Master's thesis

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

Johan Bakken Sørensen

Adaptive Stress Testing of Situational Awareness for an Autonomous Passenger Ferry

Master's thesis in Engineering and ICT
Supervisor: Øyvind Smogeli
Co-supervisor: Erik Wilthil and Børge Rokseth
June 2022

NTNU
Norwegian University of
Science and Technology

Johan Bakken Sørensen

Adaptive Stress Testing of Situational Awareness for an Autonomous Passenger Ferry

Master's thesis in Engineering and ICT
Supervisor: Øyvind Smogeli
Co-supervisor: Erik Wilthil and Børge Rokseth
June 2022

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

NTNU
Norwegian University of
Science and Technology

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

MASTER OF TECHNOLOGY THESIS DEFINITION (30 SP)

Name of the candidate:	Johan Bakken Sørensen
Field of study:	Marine Cybernetics
Thesis title (Norwegian):	Adaptiv Stress-testing av Situasjonsforståelsen til en Autonom Passasjerferge
Thesis title (English):	Adaptive Stress Testing of Situational Awareness for an Autonomous Passenger Ferry

Background

The Zeabuz autonomous mobility system for passenger transport is a complex, software intensive system subject to an unpredictable operating environment. This makes formal safety proofs practically impossible. Instead, one needs to resort to statistical considerations in the safety argumentation. In other words, there is a need to argue – based on the accumulated experience from testing, verification and validation activities – that the system is sufficiently safe. To solve these issues, a systematic and effective way of designing, running, and evaluating simulation scenarios that together give sufficient confidence in the safety is needed. The thesis shall employ, adapt and extend a method called adaptive stress testing (AST) to find likely failure events of the autonomy system with focus on the sitational awareness system using reinforcement learning (RL). The AST method has previously been used to validate aircraft collision avoidance systems, which is a similar problem.

Scope of work

- Review related litterature regarding AST and other existing methods for testing, verification and validation of autonomous systems.
- Discuss how the method can be deployed for testing of the situational awareness of an autonomous passenger ferry.
- Design and implement an AST system and connect this to a appropriate situational awareness test environment, including system under test and simulator.
- Perform initial simulations with the AST system and identify the method's potential for finding likely failure events in situational awareness.
- Compare and contrast to existing approaches, and discuss strengths and weaknesses based on the simulation results.

Specifications

The student shall at startup provide a maximum 2-page week plan of work for the entire project period, with main activities and milestones. This should be updated on a monthly basis in agreement with supervisor.

Every weekend throughout the project period, the candidate shall send a status email to the supervisor and co-advisors, providing two brief bulleted lists: 1) work done recent week, and 2) work planned to be done next week.

The scope of work may prove to be larger than initially anticipated. By the approval from the supervisor, described topics may be deleted or reduced in extent without consequences with regard to grading.

The candidate shall present personal contribution to the resolution of problems within the scope of work. Theories and conclusions should be based on mathematical derivations and logic reasoning identifying the steps in the deduction.

The report shall be organized in a logical structure to give a clear exposition of background, problem/research statement, design/method, analysis, and results. The text should be brief and to the point, with a clear language. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts., and it is not expected to be longer than 70 A4-pages, 100 B5-pages, from introduction to conclusion, unless otherwise agreed. It shall be written in English (preferably US) and contain the elements: Title page, project definition, preface (incl. description of help, resources, and internal and external factors that have affected the project process), acknowledgement, abstract, list of symbols and acronyms, table of contents, introduction (project background/motivation, objectives, scope and delimitations, and contributions), technical background and literature review, problem formulation or research question(s), method/design/development, results and analysis, conclusions with recommendations for further work, references, and optional appendices. Figures, tables, and equations shall be numerated. The contribution of the candidate shall be clearly and explicitly described, and material taken from other sources shall be clearly identified. Work from other sources shall be properly acknowledged using quotations and a

Harvard citation style (e.g. natbib Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct, which is taken very seriously by the university and will result in consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed.

The thesis shall be submitted with an electronic copy to the main supervisor and department according to NTNU administrative procedures. The final revised version of this thesis definition shall be included after the title page. Computer code, pictures, videos, data, etc., shall be included electronically with the report.

Start date: 15 January, 2022 **Due date:** 11 June 2022

Supervisor: Professor Øyvind Smogeli, NTNU/Zeabuz
Co-advisor(s): Dr. Erik Wilthil, Zeabuz
Associate Professor Børge Rokseth, NTNU

Trondheim 08.06.2022

Professor Øyvind Smogeli

Preface

This master thesis present research which is the result of individual work performed by the author in the period from January to June of 2022 at the Department of Marine Technology (IMT) at the Norwegian University of Science and Technology (NTNU). The work has been supervised by Professor Øyvind Smogeli (NTNU/Zeabuz), and the co-supervisors Dr. Erik Wilthil (Zeabuