \sin(\psi_o)], \\
c_a &= [N_o + l_a \cos(\psi_o + \pi), E_o + l_a \sin(\psi_o + \pi)], \\
c_s &= [N_o + l_s \cos(\psi_o + \pi/2), E_o + l_s \sin(\psi_o + \pi/2)], \\
c_p &= [N_o + l_p \cos(\psi_o + \pi/2), E_o + l_p \sin(\psi_o + \pi/2)],
\end{aligned} \tag{2.7}$$

where N_o and E_o denote the North and East position of the obstacle, respectively. The ψ_o denotes the heading of the obstacle. Moreover, c_f, c_a, c_s and c_p are the coordinates of the vertices in the NED-frame, for fore, aft, starboard and port of the obstacle vessel respectively. The lengths l_f, l_a, l_s and l_p denote the distance from the obstacle point to the vertices, for fore, aft, starboard and port respectively. The lengths should be assigned with consideration of both the dimensions of the obstacle and the vessel using the system, the lengths should also include some safety factors and perimeter size. Equation 2.8 shows how the lengths are calculated:

$$
\begin{aligned}
l_f &= (l_{f_o} + l_{f_s} + l_{f_{\text{region}}}),\\
l_a &= (l_{a_o} + l_{a_s} + l_{a_{\text{region}}}),\\
l_s &= (l_{s_o} + l_{s_s} + l_{s_{\text{region}}}),\\
l_p &= (l_{p_o} + l_{p_s} + l_{p_{\text{region}}}),
\end{aligned}
\tag{2.8}
$$

where l_{f_o}, l_{a_o}, l_{s_o} and l_{p_o} are the lengths from center of the obstacle vessel to fore, aft, starboard and port, respectively. And l_{f_s}, l_{a_s}, l_{s_s} and l_{p_s} are the lengths from center of the vessel using the system to fore, aft, starboard and port, respectively. Lastly, $l_{f_{\text{region}}}$, $l_{f_{\text{region}}}$, $l_{f_{\text{region}}}$ and $l_{f_{\text{region}}}$ are the chosen safety factors and perimeter sizes, for fore, aft, starboard and port respectively. All of the lengths discussed above must be set for all of the regions with the condition $l_{LPR} > l_{HPR} > l_{ROC}$ satisfied [40]. The equations discussed in this section are used to implement the true ROC of an obstacle vessel in addition to the SP-VP obstacle representation in this thesis. The true ROC differs from the SP-VP ROC, because noise to the SP-VP-tracking system makes the position estimated different from the true ROC. The SP-VP system updates with a given rate, which also makes it differ from the true ROC of the obstacle vessel. More details about this in Section 3.1.

2.3 Safety Validation

The best way to describe safety validation is to describe the meaning of each word; *safety* and *validation*. This will be done here in the context of a software program:

- **Safety** can be described as a *safety property*, which specifies that "bad things" do not happen during execution of a program. In contrast to a safety property, we have a *liveness property*, which specifies that "good things" will eventually happen. This distinction is useful, because proving that a program satisfies a safety property, can be done by using a counterexample that shows that a bad thing did happen. On the other hand, proving that a program satisfies a liveness property requires formal argumentation. This knowledge is useful when deciding how to prove these two types of properties [2]. The definition of a "bad"- or "good-event" is domain specific.

- **Validation** is the process of showing that the product accomplishes the intended purpose in the intended environment. Verification is the process of proving that the product meets every requirement. The selection of the verification or validation method is based on whether we want to prove the requirements of the system or show that the system operates as intended [18]. In our case, the term validation is chosen to emphasize that we are testing the system prototypes in a simulated operational environment.

In other words, safety validation is described as the process of ensuring the correct and safe operation of the system operating in an environment. The process is based on examination and testing in order to find if the safety validation goals are sufficient and have been achieved in its operational domain [9].

2.3.1 Challenges with Safety Validation of Autonomous Systems

An autonomous system involves software and physical systems interacting over time. Due to this, there are several challenges with the safety validation of these systems. In work done by Corso et al. [9], some of the challenges were introduced. These can be found in the list below:

- **Subtle and emergent failures can go undetected** due to autonomous systems often containing complex components where safety properties are hard to understand. Especially ML components often have high complexity. Due to this, we often get a tradeoff related to the complexity of our model and the model interpretability [45].
- **Autonomous systems are hard to model** because of their interaction with complex and stochastic environments.
- **Safety validation must be performed over the combined system, consisting of the system under test and the environment in which it operates**, this is due to safety properties being defined over both the system under test and its environment.
- **Computational efficiency** because of the search space is combinatorially large due to the sequential interactions between the system and the environment.
- **Failure events can be rare** because safety validation is applied late in development when the autonomous safety systems are more mature.

Because of these challenges, traditional methods through safety processes, engineering analysis, and conventional testing are insufficient, and more advanced techniques are necessary to perform safety validation of these systems [9].

2.4 Black-Box and Gray-Box Simulators

The term Black-Box simulator is used when no knowledge of the system's internals is known or if it is too complex to reason about explicitly. Due to their complexity, autonomous systems are often implemented as a Black-Box simulator. This contrasts with a White-Box simulator where it is possible to describe it analytically or specify it in a formal modeling language. There is also something in between, referred to as Gray-Box simulators [9]. In this thesis, the simulator has been implemented as a Black-Box and a Gray-Box. This was possible due to having access to the simulator code base. In context with AST, these types of simulators will be used to try out the AST architectures for Black-Box and Gray-Box simulators, which will further be explained in 2.8. These architectures will be extended to the proposed architectures in this thesis, which are G-MAST and B-MAST, more details about these in Section 3.3.

2.5 Machine Learning

ML is a subfield of AI, which was defined by Tom M. Mitchell as the study of algorithms that allow computer programs to automatically improve through experience [29]. These algorithms are often categorized into four categories based on the amount of human supervision in the learning process [32]:

- **Supervised Learning:** Here, the training data consists of input-output pairs, normally called features and targets, respectively. The agent then learns a function that maps from input to output.
- **Unsupervised Learning:** In the methods in Unsupervised Learning, the agent learns patterns in the input without giving any output to train on, only the features are available.
- **Semi-supervised Learning:** In this case, a combination of both labeled and unlabeled data is used during training.
- **Reinforcement Learning:** The agent is now learning by doing actions in an environment and learning from the feedback in the form of reward or punishment, called reinforcements, to tell the agent if the action was good or not. RL should not be confused with Unsupervised Learning because of the absence of labeled data. We are not trying to learn patterns in the input but rather maximizing a reward signal [37].

A more detailed introduction to the RL category will be given in Section 2.6. There will also be introduced an Unsupervised Learning method in Section 4.7 used to interpret the data from MAST in this thesis.

2.6 Reinforcement Learning

In RL, the basic idea is to let an agent interact with an environment over time to capture important features of the problem to achieve a goal [37]. This is done by letting the agent learn by receiving rewards or punishments, called reinforcements, as an outcome for choosing an action in a given state in an environment [32]. This approach to learning by interacting with an environment to achieve a goal contrasts with other categories in ML. One more unique challenge that arises in RL and not in other categories of ML is the trade-off between exploration and exploitation. Exploration is the act of trying new actions in the action space to discover more optimal solutions to the problem. On the other hand, exploitation is the act of exploiting actions already found that leads to the optimal solution. Both exploration and exploitation are important, and neither can be precluded [37]. The rest of this section will introduce the Markov Decision Processes (MDP).

2.6.1 Markov Decision Processes

MDP is a way of formalizing sequential decision making [44]. The formalization is the basis for problems solved with RL. An illustration of the formulation can be found in Figure 2.5. In this formulation, we have a decision maker called an *agent*. The agent interacts with an *environment* sequentially over time. In each discrete time step t, the agent will get some representation of the environment state $s_t \in S$, where S is the state space, and then selects an action $a_t \in A$, to take, where A is the action space. The action is selected randomly, or from its policy, this is called exploration and exploitation, respectively. Next, the environment transition into the next state $s_{t+1} \in S$ based on the action done by the agent. This next state is returned to the agent together with a reward r_{t+1} calculated by a reward function $R(s_t, a_t)$, where the reward is given for taking action a_t in state s_t. The reward is used to tell the agent if it did good or badly and make it possible for the agent to improve over time. This process creates trajectories of states, actions, and rewards. The goal is to maximize the total amount of rewards that it receives from taking actions in the environment [37].

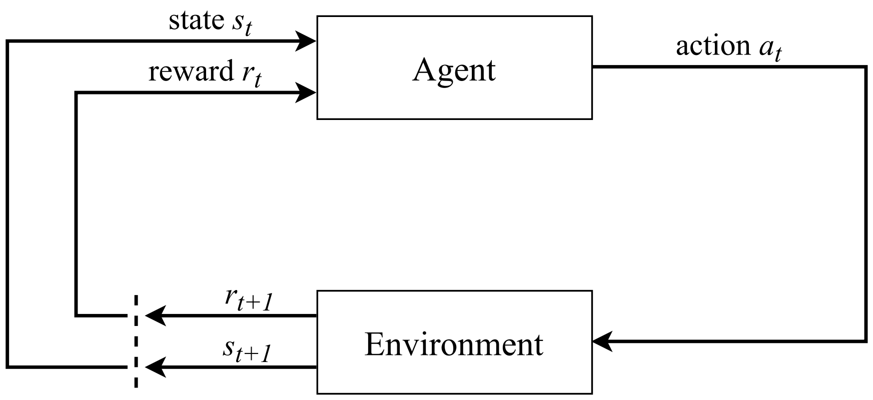


Figure 2.5: Markov Decision Process. An agent interacts with an environment sequentially over time. In each discrete time step t, the agent receives a state $s_t \in S$ and a reward r_t from the environment calculated by a reward function $R(a, s)$. In the given state s_t, it selects an action a_t to take. The environment transitions into the next state $s_{t+1} \in S$ and returns this state together with a reward r_{t+1}. This process continues until it reaches a terminal state and repeatedly happens until reaching satisfied learning goals. This figure is based on [37].

2.7 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is used in many AI search problems. For instance, it is used in the core algorithm of the AlphaGo system [34]. A lot of research has been done in MCTS algorithms, which shows that it is suitable for many different domains and is a field in AI that might have a bright future. There are also a lot of different variations of the algorithm [5]. When working with MCTS, we have to keep track of at least two different policies: the tree policy and the default policy. The tree policy is used for traversing the tree, the default policy is used during rollouts. Figure 2.6 shows an illustration of one MCTS iteration, where each node in the tree represents a state s in the search space S. The first phase is the selection phase: The algorithm starts in the initial state represented by the root node. Then child nodes are selected following the tree policy until reaching a terminal state or a node that is not fully expanded. The next phase is the expansion phase: Expands the tree by selecting an unvisited action that transitions into the next state, represented by a newly added leaf node added to the tree. The rollout phase: A simulation is run from the newly expanded leaf node until reaching a terminal state or until the simulation time is finished, this depends on the problem. The rollout phase uses the default policy, which is often a random policy. In the terminal state, a reward value is produced, this value is then backpropagated in the backpropagation phase to update the previously selected nodes in the current iteration [5].

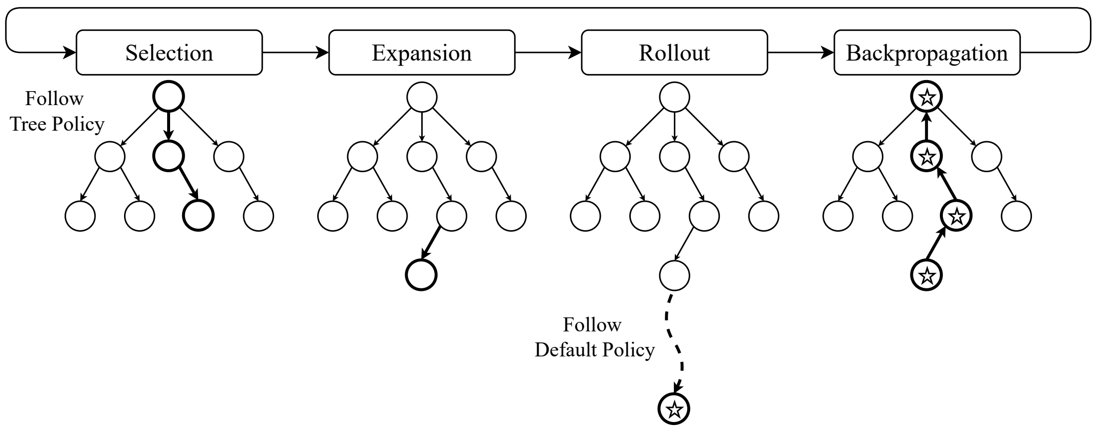


Figure 2.6: Monte Carlo Tree Search Iteration. In this figure, one MCTS iteration is illustrated. This figure is based on [7].

MCTS use the Upper Confidence Bound for Trees (UCT), to handle the exploration vs explotation challenge that arises in RL, in Equation 2.9 the UCT is expressed:

$$UCT = Q(s,a) + c\sqrt{\frac{log(N(s))}{N(s,a)}}, \tag{2.9}$$

where $N(s)$ is number of times the node s has been visited and $N(s,a)$ is the number of traversals of edge (s,a) and c is an exploration balance coefficient. The $Q(s,a)$ is the Q-value. The selection phase select the actions that maximizes the UCT, the UCT makes the selection phase able to explore more instead of only selecting actions based on previous results [19, 25].

In this thesis, a variant of MCTS that uses progressive widening is used. This variant makes MCTS perform better in large and continuous action spaces. More about this variant in Section 3.4.

It is also possible to do Deep Reinforcement Learning together with MCTS algorithms. Silver et al. achieved good results when they applied deep neural networks as critics and actors to evaluate states and select actions when they trained AlphaGo Zero [35]. This is done by applying the critic for evaluation of leaf nodes, instead of/or in combination with rollout simulations. The actor, on the other hand, is used as the default policy during rollouts and is trained by the real states as features and the action probabilities made by the MCTS algorithm as targets. However, this approach is not used in this thesis.

2.8 Adaptive Stress Testing

AST is a framework that is used to search for the *most likely path* to a failure event in a simulation. The problem is formulated as a sequential decision problem, and RL is used to optimize it [21, 22, 24, 25]. Figure 2.7 depicts the general framework. The main components of the framework are the RL-agent $\mathcal{A}$ and the simulator $\mathcal{S}$. The simulator $\mathcal{S}$ consists of the system under test $\mathcal{M}$, and the environment $\mathcal{E}$ in which it operates. The RL-agent $\mathcal{A}$ acts as an adversary to the system under test $\mathcal{M}$ by choosing disturbances x so that the environment $\mathcal{E}$ is as challenging to the system under test $\mathcal{M}$ as possible. This is done through repeated interactions with the simulator $\mathcal{S}$. The agent $\mathcal{A}$, then learns to choose disturbances x that maximize the reward r it receives. By choosing a reward function that rewards failure events and higher likelihood transitions, the agent learns to optimize for the most likely failure path [25].

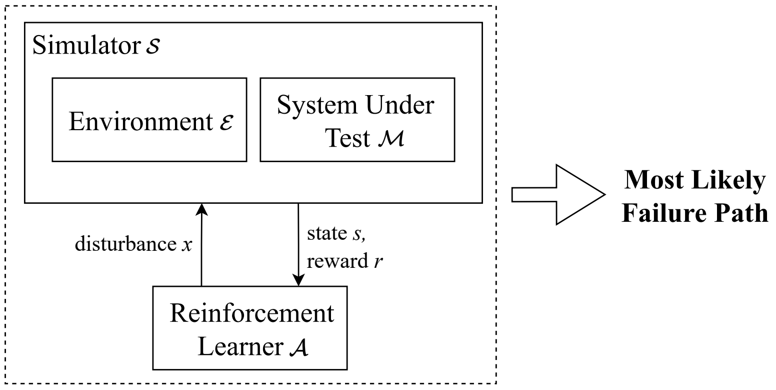

Simulator $\mathcal{S}$
Environment $\mathcal{E}$
System Under Test $\mathcal{M}$
disturbance x
state s, reward r
Reinforcement Learner $\mathcal{A}$
Most Likely Failure Path

Figure 2.7: Adaptive Stress Testing. This figure illustrates the basic idea of the AST-framework. The framework uses a simulator $\mathcal{S}$ consisting of the system under test $\mathcal{M}$ and the environment $\mathcal{E}$ in which it operates. An RL-agent $\mathcal{A}$ chooses a disturbance x and sends it to the simulator's environment to act like an adversarial to the system under test. The simulator returns the simulator state s and a reward r. The goal is to find the most likely path to a failure event. The figure is based on [25].

When talking about the most likely path, it is important to know the difference between likelihood and probability. Likelihood can be described as finding the best distribution of the data given a particular value of a situation in the data. Probability can be described as finding the chance of something given a sample distribution of the data [12]. Lee et al. [25], define the most likely failure path in the context of AST, to be the path with the highest likelihood subject to the constraint that a final state $s_{t_{\text{end}}}$ is an event, this can be expressed as:

$$\begin{aligned} &\max_{x_0,\dots,x_{t_{\text{end}}}} \prod_{t=0}^{t_{\text{end}}-1} p(x_t|s_t), \\ &\text{subject to } s_{t_{\text{end}}} \in E, \end{aligned} \tag{2.10}$$

where the $p(x_t|s_t)$ is the transition likelihood and $E \subset S$ is a subset of states in the state space S that can be defined as a failure event e.

The disturbances x are stochastic variables in the simulation sampled from a probability distribution of choice, the Gaussian distribution, for instance. These variables are influencing the environment $\mathcal{E}$ in which the system under test $\mathcal{M}$ operates. In this thesis, the disturbances are the surge speed u and heading ψ of the vessels that act like other actors in the environment, together with the system under test $\mathcal{M}$, which is the milliAmpere vessel in our case. Noise to the SP-VP system is also used as a disturbance.

In Section 2.4, Black-Box and Gray-Box simulators was introduced. In the context with AST, there are different types of architectures to fit these two types of simulators. These will be introduced in the next subsection.

2.8.1 Adaptive Stress Testing for Gray-Box Simulators

If the system gives the agent access to the complete state at each point in time, we say that the system is fully observable [32]. Because of this, we can say the Gray-Box simulator is a fully observable system. When a system is fully observable, it is possible to formulate the problem as an MDP, which was introduced in Section 2.6.1. If the system is fully observable, and the problem is formulated as an MDP, standard RL algorithms, such as MCTS and Q-learning, can be used [37]. Figure 2.8 shows the AST architecture for Gray-Box simulators. In this architecture, the RL-agent chooses a disturbance x to influence the environment $\mathcal{E}$ and acts like an adversary to the system under test $\mathcal{M}$. The simulator $\mathcal{S}$, which is Gray-Box in this case, transitions into its next state s' and returns it to the RL-agent $\mathcal{A}$. The simulator $\mathcal{S}$ also returns the event e and miss distance d to the reward function. The reward function needs this information together with the current state s and the disturbance x that was chosen in that state to be able to calculate the reward. The reward is sent to the RL-agent to give feedback if it did good or bad.

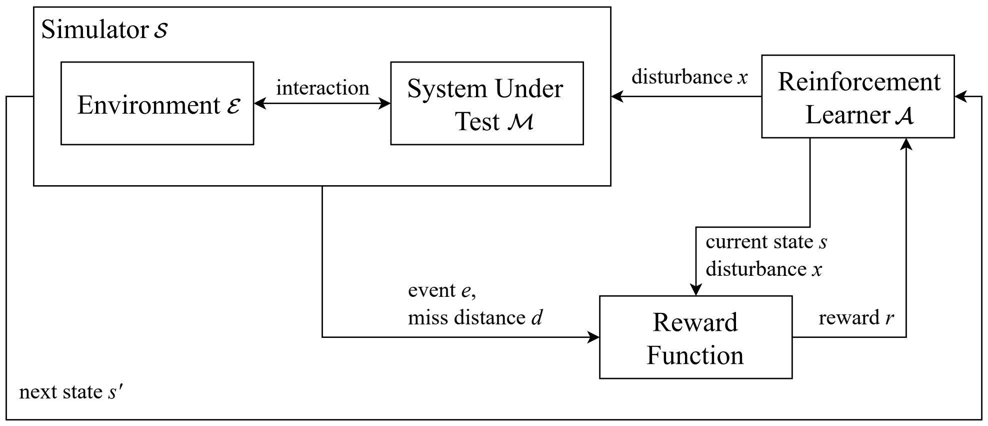


Figure 2.8: Adaptive Stress Testing for Gray-Box Simulators. This figure shows the AST architecture for Gray-Box simulators. The RL-agent $\mathcal{A}$ chooses a disturbance x and sends it to both the Gray-Box simulator $\mathcal{S}$ and the reward function. The reward function also receives the current state s, this is necessary for the reward function to be able to calculate the transition likelihood. The simulator $\mathcal{S}$ returns the next state s' to the RL-agent $\mathcal{A}$ and a boolean indicating if the next state s' of the simulator is a failure event e or not. The miss distance d from a possible failure event is also returned to the reward function. The reward is calculated and sent as feedback to the RL-agent $\mathcal{A}$. This figure is based on [25, 26].

2.8.2 Adaptive Stress Testing for Black-Box Simulators

If parts of the systems' state are missing, the system is known as partially observable [32]. This is the case for many types of simulators because they do not allow access to all or any state information, which is the case for Black-Box simulators. In other words, the simulator appears non-Markovian to external processes. AST for Black-Box simulators differs from the Gray-Box approach because the simulator states are now being maintained internally. Moreover, the RL-agent $\mathcal{A}$ now chooses a pseudorandom seed to initialize the pseudorandom number generator of the Black-Box simulator $\bar{\mathcal{S}}$ instead of sampling the disturbances x for the environment $\mathcal{E}$, which is the case for the Gray-Box approach of AST. The random processes of the simulator $\bar{\mathcal{S}}$ are assumed to be derived from the seed $\bar{x}$, this makes the simulator deterministic given the seed. Therefore, the seed $\bar{x}$ is deterministically connected to a particular sample of disturbances internally in the simulator $\bar{\mathcal{S}}$. Since the RL-agent no longer chooses the disturbances x, these must be handled internally in the Black-Box simulator $\bar{\mathcal{S}}$. The transition likelihood ρ must, therefore, also be handled internally in the simulator. Since the simulator deterministically transitions given the seed, it allows previously visited states to be revisited by replaying the sequence of seeds [25]. Figure 2.9 depicts the AST Black-Box architecture and the interactions between the parts. The next state s' is no longer available to the RL-agent $\mathcal{A}$, and the reward function is given the transition likelihood ρ directly from the simulator $\bar{\mathcal{S}}$.

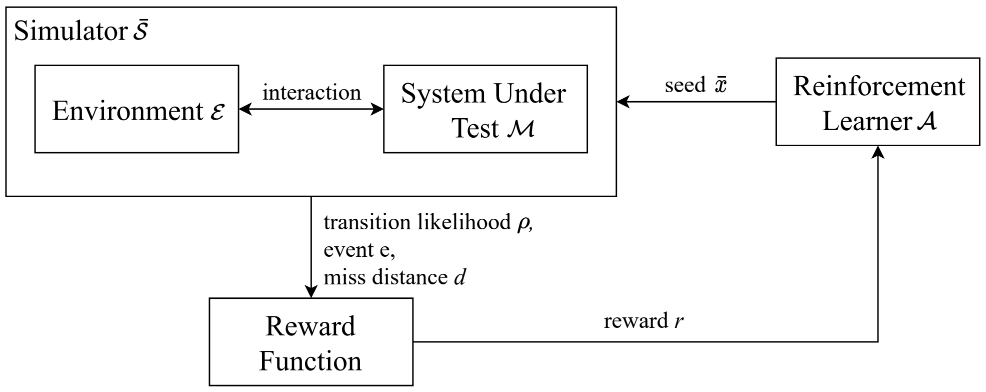

Simulator $\bar{\mathcal{S}}$
Environment $\mathcal{E}$
interaction
System Under Test $\mathcal{M}$
seed $\bar{x}$
Reinforcement Learner $\mathcal{A}$
transition likelihood ρ,
event e,
miss distance d
Reward Function
reward r

Figure 2.9: Adaptive Stress Testing for Black-Box Simulators. This figure shows the AST architecture for Black-Box simulators. The RL-agent $\mathcal{A}$ choose an action-seed $\bar{x}$ that is set on the Black-Box simulator $\bar{\mathcal{S}}$. Since the Black-Box simulator does not reveal its internals, the environment disturbances are sampled internally in the simulator $\bar{\mathcal{S}}$, and the transition likelihood ρ, therefore, needs to be calculated by the simulator. The transition likelihood ρ, event e, and miss distance d are returned to the RL-agent $\mathcal{A}$. The Black-Box approach does not return the next state s'. This figure is based on [25, 26].

2.9 Related Work

In this section, work related to AST is presented. Here, we will focus on AST related to airborne collision avoidance systems and autonomous vehicles.

2.9.1 Adaptive Stress Testing of Airborne Collision Avoidance Systems

AST was first proposed by Lee et al. [22] in 2015 to test airborne collision avoidance systems. AST is used to stress test a prototype of the next-generation Airborne Collision Avoidance System (ACAS X). Lee et al. [25] investigated this, where the goal was to find the most likely path to a near mid-air collision. They had to work in a large search space, in which exhaustive considerations were not suitable, and failure states could be hard to find.

Different architectures were presented, and depending on whether the simulator was fully observable or partially observable, this is what we refer to as Gray-Box and Black-Box in this thesis, respectively. They proposed a variant of MCTS called Monte Carlo Tree Search for Seed-Action simulators (MCTS-SA), which only requires access to the pseudorandom number generator of the simulator to overcome partial observability, this is what was presented in Section 2.8.2. The algorithm is based on progressive widening, which is done due to the huge action space consisting of all possible pseudorandom seeds. More about this variant in Section 3.4. This simulator has a deterministic behavior since the same pseudorandom seed always leads to the same next state from the previous state. Due to this, the transition behavior of the simulator is deterministic [25].

Lee et al. [23], extended the AST framework to be able to apply regression testing to find failures that occur in one system but not in another. This extended framework is called Differential Adaptive Stress Testing (DAST). The framework is used to compare ACAS X with Traffic Alert and Collision Avoidance (TCAS), to test the performance of ACAS X relative to TCAS. DAST works by searching two simulators simultaneously and maximizing the difference between their outcomes [23].

It is essential to understand how failures occur to be able to design, evaluate and certify safety-critical systems. In context with this, AST and DAST helped to contribute to the certification case of the ACAS X, which led to the acceptance of ACAS X by the RTCA [25].

Lipkis et al. [26] also use AST to test airborne collision avoidance systems. In particular, the Airborne Collision Avoidance System for smaller UASs (ACAS sXu). The purpose of their work is to provide detect-and-avoid capability for small unmanned aircraft operating beyond line-of-sight. They use a different approach compared to Lee et al. [25], where they apply Deep Reinforcement Learning (DRL) with a Proximal Policy Optimization (PPO) algorithm [33], to be able to search more efficiently through the large and continuous state space. The developed method in their work found several failure events, which were useful for the development of the system.

2.9.2 Adaptive Stress Testing for Autonomous Vehicles

In order for a vehicle to become autonomous, it is crucial that the autonomous vehicle is equipped with a decision-making system. Koren et al. [20], presents a method for testing the decision-making system of autonomous vehicles. They formulate the problem as an MDP, and use RL-algorithms to find the most likely failure scenarios. In their work, they show that extending AST to use DRL has improved the efficiency of the original AST, which use a MCTS variant. They simulated scenarios involving an autonomous vehicle, as the system under test, which is approaching a crosswalk, with pedestrians as other actors in the environment. The paper shows that DRL can find more-likely failure scenarios than MCTS in addition to finding them more efficiently [20].

Chapter 3

Methodology

This chapter presents the methodology of this thesis. First, the collision avoidance simulator used for testing in this thesis will be described. Second, the architecture and methods used in this thesis will be presented.

3.1 The milliAmpere Collision Avoidance Simulator

The maritime autonomous collision avoidance system is tested by using a simulator provided by Zeabuz. The simulator is based on the collision avoidance system milliAmpere uses, which is the SP-VP method presented in Section 2.2. The collision avoidance system is part of a larger system that milliAmpere uses, but has been isolated from the bigger system in this simulator to make it easier to test. Since milliAmpere is a safety-critical system with several complex components, and operates in a complex and stochastic environment, it is safer to perform the testing in a simulated environment. The simulator also gives the opportunity to cover a bigger part of the state space than real-world testing, which gives a higher chance of finding rare failure events.

The simulator uses the simplifying assumptions presented in Table 2.2 in Section 2.1.3. To simplify the problem into flat earth navigation in three degrees of freedom where the marine vessels operate in the NED-frame in 3DOF. The simulator provides two types of vessels, the milliAmpere vessel and a vessel referred to as a first-order vessel. The milliAmpere vessel operates with more complex dynamics than the first order vessel and is always equipped with the SP-VP controller. The first-order vessels use first-order differential equations for their dynamics and behave as obstacles to the SP-VP controller in the milliAmpere vessel. The obstacles use a simpler controller referred to as the speed heading controller, which we will get back to later in Section 3.1.1. The marine vessels' heading ψ operates in the unit circle. The name "obstacle" is chosen to comply with the same notation as Thyri et al. use in their work [40]. An alternative name could be an "adversary", for instance, which game theory with RL often use. The marine vessels' dynamics are based on the kinematic equations discussed in Section 2.1.3.

The simulator is built both as a Gray-Box and a Black-Box system in this thesis. These were introduced previously in Section 2.4. The use of a Gray-Box simulator approach would be sufficient to test the collision avoidance system in this thesis, but a Black-Box approach is also built for comparing the ease of use and test results. The difference between the two approaches is how the random processes in the simulator transition from a state s_t to the next state s_{t+1}. In the Gray-Box approach, an initial simulator seed is set and is re-set in each simulation; therefore, the states transition deterministically, and the random processes will always have the same outcome from a given state s_t to the next state s_{t+1}. In the Black-Box simulator, however, an initial seed is set in each simulation, but new simulator seeds are also performed as actions by the RL-agent. From Section 2.8.2, we remember that the simulator then deterministically transitions given the seed, which allows previously visited states to be revisited by replaying the sequence of seeds. This is the same approach that Lee et al. [25], refer to as a seed-action simulator in his article.

When the simulator was first received, it had features to simulate milliAmpere together with multiple obstacles, and the dynamics of the vessels were already implemented. The simulation visualizer was also in place and ready to use. The SP-VP controller was also implemented together with tools to visualize the path-time graph and a SP-VP cost function. However, the simulator was not ready to be used together with the AST-framework, and additional features had to be implemented. The functions implemented have similarities with the related work presented in Section 2.9. However, adjustments had to be made to fit the maritime domain. The functions are presented below, due to the non-disclosure agreement, the code can not be shared:

Function to make it possible for the RL-agent to steer the obstacle vessels. The function is implemented by using the existing speed heading controller in the provided simulator code base. The RL-agent chooses a reference for the surge speed u and the heading ψ of the vessel, and the reference is used by the speed heading controller to control the motions of the obstacle vessels. More about this controller in Section 3.1.1.

Function to make it possible for the RL-agent to control the SP-VP noise. The function is implemented by using the existing noise model in the provided simulator code base. The RL-agent sample noise values from a Gaussian distribution with mean $\mu = 0.0$ and standard deviation $\sigma = 1.0$. The noise model use time constants and gain parameters T_p, K_p, T_v, K_v, where subscript p is for the vessel position and v is the vessel velocities.

Function to check if a failure event has occurred. The failure event function is implemented to check if the milliAmpere vessel intersects with the obstacle vessel's true ROC, which is the ROC without any noise. In Section 2.2.1, equations for calculating the true ROC is discussed. A failure event is therefore defined as a collision between milliAmpere and one of the obstacles in the simulation, if milliAmpere intersects one of the obstacle vessels' true ROC.

Function to calculate the distance between one or more obstacles. To calculate the distance between two vessels in the NED-frame, the Euclidean distance formula shown in Equation 3.1 is used:

$$d(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}, \tag{3.1}$$

where p and q are points in with Cartesian coordinates (p_1, p_2) and (q_1, q_2), respectively. When there are multiple vessels in the simulation, the average of all distances between milliAmpere and the other obstacle vessels is returned.

Function to calculate the transition likelihood of the simulator. This function calculates the simulator's transition likelihood $p(x|s)$.

Function to reset the simulator. This function resets the simulator to its initial conditions and re-sets the initial seed of the simulator.

Function to step the simulator by providing an RL-action. The function takes disturbances given by the RL-agent and performs the actions caused by the disturbances. The simulator is then updated with a resolution of $0.1s$ time step and returns the simulator state s back to the RL-agent with a resolution of $1s$ time step. This means that the RL-agent only interacts with the simulator with a resolution of $1s$ time step.

Simulator Data Extractor. The purpose of the data extractor was to extract all the data needed to analyze the results manually and with the Soft-DTW algorithm.

To develop the functions, domain knowledge in marine vessel dynamics introduced in Section 2.1 and about the collision avoidance system SP-VP, introduced in Section 2.2, was needed. They are also fundamental for the architecture proposed in this thesis, more about this architecture is in Section 3.3.

In Figure 3.1 an example simulation frame is shown, consisting of the milliAmpere vessel and two first order vessels referred to as *Obstacle 1* and *Obstacle 2*. The milliAmpere vessel is illustrated by an orange solid circle representing the center of the vessel together with the belonging trajectory as a solid line in the same color. The milliAmpere vessel's waypoint mission is shown as a dashed black line. The centers of the obstacle vessels are also represented by solid circles with their corresponding colors, but the trajectories differ from the milliAmpere because they also consist of markers showing their position every second of the simulation. The solid red, green and blue lines, are ROC, HPR and LPR of the SP-VP obstacle representation respectively, these are presented in Section 2.2.1. The diamond-shaped dashed lines surrounding the obstacles are the true ROC, this differs from the ROC that SP-VP uses to represent obstacles by not being affected by noise.

3.1.1 Obstacle Vessels with Speed Heading Controller

The obstacle vessels in the simulator also referred to as the first-order vessels, are first-order control systems. These systems use a transfer function that is a first-order differ-

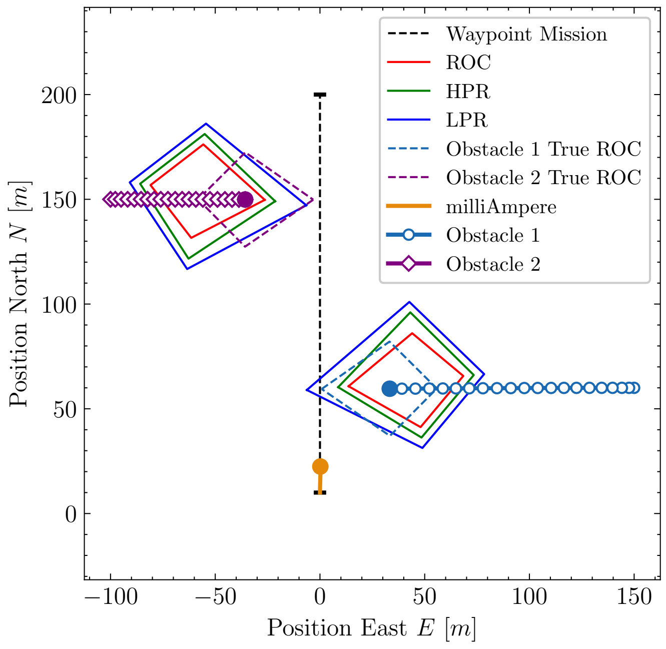


Figure 3.1: Simulation Frame. This figure shows an example simulation frame. The milliAmpere vessel is illustrated with an orange solid line, and the dashed black line is milliAmpere's waypoint mission. The solid red, green and blue lines, are ROC, HPR and LPR of SP-VP, respectively. The obstacles are represented by lines with markers for every second of the simulation. We have obstacle 1 in light blue with circles as markers and obstacle two in purple with diamonds as markers. True ROC is a region of an obstacle that indicates an actual collision, it differs from the SP-VP ROC due to noise given to the collision avoidance system.

ential equation. The vessels are based on Proportional Integral Derivative (PID), where the proportional part is the Speed Heading Controller, where a reference surge speed u_* and vessel heading ψ_* are given to the controller. The controller use Equation 3.2:

$$r_* = -K_\psi(\psi_* - \psi) \tag{3.2}$$

where r_* is the desired yaw angular velocity, K_ψ is the proportional gain in yaw. The reference heading ψ_* and current heading ψ. Moreover, the feedback error term is the $(\psi_* - \psi)$ part of the equation [42]. The Speed Heading Controller returns the desired body velocity back to the vessel by Equation 3.3:

$$\boldsymbol{\nu}_* = [u_*, 0, r_*]^T \tag{3.3}$$

where $\boldsymbol{\nu}_*$ is the desired body velocity, and u_* and r_* are the desired body velocities in surge and yaw, respectively. Another important part to mention about the PID controller, is that it uses time constants to control how long it should take to reach

63.2% of the desired velocity in these first-order vessels. The time constants that the first-order vessels use in this thesis are T_u, T_v, and T_r, which are the time constants for surge velocity u, sway velocity v and yaw angular velocity r. In the experiments conducted in this thesis, these are set to $T_u = 3$, $T_v = 5$ and $T_r = 3$.

3.1.2 The milliAmpere Vessel Configurations

Table 3.1 shows an overview of the milliAmpere configurations. These configurations are used in all experiments conducted with the AST-framework. The regions in these configurations have been chosen by Zeabuz, which is why they are used in this thesis. Here, the vessel has an initial position of 10 meters North N and 0 meters East E with a heading straight towards North, in the northeast frame. The initial position is the start waypoint of the waypoint mission given to the vessel, and the goal waypoint is 200 meters straight North N. The SP-VP collision avoidance system is updated every fourth second of the simulation and has the obstacle region margins ROC, HPR and LPR as shown in the table.

It is important to remember from Section 2.2.1, why the large region margins are chosen for the obstacles. The reason is that it includes the dimensions for itself together with the vessel using the SP-VP collision avoidance system, which is milliAmpere in our case, in addition to some safety factors and perimeter size. The regions are also larger towards the starboard due to COLREG saying that you should stop for other vessels from the starboard. The table also shows the gains and time constants in use by the SP-VP noise model The time constants T_p and T_v are time constants for noise in position and velocity, respectively. The gain parameters K_p and K_v are time constants for position and velocity, respectively.

3.1.3 Infeasible Problem

The SP-VP method cannot always find a velocity profile that does not lead to a collision. In these cases, the simulator provided by Zeabuz throws an *infeasible problem* exception. The work in this thesis had to handle this exception to avoid the simulator crashing during testing. As mentioned earlier, SP-VP is part of a bigger system that milliAmpere uses, and other parts take control when this exception is present. For testing in this thesis, the infeasible problem is handled by setting the next trajectory waypoint of the milliAmpere vessel to the vessel's current position in North N and East E, which makes the vessel stop to avoid a collision. This behavior was evaluated as most safe for milliAmpere. The adjustment was done to be able to easier test AST, and to be able to find more failure events. Furthermore, the infeasible problem is also handled by setting it as a terminal state in AST. This is done to be able to find even more realistic and rare failure events. In most of the experiments conducted in this thesis, the infeasible problem is handled by stopping the milliAmpere vessel. However, in Section 4.6, the experiment conducted with infeasible problems handled as a terminal state is discussed.

Table 3.1: The milliAmpere vessel configurations

Initial Positions and Velocities			**SP-VP Noise Parameters**	
North N	East E	Heading ψ	T_p	30
10	0	0	K_p	100
Surge Velocity u	Sway Velocity v	Yaw Velocity r	T_v	30
0	0	0	K_v	4.66

Max Velocity	**Min Velocity**	**SP-VP Rate**	**Waypoint Mission**
1.2	-0.2	0.25	[N:10, E:0] $\rightarrow$ [N:200, E:0]

SP-VP Obstacle Region Margins				
	Fore l_f	Starboard l_s	Aft l_a	Port l_p
ROC	22.5	12.5	12.5	12.5
HPR	27.5	22.5	17.5	17.5
LPR	42.5	27.5	22.5	22.5

3.2 AdaStress

The software package AdaStress[1] implements the AST-framework introduced in Section 2.8. AdaStress is developed by the Robust Software Engineering technical area, based in the Intelligent Systems Division at NASA's Ames Research Center. The software package is based on the original AST formulation first introduced by Lee et al. [25]. It is written in the Julia[2] programming language and provides three primary services:

- Interfaces between simulators and the AST-framework.
- Different solvers based on RL.
- Tools for analyzing and visualizing the results from the AST.

When using AdaStress, it is possible to choose from two different types of simulator interfaces, which are the Gray-Box and Black-Box interfaces. The Gray-Box interface is used when the simulator's environment is available to the solver, this makes it possible to sample the random variables directly and apply them in the simulation. The Black-Box interface, on the other hand, is used when the simulator does not reveal its environment. In this case, the solver has to interact with the simulator by setting a random seed for each step in the simulation.

To make use of AdaStress, the simulator has to provide methods that satisfy the AdaStress methods below:

- **AdaStress.reset!**: Resets the simulator to its initial conditions.

[1] AdaStress
[2] The Julia Programming Language: https://julialang.org/

- **AdaStress.isterminal**: Checks if the current simulation state is terminal or not.
- **AdaStress.isevent**: Checks if the current state is a failure event or not.
- **AdaStress.distance**: Returns the distance to a failure event.
- **AdaStress.step!**: Transitions the simulator into its next state. If the simulator is a Gray-Box it is done by performing a set of disturbances in the simulator. If the simulator on the other hand is a Black-Box, it is done by setting a random seed in the simulator. The function also returns the current state log probability of the environment if it is a Black-Box.
- **AdaStress.environment (Gray-box)**: This method is only needed when the simulator is a Gray-Box, which makes the environment available to the solver. The methods are used to set up the stochastic variables by sampling from a probability distribution. This method is not needed by the Black-Box interface due to environment variables updating internally in the simulator.
- **AdaStress.observe**: This is an optional method that returns the current simulation state.

AdaStress has been used in this thesis to perform AST on the milliAmpere simulator discussed in Section 2.8. As mentioned previously, both Gray-Box and Black-Box simulators have been implemented and tested.

3.3 Maritime Adaptive Stress Testing

This section presents the Maritime Adaptive Stress Testing (MAST) architecture proposed in this thesis. This architecture is the one this thesis uses for testing the maritime autonomous collision avoidance system SP-VP. The architecture consists of three main parts, the AST-framework, the collision avoidance simulator, and a simulator interface needed by the AST-framework to communicate with the simulator. Figure 3.2 shows an abstract overview of the MAST architecture. Moreover, two different programming languages are used in this work, the reason is that the simulator provided by Zeabuz is written in Python[3] and the AST-framework AdaStress developed by NASA is written in the programming language Julia, the AdaStress software package was presented in Section 3.2. The collision avoidance simulator was not ready to be used together with the AST-framework when first received. Therefore, a simulator interface that provided all the necessary communication between the AST-framework and the simulator had to be implemented. The functions and extra features of the simulator were presented in Section 3.1.

In the previous Section 3.1, there was briefly mentioned that the simulator is implemented both as a Gray-Box and Black-Box simulator in this thesis. This section will give a more detailed introduction to these simulators and how they fit in with the MAST

[3]Python: https://www.python.org/

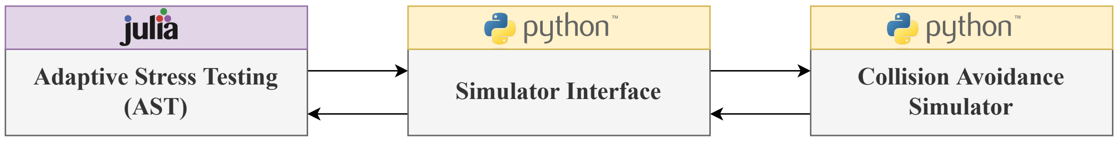


Figure 3.2: Maritime Adaptive Stress Testing. In this figure, an abstract overview of the MAST architecture is shown. Two different programming languages are used; the simulator provided by Zeabuz is implemented in Python, and the AST-framework, developed by NASA, is implemented in Julia. In between, we have the simulator interface developed in this work and implemented in Python, which helps the communication between the AST software package and the simulator. The arrows indicate communication between the parts.

architecture. First, the Gray-Box approach will be discussed, and next, the Black-Box approach will be discussed.

3.3.1 Gray-Box Maritime Adaptive Stress Testing

The Gray-Box Maritime Adaptive Stress Testing (G-MAST) is based on what we referred to as AST for Gray-Box simulators in Section 2.8.1. The proposed architecture extends the existing architecture by adapting it to the maritime domain. G-MAST is a suitable solution when the simulator makes its environment variables and state available. The RL-agent can then sample the variables, also referred to as disturbances, directly from a modeled probability distribution for each type of disturbance. The chosen probability distributions will be described in more detail in Section 3.6.

Figure 3.3 shows the G-MAST architecture. The architecture is an extended version of Figure 2.8 in Section 2.8.1. Here, the RL-agent samples a set of disturbances, which is further sent to the simulator. The disturbance handler then splits up the disturbances vector and sends the correct disturbance into their belonging parts in the simulator. The reference surge speed u_* and heading ψ_* are sent directly to the Speed Heading Controller of the obstacle vessel. It is possible to simulate more than one obstacle, but for simplicity, only one obstacle vessel is illustrated. The sampled noise is sent to the SP-VP noise model, which generates some noise on the SP-VP tracking system, which then returns the estimated position of the obstacles to the SP-VP controller of the milliAmpere vessel. The simulator then transitions into its next state. The vessel state for milliAmpere and the obstacle consists of the positions and orientations η and the velocities ν of the vessels. Then the simulator checks if the state is a failure event e, the distance to failure e, and formats the state to a one-dimensional vector which is sent back to AST. The reward function calculates the transition likelihood in this architecture and returns the reward r to the RL-agent.

3.3.2 Black-Box Maritime Adaptive Stress Testing

The Black-Box Maritime Adaptive Stress Testing (B-MAST) architecture is suitable for simulators that do not reveal their environment variables and where all the updates of

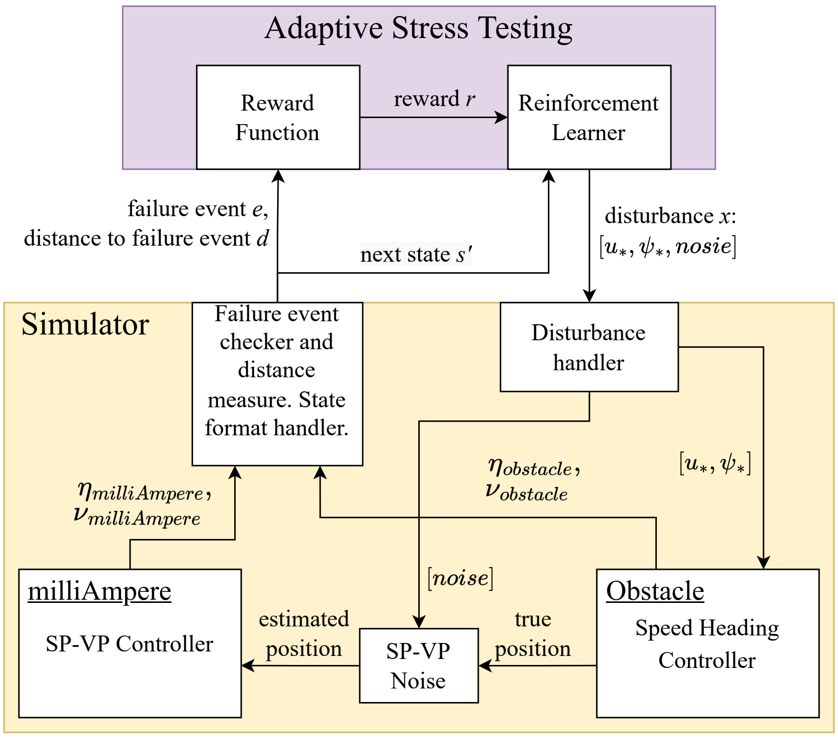


Figure 3.3: Gray-Box Maritime Adaptive Stress Testing. This figure shows a more detailed architecture setup of G-MAST. The RL-agent chooses disturbances that are sent to the simulator's disturbance handler. The disturbance handler sends the disturbances to the right parts of the simulator. The reference surge speed u_* and heading ψ_* are sent to the speed heading controller. And the noise is given to the SP-VP tracker. The simulator then updates and transitions into its next state, and the state of the vessels is given to the failure event checker, distance measure, and state format handler. Then the simulator returns a boolean indicating if the state is a failure state or not e, the distance to failure d, and the next state of the simulator. The reward function calculates the transition likelihood in this architecture and the reward r is sent to the RL-agent.

the simulator happen internally. This architecture is based on what we referred to as AST for Black-Box simulators in Section 2.8.2. The RL-agent only interacts with the simulator by setting the random seed. This differs from the Gray-Box approach because the random stochastic disturbances are now sampled internally in the simulator instead of by the RL-agent. In Figure 3.4, the architecture is shown. Only a random seed, referred to as action-seed $\bar{x}$, is chosen by the RL-agent and given to the pseudorandom number generator of the simulator. The pseudorandom number generator is then used by the Speed and Heading Reference handler to sample a random reference surge speed u_* and heading ψ_* for the obstacle's Speed Heading Controller. The pseudorandom number generator is further used to generate random noise in the SP-VP tracking system. The simulator works the same way as in the Gray-Box approach, but the disturbances in the environment are now sampled internally in the simulator. This means that the transition likelihood ρ has to be calculated internally in the simulator and returned to the reward function, which is needed for the reward function to be able to calculate the reward r. The Black-Box approach can also be suitable for simulators that are provided as a software binary not revealing its state, which is the case in the work of Lee et al. [25].

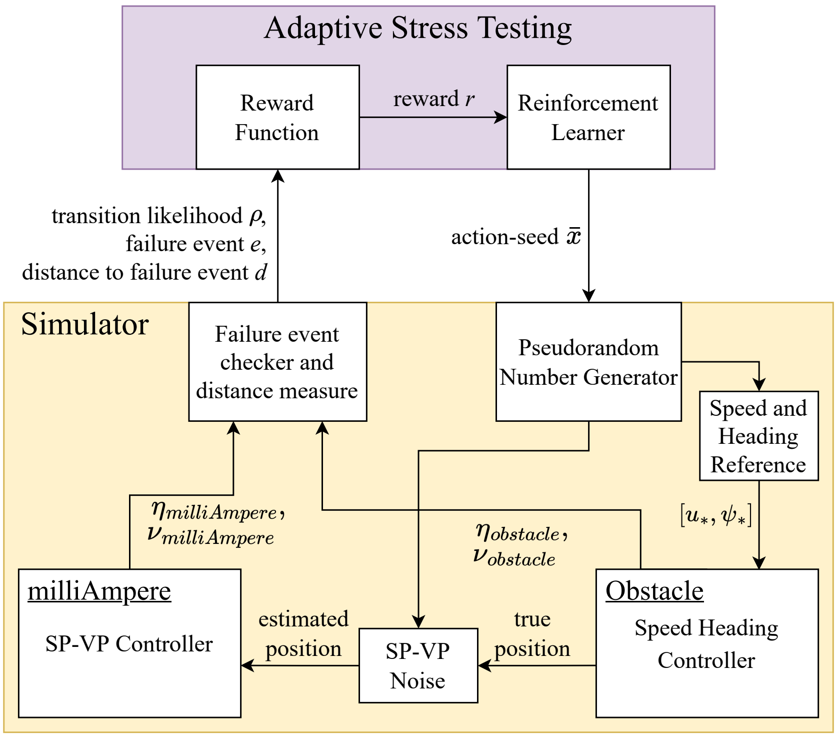

Adaptive Stress Testing
Reward Function
reward r
Reinforcement Learner
transition likelihood ρ, failure event e, distance to failure event d
action-seed $\bar{x}$
Simulator
Failure event checker and distance measure
Pseudorandom Number Generator
Speed and Heading Reference
$\eta_{milliAmpere}$, $\nu_{milliAmpere}$
$\eta_{obstacle}$, $\nu_{obstacle}$
$[u_*, \psi_*]$
milliAmpere
SP-VP Controller
estimated position
SP-VP Noise
true position
Obstacle
Speed Heading Controller

Figure 3.4: Black-Box Maritime Adaptive Stress Testing. This figure shows the B-MAST architecture. The RL-agent chooses an action-seed to be set on the simulator in each simulator step. The pseudorandom number generator in the simulator is used to sample environment disturbances internally. The disturbances are the reference surge speed u_* and heading ψ_* and noise to SP-VP. The transition likelihood has to be calculated internally in the simulator for B-MAST.

3.4 Monte Carlo Tree Search with Progressive Widening

In this thesis, the Monte Carlo Tree Search with Progressive Widening (MCTS-PW) is used [7]. The progressive widening extends MCTS, introduced in Section 2.7, to be able to handle large or continuous action spaces. This is the algorithm the software package AdaStress offers [7]. The progressive widening variant of MCTS, is also what Lee et al. [25], use in their proposed variant referred to as MCTS-SA.,

The MCTS-PW algorithm is used in the already visited state nodes to determine whether to use existing disturbances already tried, or expand the node to try out new disturbances by using the progressive widening criteria, expressed in Equation 3.4:

$$|X(s)| < kN(s)^{\alpha}, \tag{3.4}$$

where s is the traversed node, $|X(s)|$ is the number of disturbances, also referred to as children of the traversed node s. The $N(s)$ is the number of times the node s has been traversed. The parameters k and α are the tree expansion coefficient and exponent, respectively. The tree expansion coefficient k and the tree expansion exponent α are additional hyperparameters that control the rate of adding new nodes.

3.5 Reward Function

The G-MAST and B-MAST use different types of reward functions. Since the simulator in the G-MAST architecture returns the state s, the reward function can calculate the transition likelihood itself. The reward function for G-MAST is expressed in Equation 3.5:

$$R(s,x) = \begin{cases} R_E & \text{if } s \text{ is terminal and } s \in E \\ -d & \text{if } s \text{ is terminal and } s \notin E \\ \log(p(x|s)) & \text{otherwise,} \end{cases} \tag{3.5}$$

where R_E is the reward when a failure event is found. This is set to be high enough to outweigh the maximum cumulative unlikeliness.

Since the state is not returned in the B-MAST architecture and environment variables are handled internally in the simulator, the transition likelihood must also be calculated internally in the simulator. The transition likelihood is returned to the reward function and can thereby be expressed as in Equation 3.6:

$$R(\rho,e,d,\tau) = \begin{cases} R_E & \text{if } \tau \wedge e \\ -d & \text{if } \tau \vee \neg e \\ \log \rho & \text{otherwise,} \end{cases} \tag{3.6}$$

where the reward function no longer has the state s and the disturbance x available from the simulator. Instead, the simulator returns the transition probability ρ, a boolean indicating if it is a failure event e and a miss distance d, and a boolean indicating if the simulator is a terminal state τ. If we compare the reward function for the Gray-Box approach and the Black-Box approach, we see that the reward is calculated in the same way; but the difference is that some of the parts needed in the equations are calculated internally in the simulator for the Black-Box approach.

In this thesis, the reward is also marginalized when calculating the log probabilities.

3.6 Representation of States and Disturbances

In the G-MAST architecture, a state is returned to the solver. The state s is then represented as:

$$s = [N_m, E_m, \psi_m, N_{o1}, E_{o1}, \psi_{o1}, ..., N_{on}, E_{on}, \psi_{on}], \tag{3.7}$$

where N, E, and ψ are the North position, the East position, and the heading of the vessel, respectively. The subscript m denotes the milliAmpere vessel, which is the system under test. The subscript on denotes that the vessel is an obstacle o and the number of the obstacle vessel n. The B-MAST architecture does not use the state due to handling states internally.

The disturbances are sampled from the same types of distributions in both G-MAST and B-MAST. The difference is that in B-MAST, the disturbances are sampled internally in the simulator and not by the RL-agent as it is in the B-MAST. Three types of disturbances are used in this thesis, which are the reference surge speed u_* and the reference heading ψ_* of the obstacle vessel, and the noise to the SP-VP tracking system. The vessel reference speed u_* and reference heading ψ_* are sampled from a truncated normal distribution [6], which truncates the values returned from the distribution into an interval. The reference surge speed u_* use the truncated normal distribution with a minimum allowed velocity $u_{\min}$ and a maximum allowed velocity $u_{\max}$. The mean of the distribution is set to the velocity that should be treated as most likely in the given scenario. The standard deviation σ is set to be reasonably high so that it covers the given interval such that it samples the minimum and maximum values with some frequency. The same approach is done for the heading ψ, but here the mean is set to the initial heading, which makes the initial heading of the vessel to be the most likely, which makes deviation from this heading less likely. The noise is sampled using a Gaussian distribution with the mean $\mu = 0$ and standard deviation $\sigma = 1.0$.

3.7 Representation of Failure Events

In this section, the definition of a failure event is introduced together with the types of maritime collisions used in this thesis. A failure event occurs when the milliAmpere intersects with one of the other obstacle vessels' true ROC. Maintenance and Cure[4] classify four types of maritime collision types:

- **Side collisions:** when a vessel is struck on the side by another vessel
- **Bow-on collisions:** when two vessels strike each other head-on
- **Stern collisions:** when one vessel runs into the back of another.

[4]Maintenance and Cure, Maritime Collision Types: www.maintenanceandcure.com/maritime-blog/types-of-maritime-collision-causes

- **Allisions:** when a vessel strikes an object, such as a bridge.

These types of maritime collisions are used in this thesis to explain and group the failure events. We will come back to these collision types in Chapter 4, except for *allisions*, which is not relevant in this thesis.

Chapter 4

Results and Discussion

In this chapter, the results from G-MAST and B-MAST applied to the Zeabuz ferry will be presented and discussed. In all of the experiments, the milliAmpere vessel is using the configurations presented in Section 3.1.2. The most interesting failure events will be presented together with the scenario setup and hyperparameters used to find them.

4.1 Hardware and Software

For reproducibility purposes, the hardware specifications of the computer used in this thesis are presented here. The operating system is *Arch Linux x86_64*, with the kernel version *5.19.5-arch1-1*. The CPU is a *Intel i7-6700K 4.2 GHz*, the GPU is *NVIDIA GeForce GTX 980* and *32 GB RAM*.

4.2 Scenarios Setup

It is common when testing maritime autonomous collision avoidance systems to check if it is COLREG-compliant. But as mentioned earlier in Section 2.2, the SP-VP method studied here does not comply well with these rules due to having a fixed path, and the COLREG is weighted more on course change maneuvers. This thesis does, therefore, not focus on checking if the SP-VP method is COLREG-compliant, but instead tries to find rare events where the SP-VP tracking system gets confused, which makes the algorithm plan dangerous velocity profiles. Three types of scenarios were designed to do this, the first one is referred to as fast-moving obstacles, the second is referred to as slow-moving obstacles, and the last one is referred to as multiple obstacles, which includes two obstacles. In all of the scenarios, milliAmpere was configured the same way, as presented in Table 3.1 in Section 3.1.2. The obstacle vessel's initial position was placed on the starboard side of milliAmpere, this is because vessels should not conflict with the path of the vessels coming on their starboard side given rule 15 in COLREG.

4.3 Fast-moving Obstacles

In the first experiments conducted, the obstacles steered by the RL-agent operate with relatively large velocities. The reason why this was tested was not to look for the most realistic scenarios, but instead to have a proof of concept and a base to work from. In Table 4.1, the configurations for the obstacle vessel for the experiments performed with fast-moving obstacles are presented. The initial position of the obstacle vessel is 150 meters North N and 200 meters East E with a heading ψ straight west towards milliAmpere's waypoint mission. The initial distance between the obstacle and system under test might seem large, but it is important to remember that the system under test is represented as a point in the northeast frame, but it takes more space in the real world. The RL-agent has the freedom to steer the obstacle however it wants with the constraint of the surge velocity $u \in [-1, 10]$ and the heading $\psi \in [-\pi, \pi]$. The SP-VP noise was sampled from a Gaussian distribution as presented in 3.6. In this experiment, the infeasible problem is handled by stopping the vessel. The infeasible problem was introduced in Section 3.1.3.

Table 4.1: Fast-moving obstacle scenario

Obstacle Vessel 1		
Initial Positions and Velocitites		
North N	East E	Heading ψ
150	200	$-\pi/2$
Surge Velocity u	Sway Velocity v	Yaw Velocity r
2.0	0	0
Velocity and Gain		
Max Surge Velocity u_{max}	Min Surge Velocity u_{min}	Yaw Gain
10.0	-1.0	0.2

In Table 4.2 the hyperparameters for the MCTS-PW solver are presented. These hyperparameters are the default hyperparameters in the AdaStress software package. The number of iterations could be set lower, but because of benchmarking the results from G-MAST and B-MAST, a minimum of 50 000 iterations are set for all experiments. The tree expansion coefficient k and the tree expansions exponent α are set to 1.0 and 0.7, respectively. These are the parameters for the progressive widening part of the MCTS-PW, which is presented in Section 3.4. The exploration balance constant c is for the UCT in MCTS-PW.

In the rest of this section, interesting failure events found with one fast-moving obstacle are presented. All of the results presented in this section are found with G-MAST. However, all types of failure events presented here are also found with B-MAST. First, the most common failure event found is shown in Figure 4.1. The collision occurs at

Table 4.2: MCTS-PW hyperparameters

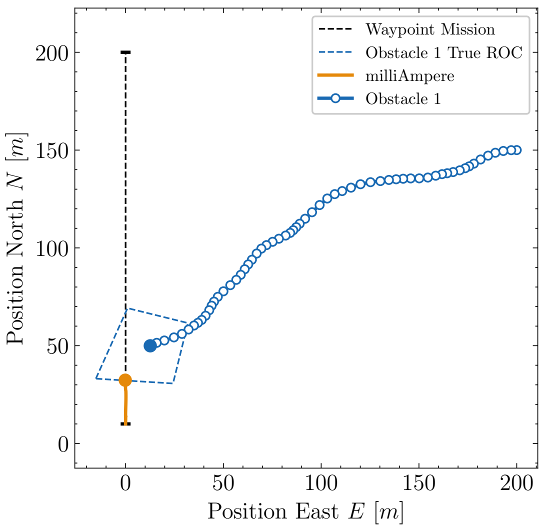
Hyperparameter	**Value**
Number of iterations n	50 000
Tree expansion coefficient k	1.0
Tree expansion exponent α	0.7
Exploration balance coefficient c	1.0

time t 60 seconds, which happens while the milliAmpere vessel is standing still. The results presented here show that the failure event found with G-MAST is a collision where milliAmpere is not speeding when the collision occurs. One might argue that these types of failures are not caused by the system under test but rather an apparent attack by the obstacle vessel. Others might argue that it is a form of failure because it tells us that the collision avoidance system favors stopping when it is uncertain. The obstacle vessel should maybe be avoided instead by using other collision avoidance techniques where the system under test will steer away from the situation. Due to legal reasons, the latter option is often precluded because it is easier to argue that the system did not cause the failure because it is not speeding. It also depends on the maneuverability of the obstacle, and the argument of the obstacle is chasing the milliAmpere, and the collision would occur anyway due to the obstacle being unavoidable in this case. However, these types of failures are not failures in the SP-VP system due to it being designed to handle these situations by stopping the vessel in these experiments.

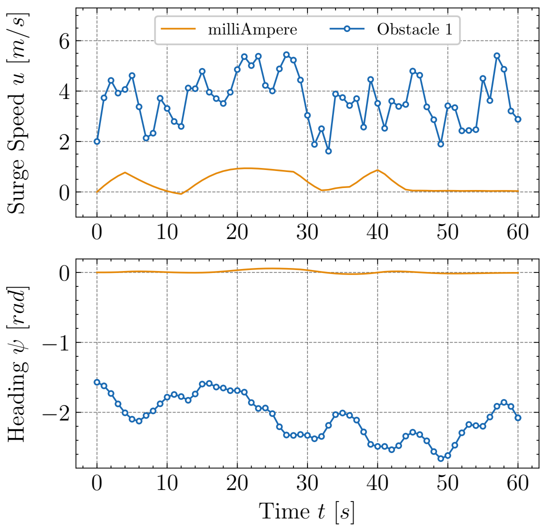
Furthermore, this is a common failure event found in some autonomous systems, such as autonomous vehicles, for instance. An example of this is shown in work by Koren et al. [20], where the blame for the collision in some of the scenarios is on the pedestrian. The pedestrians in their work behave a lot like the obstacle vessels in this thesis. It is a common result because many autonomous systems have parts built in so that when they are uncertain, they will slow down, as opposed to speeding up.

The next interesting failure event found with fast-moving obstacles, is the bow-on collision shown in Figure 4.2. In this case, milliAmpere is, in fact, speeding when the collision occurs at time t 60 seconds, with a speed of $0.73m/s$. The interesting part here is that it looks like the obstacle vessel tricks the milliAmpere into crashing by changing its heading from almost straight west to straight south towards the milliAmpere vessel at time t 53 seconds. When looking at the graph for the surge speed u for milliAmpere, it shows that it has an almost constant speed over 10 seconds before the collision occurs.

Figure 4.3 shows a side collision at time t 53 seconds, where the milliAmpere vessel contributes to the crash. It crashes into the obstacle vessel when it tries to cross. This violates rule 15 of COLREG, which states that milliAmpere should stop for the obstacle vessel coming on its starboard side. Figure 4.3b shows that the milliAmpere is speeding when the collision occurs. Rule 19 in COLREG states that “every vessel should proceed at a safe speed adapted to prevailing circumstances and restricted visibility. A vessel



(a) Northeast plot.

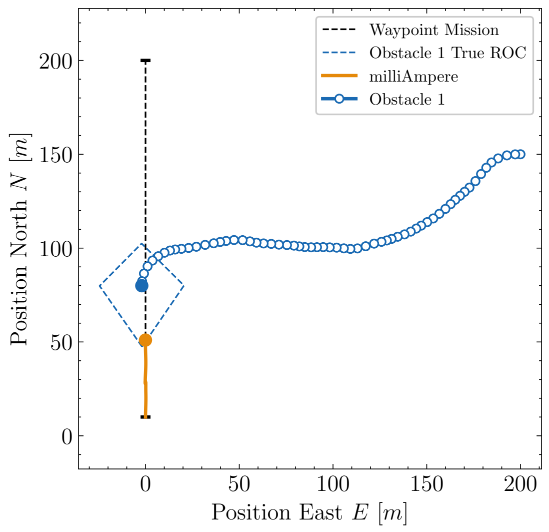


(b) Surge speed u and heading ψ.

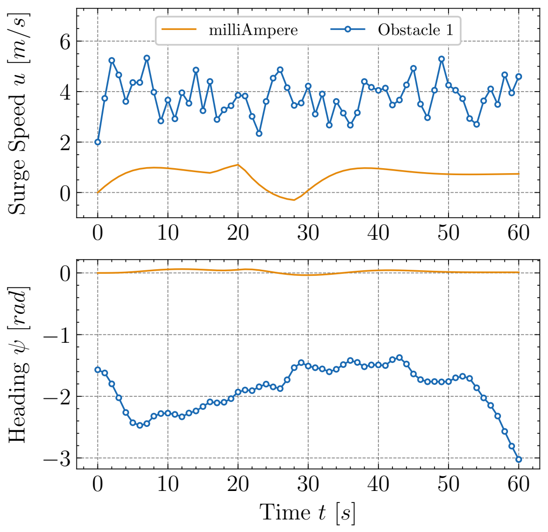
Figure 4.1: The Most Common Failure Event Found. This figure shows the most common failure event found. Figure 4.1a shows the trajectories for the milliAmpere vessel and the obstacle vessel from their start position until the collision occurs at time t 60 seconds. The obstacle vessel is steering right into the milliAmpere vessel in this case. Figure 4.1b shows the vessels' surge speeds u and headings ψ in each time step of the simulation. The key observation from this figure is that the milliAmpere vessel stops for the rest of the simulation at time t 44 seconds and is not speeding when the collision occurs. The obstacle heading ψ is towards the milliAmpere vessel. The obstacle vessel has a surge speed u of 2.88 m/s when the collision occurs. This failure event is found with G-MAST.

detecting by radar another vessel should determine if there is risk of collision and if so take avoiding action". When considering this rule, one can argue that the obstacle vessel is not operating at a safe speed. However, as rule 19 states, the milliAmpere should probably operate safer in these situations and thereby try to avoid the collision.

Figure 4.3c shows the last tracked SP-VP position of the obstacle vessel. In Table 3.1 in Section 3.1.2, we introduced the SP-VP-rate, which was set to 0.25. This means that the SP-VP-tracking system updates every 4 seconds. In the time between SP-VP updates, a lot can happen, as shown in this figure. The key observation in this failure event, is that the milliAmpere tracks the obstacle vessel to be further away than it is due to noise. Figure 4.3d shows the distance measurement data, where the last seconds of the figure show that the estimated distance is greater than the true distance. It is important to mention that the estimated distance is not the distance SP-VP system tracks the obstacle vessel at all times, as mentioned above, the SP-VP-tracker updates the position every 4 seconds.

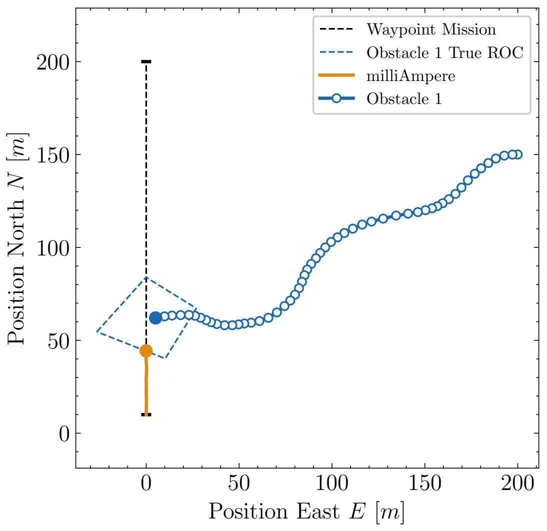


(a) Northeast plot.

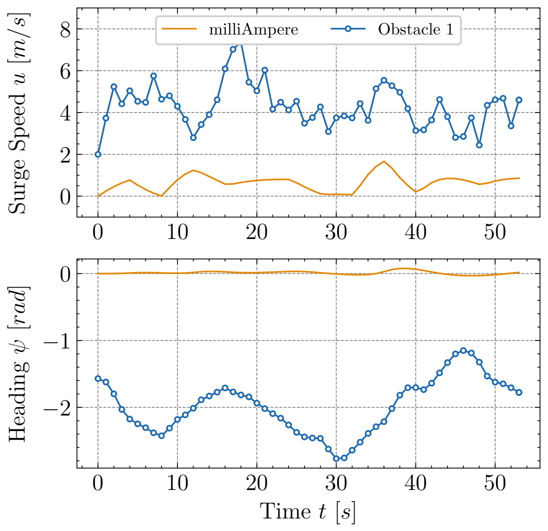


(b) Surge speed u and heading ψ.

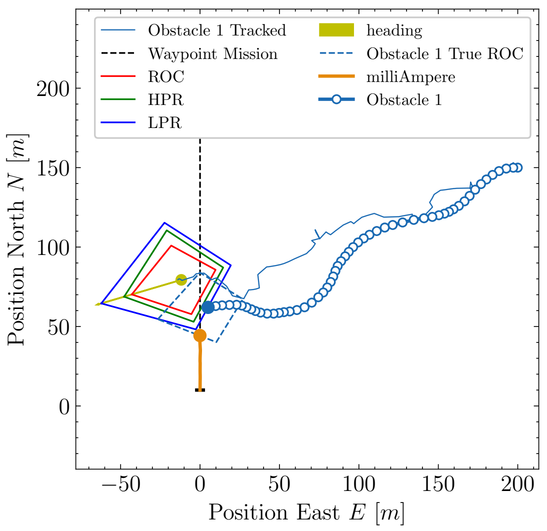
Figure 4.2: Bow-on Collision with Fast-moving Obstacle. The figure shows an example of a bow-on collision with a fast-moving obstacle. In Figure 4.2a it is shown how the obstacle vessel's trajectory changes right before the collision occurs at time t 60 seconds. The change in the obstacle vessel's heading ψ is shown clearly in Figure 4.2b where the heading changes almost from straight west to straight south towards the milliAmpere vessel at time t 53 seconds. Another interesting observation in this failure event is that the milliAmpere vessel has almost constant speed over 10 seconds before the collision occurs. The speed of the milliAmpere vessel and the obstacle vessel is 0.73 m/s and 4.59 m/s, respectively, when the collision occurs. This failure event is found with G-MAST.



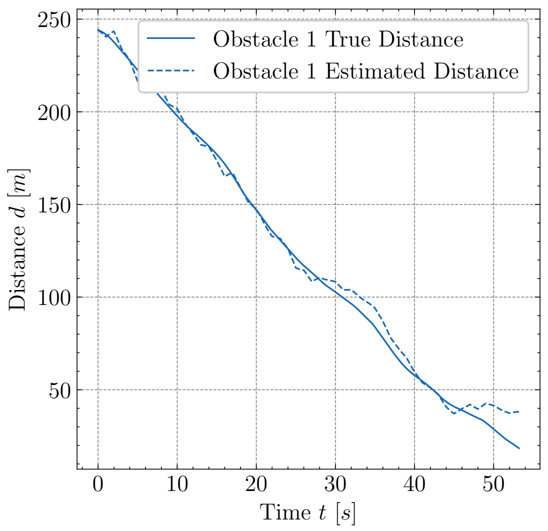
(a) Northeast plot.



(b) Surge speed u and heading ψ.



(c) Northeast plot with SP-VP data.



(d) Distance measure data.

Figure 4.3: Side Collision with Fast-moving Obstacle. The figure shows an example of a side collision with a fast-moving obstacle. Figure 4.3a shows that the milliAmpere vessel contributes to the crash at time t 53 seconds. Figure 4.3b shows that milliAmpere is speeding and that the heading ψ of the obstacle vessel is not changing much after time t 49 seconds. Figure 4.3c shows the northeast plot with SP-VP data, the plot contains the trajectory with tracked positions of the obstacle vessel and the last tracked obstacle position shown in yellow together with the SP-VP regions. The SP-VP regions shown is updated at time t 52 seconds. The key observation is that due to noise, it tracks the obstacle to be further away than it is. The milliAmpere vessel is close to the LPR, but it crashed into the obstacle vessel before intersecting with the region. Figure 4.3d shows the tracked distance of the obstacle vessel, referred to as estimated distance, together with the true distance to the vessel. This data shows more clearly that milliAmpere tracks the obstacle to be further away than it is. The failure event is found with G-MAST.

4.4 Slow-moving Obstacles

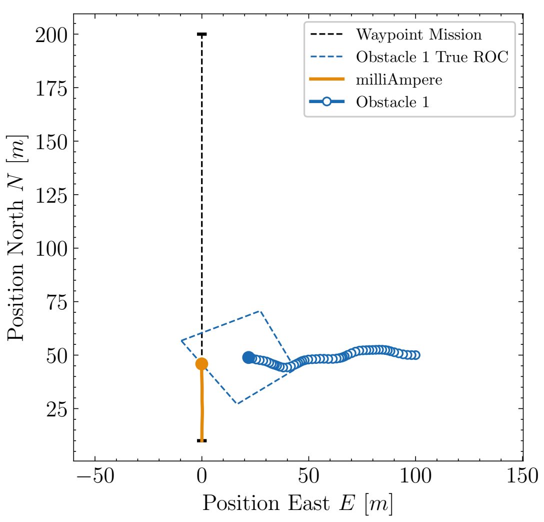
In Table 4.3, the configurations for the slow-moving obstacle vessel are presented. The main difference in this scenario setup is the initial position and velocity interval of the obstacle vessel. The initial North N position is changed from 150 meters to 50 meters, and East E is changed from 200 meters to 100 meters, which places the vessel closer to the milliAmpere vessel. The heading ψ is still straight west towards milliAmpere's waypoint mission. The surge velocity $u \in [-1.0, 3.0]$ is the key difference in this experiment, this is the reason why the vessel is referred to as slow-moving. The rest of the configurations are the same as the fast-moving obstacle scenario. The same hyperparameters as Table 4.2 are also used in this experiment. The failure events presented in this section are also found with G-MAST, but the same failure events are also found with B-MAST for the same scenario. The infeasible problem, introduced in Section 3.1.3, is also here handled by stopping the milliAmpere vessel.

Table 4.3: Slow-moving obstacle scenario

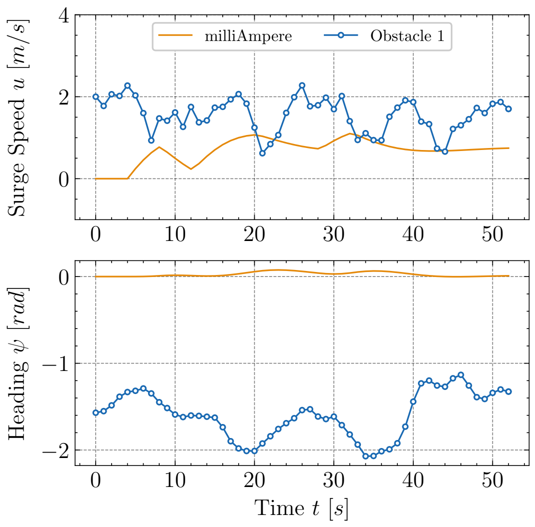
Obstacle Vessel 1		
Initial Positions and Velocitites		
North N	East E	Heading ψ
50	100	$-\pi/2$
Surge Velocity u	Sway Velocity v	Yaw Velocity r
2.0	0	0
Velocity and Gain		
Max Surge Velocity u_{max}	Min Surge Velocity u_{min}	Yaw Gain
3.0	-1.0	0.2

The failure events found with the slow-moving obstacle scenario will be presented in the rest of this section. First, a side collision with the slow-moving obstacle is shown in Figure 4.4. This failure event is similar to the one in Figure 4.3 found with the fast-moving obstacle scenario. However, this failure event shows that side collision also occurs with slow-moving obstacles.

Figure 4.5 shows another interesting failure event where a stern collision occurs with a slow-moving obstacle. In this failure event, we can see that the milliAmpere tries to speed away from the collision but is followed by the obstacle vessel. Figure 4.5b shows that the heading ψ of the obstacle vessel is changing towards the milliAmpere the last seconds before the collision occurs, and the surge speed u of the vessels shows that both vessels are speeding before the collision.

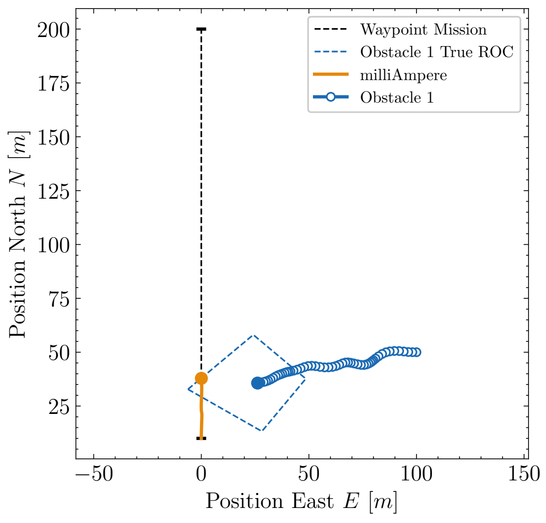


(a) Northeast plot.

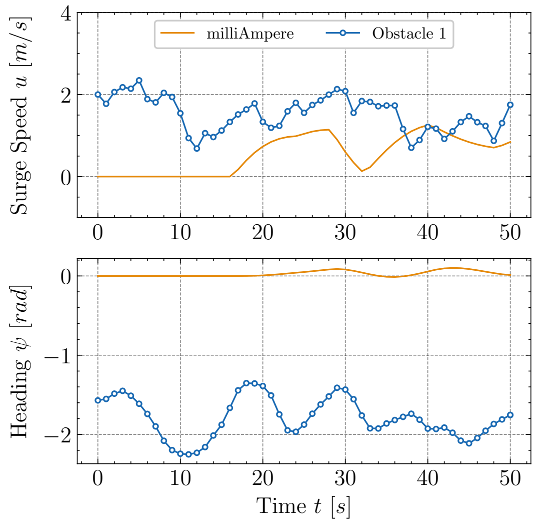


(b) Surge speed u and heading ψ.

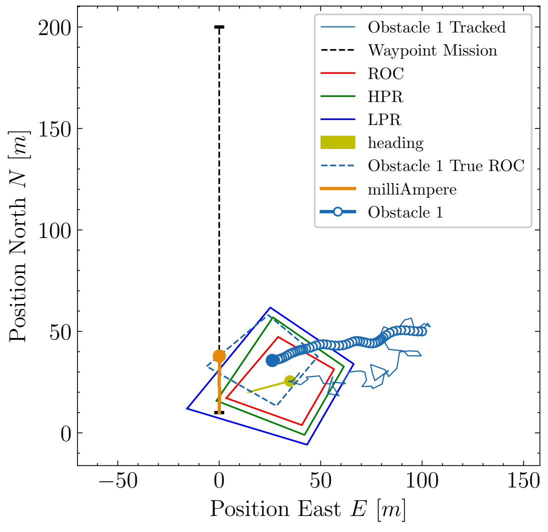
Figure 4.4: Side Collision with Slow-moving Obstacle. This figure shows a side collision with a slow-moving obstacle occurring at time t 52 seconds. Figure 4.4b shows that milliAmpere is speeding when the collision occurs and that the heading ψ is not changing drastically in the last seconds before the crash. This failure event is found with G-MAST.



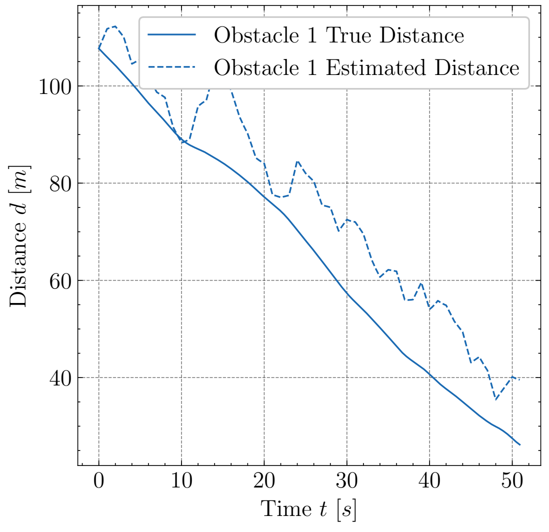
(a) Northeast plot.



(b) Surge speed u and heading ψ.



(c) Northeast plot with SP-VP data.



(d) Distance measure data.

Figure 4.5: Stern Collision with Slow-moving Obstacle. Figure 4.5a shows a stern collision with a slow-moving obstacle occurring at time t 50 seconds. Figure 4.5b shows that the obstacle is changing heading ψ towards the milliAmpere vessel in the last 5 seconds before the collision. The surge speed u of milliAmpere shows that milliAmpere tries to speed away from the collision but is followed by the obstacle. 4.5c shows the SP-VP data, here we can see that milliAmpere tracks the obstacle vessel to be further behind than it is. The SP-VP regions shown is updated at time t 48 seconds. Figure 4.5d shows the estimated distance together with the true distance of the obstacle vessel. This failure event is found with G-MAST.

4.5 Multiple Obstacles

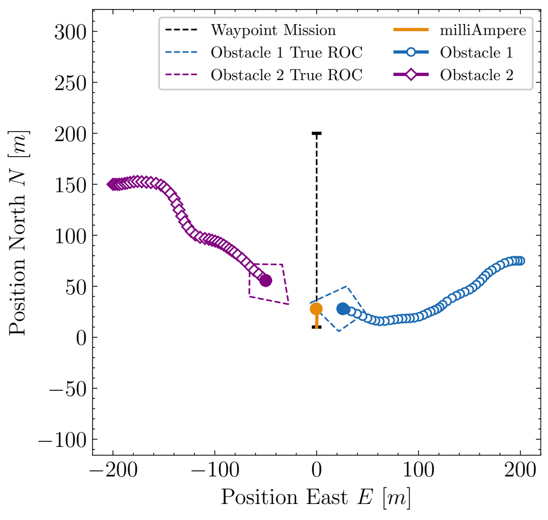
A scenario setup with multiple obstacles is used to investigate how the milliAmpere vessel behaves when exposed to multiple obstacles. Here, two obstacles are used, each with a designated obstacle scenario as described in table 4.4. The surge velocities for both obstacles were set to the same as the fast-moving obstacle scenario, $u \in [-1.0, 10.0]$. The main difference between the two obstacles is the initial positions, where obstacle vessel 1 is positioned 75 meters North and 200 meters East, and obstacle vessel 2 is positioned 150 meters North and -200 meters East. The same hyperparameters as Table 4.2 are also used in this experiment. The infeasible problem, introduced in Section 3.1.3, is also here handled by stopping the milliAmpere vessel.

Table 4.4: Multiple obstacles scenario

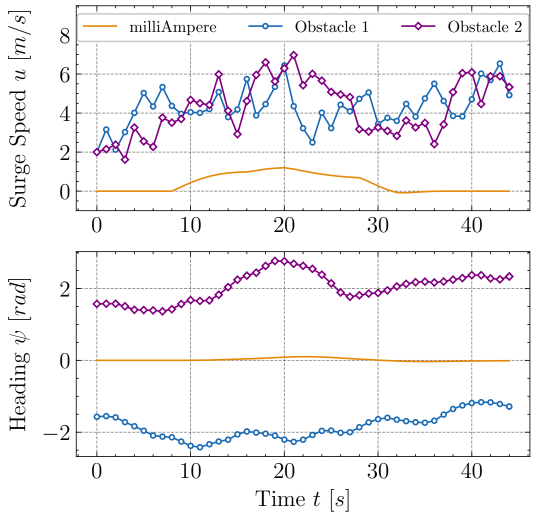
Obstacle Vessel 1		
Initial Positions and Velocitites		
North N	East E	Heading ψ
75	200	$-\pi/2$
Surge Velocity u	Sway Velocity v	Yaw Velocity r
2.0	0	0
Velocity and Gain		
Max Surge Velocity u_{max}	Min Surge Velocity u_{min}	Yaw Gain
10.0	-1.0	0.2

Obstacle Vessel 2		
Initial Positions and Velocitites		
North N	East E	Heading ψ
150	-200	$\pi/2$
Surge Velocity u	Sway Velocity v	Yaw Velocity r
2.0	0	0
Velocity and Gain		
Max Surge Velocity u_{max}	Min Surge Velocity u_{min}	Yaw Gain
10.0	-1.0	0.2

Figure 4.6 shows an example of a collision in the multiple obstacle scenario. The failure events that were found when multiple obstacles were used in the simulation, consisted of the milliAmpere vessel standing still a long time before the collision occurred. In these scenarios, the obstacles struggled to trick the SP-VP system.



(a) Northeast frame



(b) Surge speed u and heading ψ

Figure 4.6: Collision in the Multiple Obstacles Scenario. This figure shows an example of a collision occurring with multiple obstacles nearby at time t 44 seconds. Figure 4.6b shows that the milliAmpere vessel is standing still a very long time before the collision occurs.

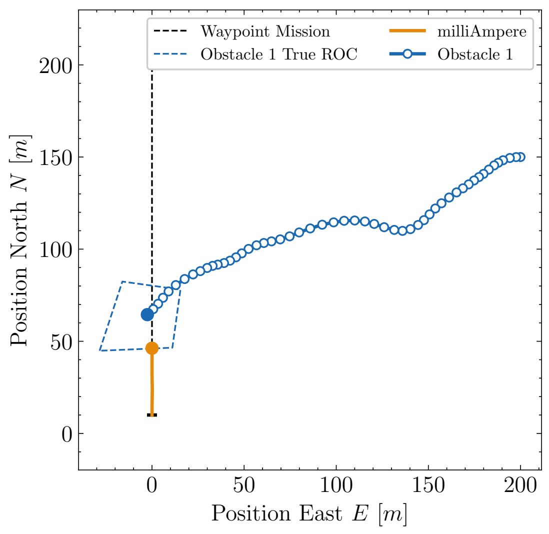
4.6 Infeasible Problem as Terminal State

In Section 3.1.3, the infeasible problem was introduced. To recap, the SP-VP method throws an infeasible problem exception in the provided simulator from Zeabuz if it cannot find a velocity profile that does not lead to a collision. This exception must be handled due to AST crashing otherwise. In the experiments conducted in this section, the infeasible problem is handled by setting it as a terminal state instead of just stopping the milliAmpere vessel, as done in the other experiments. The experiments aim to find more rare failure events in the SP-VP system. The experiment use the fast-moving obstacle scenario in Table 4.1 in Section 4.3. The hyperparameters for the MCTS-PW is also set to the same as Table 4.2 in this section.

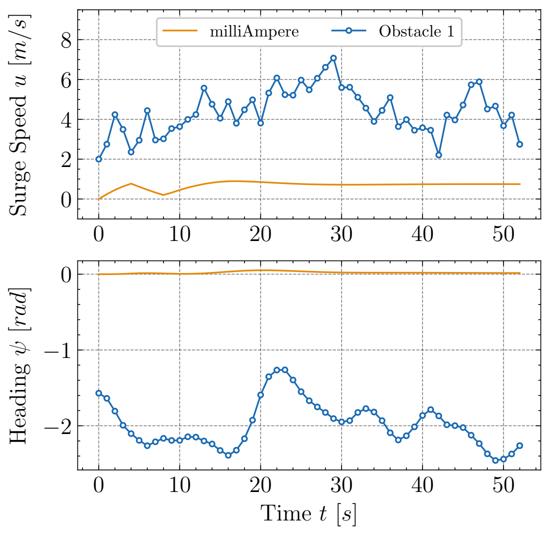
Figure 4.7 shows one of the most interesting failure events found when handling the infeasible problem as a terminal state. Figure 4.7b shows that the milliAmpere vessel has constant speed over 30 seconds before the collision occurs. This failure event is similar to the side collision with fast-moving obstacles in Figure 4.3d in Section 4.3. The milliAmpere vessel does not stop for the obstacle vessel on its starboard side, which might be a violation of rule 15 in COLREG. As discussed earlier in the similar example above, it depends on how the speed of the obstacle vessel is treated by rule 19 in COLREG.

Figure 4.8 shows a bow-on collision where the milliAmpere is speeding when the collision occurs at time t 62 seconds. In Figure 4.8c, we can see that the last updated SP-VP regions, which are updated at time t 60 seconds is not far away from the true position. However, the collision occurs right before intersecting with the LPR region of the obstacle.

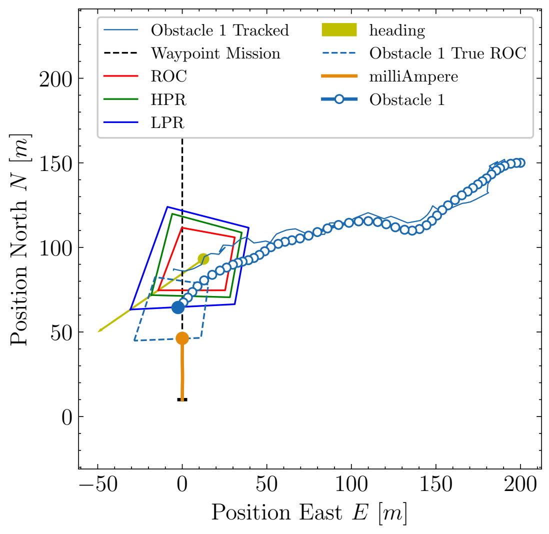
Figure 4.9 shows the number of failure events found when handling the infeasible problem as a terminal state versus handling it by stopping the milliAmpere vessel. It shows clearly that the failure events are more difficult to find when the infeasible problem is handled as a terminal state.



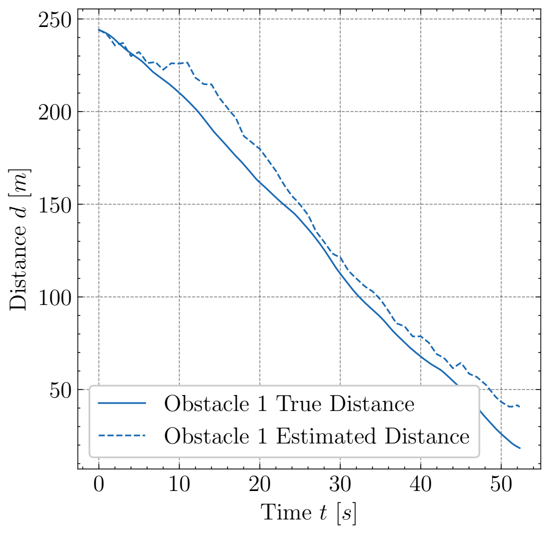
(a) Northeast plot.



(b) Surge speed u and heading ψ.

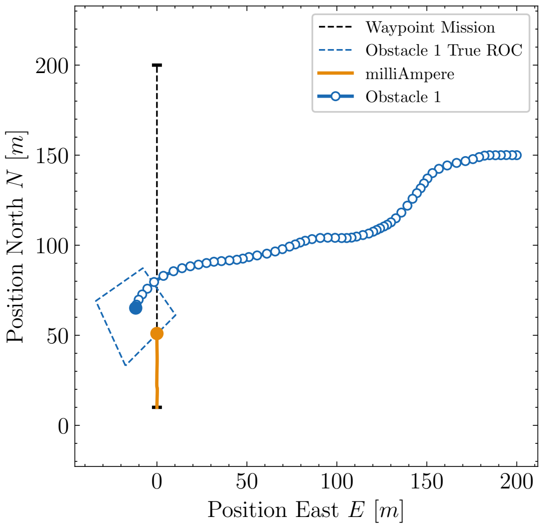


(c) Northeast plot with SP-VP data.

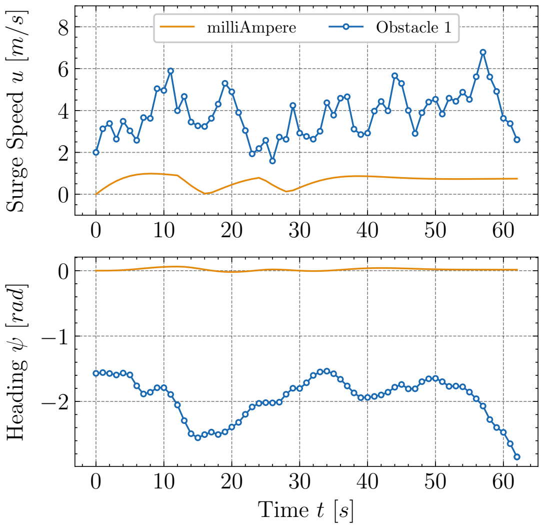


(d) Distance measure data.

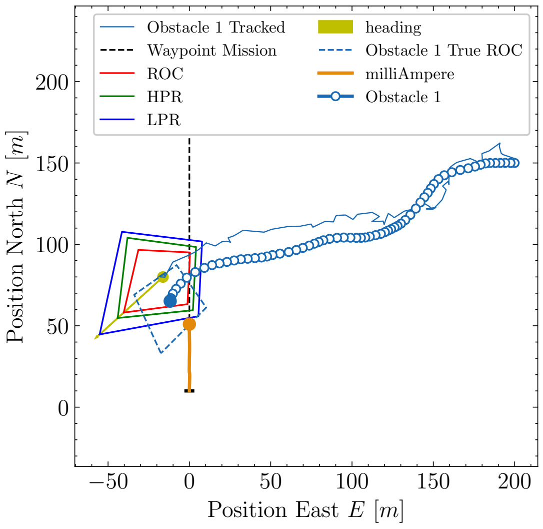
Figure 4.7: Side Collision when Handling Infeasible Problem as Terminal State. The figure shows a side collision with a fast-moving obstacle when the infeasible problem is handled as a terminal state. Figure 4.7a shows that the milliAmpere vessel crashes into the obstacle vessel at time t 52 seconds. Figure 4.7b shows that milliAmpere has constant speed over 30 seconds before the collision occurs. Figure 4.7c shows that the milliAmpere vessel tracks the obstacle vessel to be further away than it is. The SP-VP regions is updated at time t 48 seconds. Figure 4.7d shows the distance measure data of the SP-VP-tracker. This failure event is found with B-MAST.



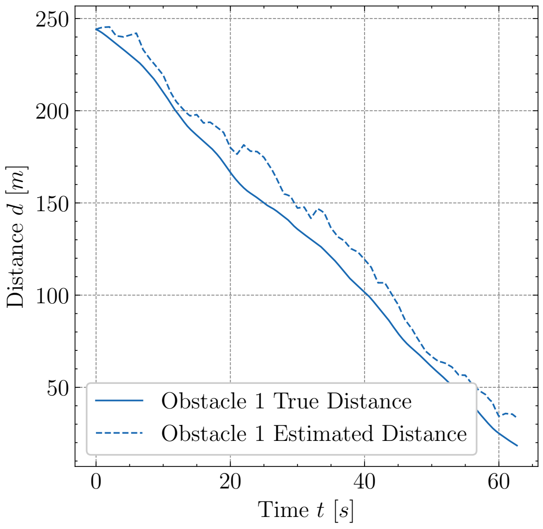
(a) Northeast plot.



(b) Surge speed u and heading ψ.



(c) Northeast plot with SP-VP data.



(d) Distance measure data.

Figure 4.8: Bow-on Collision when Handling Infeasible Problem as Terminal State. The figure shows a bow-on collision with a fast-moving obstacle when the infeasible problem is handled as a terminal state. Figure 4.8a shows that the milliAmpere vessel crashes into the obstacle vessel at time t 62 seconds. Figure 4.8b shows that milliAmpere has a constant speed over 20 seconds before the collision occurs and that the change in the obstacle vessel's heading ψ is the cause of the bow-on collision. Figure 4.8c shows that the milliAmpere vessel's SP-VP tracker does not track the obstacle vessel far from its true position in this case. The SP-VP regions is updated at time t 60 seconds. Figure 4.8d shows the distance measure data of the SP-VP-tracker. This failure event is found with B-MAST.

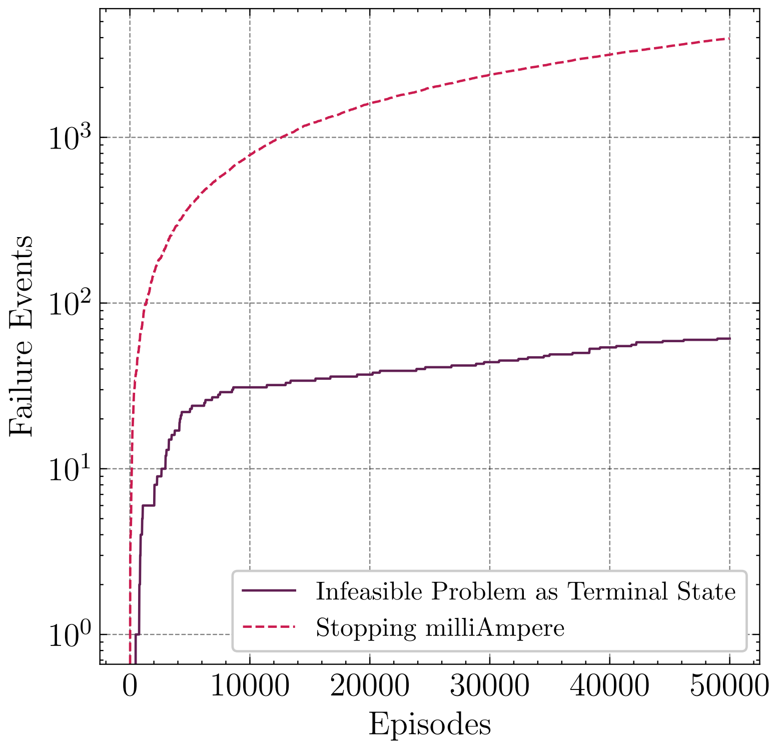


Figure 4.9: Handling the Infeasible Problem Results. This figure shows the number of failure events found over 50 000 episodes with B-MAST. When handling the infeasible problem as a terminal state, the B-MAST was able to find 61 failure events, compared to handling the infeasible problem by stopping the milliAmpere vessel, which found 3955 failure events. The simulations are using the fast-moving obstacle scenario in Table 4.1.

4.7 Data Interpretation with Soft-Dynamic Time Warping

Dynamic Time Warping (DTW) is useful to compare time series of variable size and to shift across time dimensions [15]. This can be computed by solving a minimal-cost alignment between the two time series using dynamic programming. A Soft-DTW can be used to compute the soft-minimum of all alignment costs [10]. This is a differentiable and less time-demanding extension of the DTW.

Hanna W. Hjelmeland used Soft-DTW to cluster her results from AST in her master's thesis [16]. This was a useful method to show clusters of obstacle trajectories that led to failure events, and it is also used in this section to interpret the results from MAST.

Figure 4.10 shows the resulting clusters from Soft-DTW, with data consisting of North N positions and East E positions of the obstacle vessel. Four clusters of obstacle trajectories leading to failure events found with G-MAST using the fast-moving obstacle scenario found in Table 4.1 in Section 4.3, are shown. The patterns of the different clusters might be recognizable with the three types of maritime collisions discussed in Section 3.7, side collisions, bow-on collisions, and stern collisions. However, all of the clusters seem to include all types of maritime collision but rather cluster the trajectories pattern a long time before the collision.

4.8 Rare Software Bug in the Simulator

During long runs with AST, the simulator crashed due to a software bug in the simulator. A software bug is here referred to as an error that crashed the whole simulator. The software bug appeared only now and then, once every 25 000 MCTS-iteration approximately. This is not a failure in the system found explicitly with AST, but rather a positive side effect of using MCTS as a solver in AST, due to it searching a huge part of the state space of the simulator. The reason why we are calling it a positive side effect is that these types of errors could potentially be dangerous if they go undetected. The error can be present in the system when deployed and might cause a dangerous situation when operating in the real world.

One experiment was conducted with the purpose of finding the situations where the software bug occurred. The way it was done, was by throwing an exception when the error occurred, then a failure event was defined as this exception in the AST. This led to the error being added to the priority queue and returned by the AST. The simulator object containing this error could then be extracted and analyzed further. This approach should not be confused with a normal search for finding failure events with AST. The reason is that neither the transition likelihood or the distance measure is adapted to find these rare software bugs easier, they are still implemented to find failure events in the domain of collision avoidance systems. However, the transition likelihood and the distance measure might help to find the error if it occurs due to the vessels behaving a certain way. In this case, a total random policy would make it harder to reproduce the error.

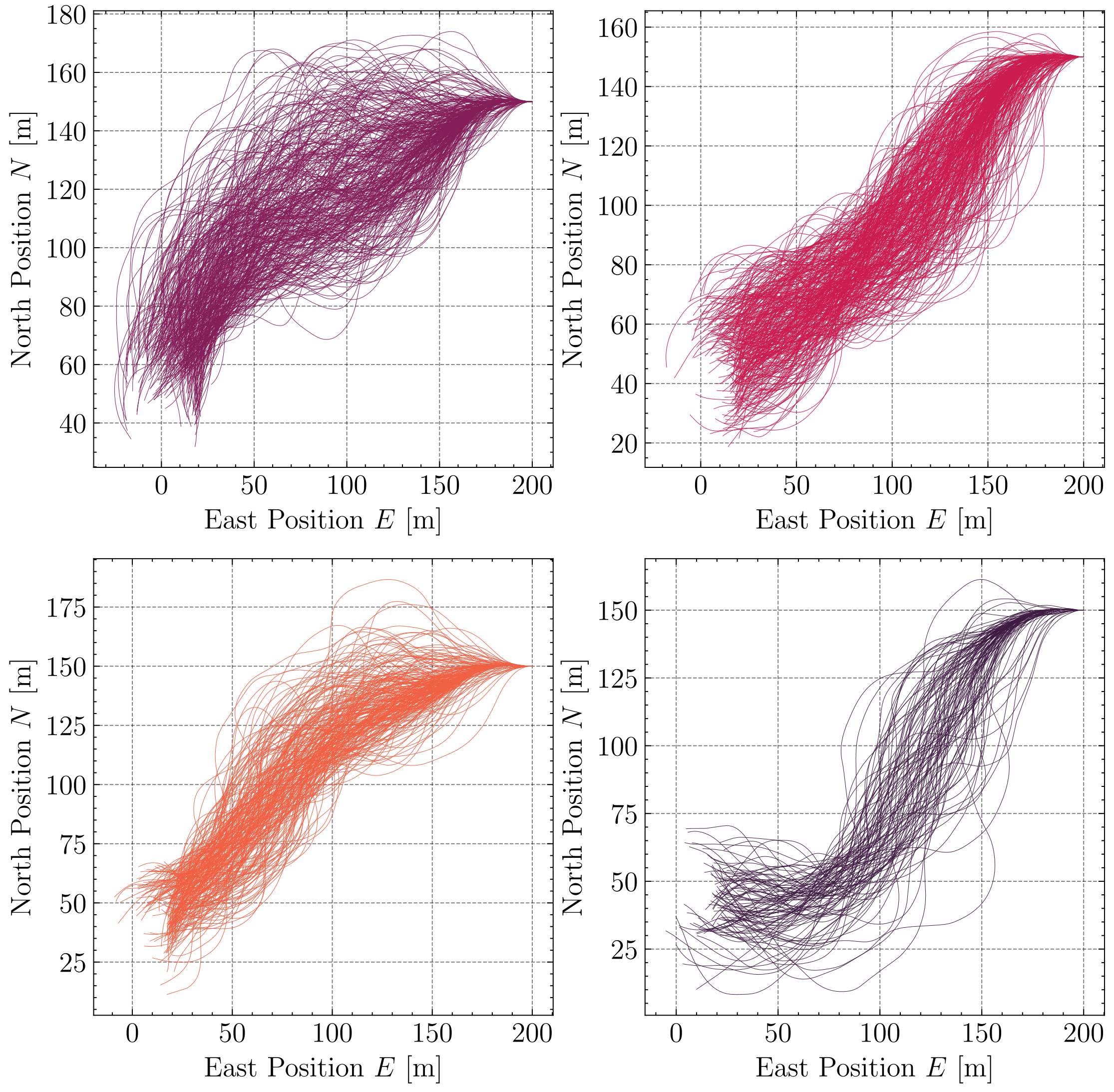


Figure 4.10: Clusters of Failure Events. This figure shows the clusters produced with Soft-DTW. At first glance, it may look like the trajectories fit with the three types of maritime collisions. A closer look shows that all of the trajectory clusters seem to include all of the collision types.

4.9 Failure Event Filtering

In the early stage of testing, AST only returned failure events where milliAmpere was standing still when the collision occurred. This was due to the setup not being optimal at the time. However, some investigation on how to find failure events where milliAmpere is speeding at the time of the collision is conducted. This led to a method referred to as failure event filtering in this thesis. This method is based on filtering out already found failure events, and searching for new failure events in the next AST run. Moreover, this can be done by eliminating these types of failures in the function which defines the failure event. Here, we are essentially filtering out these cases because we have already found them. This can be done for the next found failures, and we can then continue to make new failure event definitions relative to what we are looking for. The failure event filtering process is done manually between AST runs.

4.10 Black-Box vs Gray-Box Maritime Adaptive Stress Testing

In this section, the two variants of the MAST architecture will be discussed further, namely the G-MAST and B-MAST architectures. Our personal experience with both of these approaches will be discussed here. Much effort and time have been spent with the simulator itself during this thesis. With a starting point of having almost no domain knowledge about marine vessel dynamics and the maritime collision avoidance system, it was time-consuming work. As mentioned earlier in Section 3.1, the simulator provided was not ready to be used together with AST. The definition of what a failure event in the system is, and the need for implementing the option for the RL-agent to steer the obstacle vessels, were some of the challenges, to recap a few. Especially redefining the failure event definition in iterations led to the need to rerun the experiments. However, this experience led to a better understanding of the pros and cons of using the Black-Box and Gray-Box approaches.

Furthermore, the most fundamental reason why one should choose one approach over the other is when the simulator gives no choice to choose. If the simulator contains complex parts that are too difficult to understand, or the simulator does not reveal its internal variables or state, the Black-Box approach should be used. However, this does not mean that a Gray-Box approach should be used otherwise. The Black-Box approach gives the opportunity for the simulator designers to prepare everything for the one testing the system with AST, by making it only necessary to provide an action-seed to step the simulator. The only thing required by the simulator designers is to provide a boolean indicating if the simulator is in a failure state or not, the transition likelihood from one state to another, and preferable a distance measure to failure. If the tester is not the same person as the one designing the simulator, and the tester does not have enough domain knowledge, the Black-Box approach might be the most suitable.

However, the fact that the Black-Box simulator does not reveal its internals may lead to limited options to configure it. The failure event filtering discussed in Section 4.9, may be difficult to perform. It really depends on how isolated the internals of the simulator is. In some cases, the simulator can be provided as a Black-Box in terms of the design, but the code base is available to the tester. If that is the case, modifications can be made to the Black-Box approach as well.

In an early stage of this thesis, the noise to the SP-VP system was overlooked when testing. The Gray-Box approach was used, and the initial seed was set on the simulator, which led to the simulator behaving deterministically between states, with the same seed every simulation. The noise was not added as a disturbance nor added to the calculations of the transition likelihood. This could be avoided by providing the simulator as a Black-Box, then, the lack of domain knowledge might not lead to this mistake. Another problem with handling noise this way is that the simulator generates the same noise in each simulation, and the state space of different types of noise is excluded. This might lead to not finding important failure events in the system.

In Figure 4.11, we can see the performance of the Black-Box approach and the Gray-Box approach. The two types of architectures returned almost the same number of failure events.

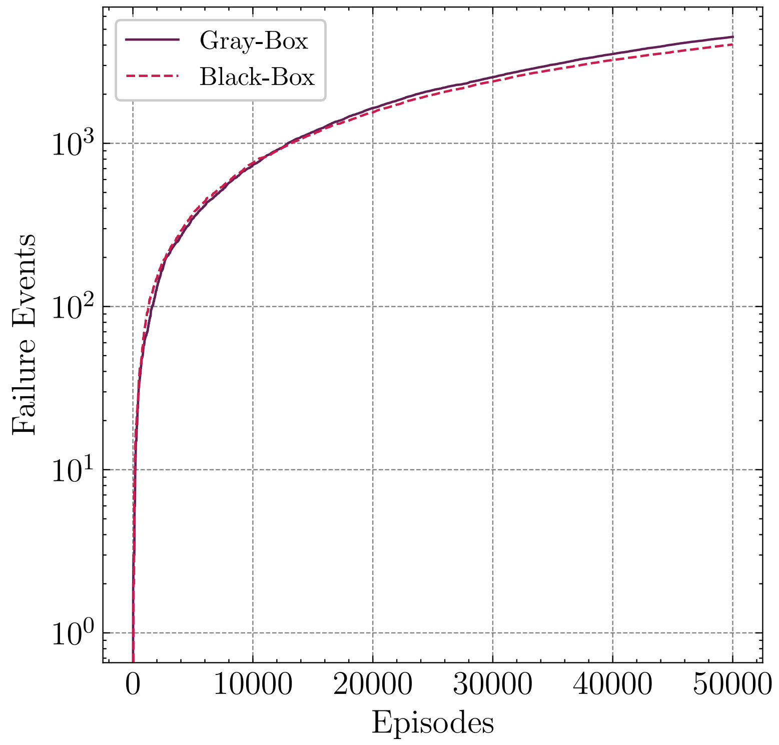


Figure 4.11: G-MAST and B-MAST Results. The figure shows the number of failure events found over the episodes. Black-Box found 4035 failure events, and Gray-Box found 4493. The Fast-moving obstacle scenario was used.

Chapter 5

Conclusions and Future Work

In this thesis, the MAST architecture is proposed. The architecture uses the framework AST for testing a maritime autonomous collision avoidance system, in particular the SP-VP method that is implemented in Zeabuz's milliAmpere passenger ferry. The MAST architecture is able to find interesting failure events in the system. In Section 1.2, the goal and research questions of this thesis were introduced, and they are repeated and answered here. First, the research questions will be answered, followed by a conclusion if the goal has been reached. Finally, future work will be discussed.

5.1 Conclusions of Goal and Research Questions

Research question 1 *How should we set up the scenarios for testing the collision avoidance system with AST in the maritime domain?*

Because the SP-VP method only plans a velocity profile for a predefined waypoint mission, and the fact that Brekke et al. [4], stated that it does not comply well with COLREG, scenarios were not set up to check for COLREG-compliance in this thesis. However, they were set up with rule 15 in mind, which states that a vessel that has another vessel on its starboard side should not conflict with the path of that vessel. However, for collision avoidance systems that are built to be more COLREG-compliant, the scenarios might be preferred to be set up to check for adherence to the rules in COLREG.

Research question 2 *Is Black-Box or Gray-Box simulators best suited when testing maritime autonomous collision avoidance systems with AST?*

Which type of simulator is best suited for testing maritime autonomous collision avoidance systems with AST really depends on how complex the different parts of the simulator are and how much domain knowledge the tester has. The Black-Box approach is preferred if the simulator is too complex to understand and if the tester does not have sufficient domain knowledge. Then the simulator should preferably be built by professionals in the field. Here with the requirements of returning the transition likelihood of states, a boolean indicating if a failure has occurred, and a distance measurement that tells how far away we are from a failure event. In addition to having the opportunity to set a simulator seed in each step of the simulator. The Gray-Box approach might be the best solution if the simulator does not consist of too complex parts, and the tester has sufficient domain knowledge.

Research question 3 *How useful are the failure events found with AST for safety validation of the maritime autonomous collision avoidance system SP-VP?*

In this thesis, several interesting failure events are found with AST. However, the obstacle vessels' behavior can be argued to be irrational due to their high speed in some of the scenarios and the fact that they are chasing the milliAmpere vessel. Moreover, the cause of the collision was often due to the SP-VP update rate combined with noise given to its tracker. Furthermore, we think that the failure events found in this thesis are useful enough to be analyzed further by the Zeabuz team for safety validation purposes.

Moreover, the search through the state space did lead to finding a rare failure in the simulator implementation itself. This failure occurred approximately every 25 000 steps of the simulator. This should be looked into as well, for the purpose of not having this rare error in the system when operating in the real world.

Research question 4 *How to interpret the failure events in the maritime autonomous collision avoidance system found with AST?*

The AST might return a high number of failure events, and it can be tedious to analyze each result manually. It might therefore be preferred to analyze the results with a clustering or classification algorithm, which supports time series data. In this thesis, Soft-DTW is used to cluster the obstacle vessels' trajectories to find patterns to classify them into groups of failure event types. However, the results returned the patterns in the trajectories long before the collision occurred and were, therefore, not easy to interpret.

Goal *Investigate if Adaptive Stress Testing (AST) is a suitable safety validation method for testing maritime autonomous collision avoidance.*

After having investigated the AST framework, our conclusion is that it seems suitable for testing maritime autonomous collision avoidance systems. However, to say this for sure, the method requires more testing of other types of collision avoidance systems as well. For the purpose of using AST for safety validation, the scenarios and the simulator might need more detailed setup and environment variables than provided in this work. For instance, environmental variables such as wind and flow in the sea could be added.

5.2 Future Work

In this section, the future work for testing maritime autonomous collision avoidance systems with AST in general and the SP-VP method specifically, will be proposed. First, the future work for the SP-VP method is proposed here. Second, we suggest testing more COLREG-compliant collision avoidance systems. Finally, comparing maritime collision avoidance systems with a regression testing approach of AST is proposed.

5.2.1 Future Work for Testing the Single Path Velocity Planner

In this thesis, the obstacle vessels have been using a simple first-order control system for motion control, it might be worth testing with a more realistic behaving motion control of the obstacle vessels. The RL-agent was also able to steer the obstacle vessels freely in work in this thesis, this lead to more unrealistic failure events in the system. Therefore, scenarios built with obstacle vessels crossing with constant speed and heading, not steered by the RL-agent, and by adding more environmental variables instead, might be interesting to test. An example of environmental variables to add could be snow variables which cause additional noise to the SP-VP system, and sea currents that affect the vessels' dynamics. The relevance of these extra variables should, of course, be evaluated before testing. Foggy weather might also be an interesting environmental variable to add.

5.2.2 Testing COLREG-compliant Collision Avoidance

Other types of maritime autonomous collision avoidance systems should also be tested for the purpose of approving the AST method as a suitable safety validation method for testing these systems. In particular, a COLREG-compliant collision avoidance system would be interesting to test. When setting up the scenarios for these systems, we believe that it is important to consider the rules in COLREG.

5.2.3 Differential Adaptive Stress Testing for Comparing Collision Avoidance Systems

In work done by Lee et al. [23], a regression testing approach of AST called Differential Adaptive Stress Testing (DAST) is proposed. It would be interesting to check if this method can also be used to evaluate what type of maritime collision avoidance system performs best in given scenarios. This idea could be used to compare the SP-VP method with, for example, the more reactive method first described by Thyri et al. [39].

Bibliography

[1] European Maritime Safety Agency. Marine casualties and incidents - preliminary annual overview of marine casualties and incidents 2014-2019. *EMSA*, Online:Available: http://www.emsa.europa.eu/tags/download/6132/3869/23.html, 2020.

[2] Bowen Alpern and Fred B Schneider. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987.

[3] Herman Berget. An area-time trajectory planning approach to collision avoidance for confined-water vessels. Master's thesis, NTNU, 2021.

[4] Edmund F Brekke, Egil Eide, Bjørn-Olav H Eriksen, Erik F Wilthil, Morten Breivik, Even Skjellaug, Øystein K Helgesen, Anastasios M Lekkas, Andreas B Martinsen, Emil H Thyri, et al. milliampere: An autonomous ferry prototype. In *Journal of Physics: Conference Series*, volume 2311, page 012029. IOP Publishing, 2022.

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[6] John Burkardt. The truncated normal distribution. *Department of Scientific Computing Website, Florida State University*, 1:35, 2014.

[7] Guillaume Maurice Jean-Bernard Chaslot Chaslot. *Monte-carlo tree search*, volume 24. Maastricht University, 2010.

[8] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, Roderick Bloem, et al. *Handbook of model checking*, volume 10. Springer, 2018.

[9] Anthony Corso, Robert Moss, Mark Koren, Ritchie Lee, and Mykel Kochenderfer. A survey of algorithms for black-box safety validation of cyber-physical systems. *Journal of Artificial Intelligence Research*, 72:377–428, 2021.

[10] Marco Cuturi and Mathieu Blondel. Soft-dtw: a differentiable loss function for time-series. In *International conference on machine learning*, pages 894–903. PMLR, 2017.

[11] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[12] Anthony William Fairbank Edwards. *Likelihood*. CUP Archive, 1984.

[13] Thor I Fossen. *Handbook of marine craft hydrodynamics and motion control*. John Wiley & Sons, 2011.

[14] Mohanad M Hammad, Ahmed K Elshenawy, and MI El Singaby. Trajectory following and stabilization control of fully actuated auv using inverse kinematics and self-tuning fuzzy pid. *PloS one*, 12(7):e0179611, 2017.

[15] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[16] Hanna Waage Hjelmeland. Safety validation of marine collision avoidance systems using adaptive stress testing. 2022.

[17] Kamal Kant and Steven W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The international journal of robotics research*, 5(3):72–89, 1986.

[18] Stephen J Kapurch. *NASA systems engineering handbook*. Diane Publishing, 2010.

[19] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[20] Mark Koren, Saud Alsaif, Ritchie Lee, and Mykel J Kochenderfer. Adaptive stress testing for autonomous vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–7. IEEE, 2018.

[21] Mark Koren, Anthony Corso, and Mykel J Kochenderfer. The adaptive stress testing formulation. *arXiv preprint arXiv:2004.04293*, 2020.

[22] Ritchie Lee, Mykel J Kochenderfer, Ole J Mengshoel, Guillaume P Brat, and Michael P Owen. Adaptive stress testing of airborne collision avoidance systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages 6C2–1. IEEE, 2015.

[23] Ritchie Lee, Ole Mengshoel, Anshu Saksena, Ryan Gardner, Daniel Genin, Jeffrey Brush, and Mykel J Kochenderfer. Differential adaptive stress testing of airborne collision avoidance systems. In *2018 AIAA Modeling and Simulation Technologies Conference*, page 1923, 2018.

[24] Ritchie Lee, Ole J Mengshoel, and Mykel J Kochenderfer. Adaptive stress testing of safety-critical systems. In *Safe, Autonomous and Intelligent Vehicles*, pages 77–95. Springer, 2019.

[25] Ritchie Lee, Ole J Mengshoel, Anshu Saksena, Ryan W Gardner, Daniel Genin, Joshua Silbermann, Michael Owen, and Mykel J Kochenderfer. Adaptive stress testing: Finding likely failure events with reinforcement learning. *Journal of Artificial Intelligence Research*, 69:1165–1201, 2020.

[26] Rory Lipkis, Ritchie Lee, Joshua Silbermann, and Tyler Young. Adaptive stress testing of collision avoidance systems for small uass with deep reinforcement learning. In *AIAA SCITECH 2022 Forum*, page 1854, 2022.

[27] Tomas Lozano-Perez. Spatial planning: A configuration space approach. In *Autonomous robot vehicles*, pages 259–271. Springer, 1990.

[28] Chengxing Lv, Haisheng Yu, Zhili Hua, Lei Li, and Jieru Chi. Speed and heading control of an unmanned surface vehicle based on state error pch principle. *Mathematical Problems in Engineering*, 2018, 2018.

[29] Tom M. Mitchell. *Machine Learning.* McGraw-Hill Science/Engineering/Math, 1997.

[30] Namireddy Praveen Reddy, Mehdi Karbalaye Zadeh, Christoph Alexander Thieme, Roger Skjetne, Asgeir Johan Sorensen, Svein Aanond Aanondsen, Morten Breivik, and Egil Eide. Zero-emission autonomous ferries for urban water transport: Cheaper, cleaner alternative to bridges and manned vessels. *IEEE Electrification Magazine*, 7(4):32–45, 2019.

[31] Guido Rizzi and Matteo Luca Ruggiero. *Relativity in rotating frames: relativistic physics in rotating reference frames.* Springer, 2004.

[32] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, USA, 3rd edition, 2009.

[33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[34] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[35] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[36] TheSocietyofNavalArchitectureandMarineEngineers SNAME. Nomenclature for treating the motion of a submerged body through a fluid. *The Society of Naval Architects and Marine Engineers, Technical and Research Bulletin*, pages 1–5, 1950.

[37] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[38] Marius Stian Tannum and Jon Herman Ulvensøen. Urban mobility at sea and on waterways in norway. In *Journal of Physics: Conference Series*, volume 1357, page 012018. IOP Publishing, 2019.

[39] Emil H Thyri, Erlend A Basso, Morten Breivik, Kristin Y Pettersen, Roger Skjetne, and Anastasios M Lekkas. Reactive collision avoidance for asvs based on control barrier functions. In *2020 IEEE Conference on Control Technology and Applications (CCTA)*, pages 380–387. IEEE, 2020.

[40] Emil H Thyri, Morten Breivik, and Anastasios M Lekkas. A path-velocity decomposition approach to collision avoidance for autonomous passenger ferries in confined waters. *IFAC-PapersOnLine*, 53(2):14628–14635, 2020.

[41] Emil Hjelseth Thyri. A path-velocity decomposition approach to collision avoidance for autonomous passenger ferries. Master's thesis, NTNU, 2019.

[42] Liuping Wang. Basics of pid control. 2020.

[43] Wei Wang, Tixiao Shan, Pietro Leoni, David Fernández-Gutiérrez, Drew Meyers, Carlo Ratti, and Daniela Rus. Roboat ii: A novel autonomous surface vessel for urban environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1740–1747. IEEE, 2020.

[44] Chelsea C White III and Douglas J White. Markov decision processes. *European Journal of Operational Research*, 39(1):1–16, 1989.

[45] David Yeh. Autonomous systems and the challenges in verification, validation, and test. *IEEE Design & Test*, 35(3):89–97, 2018.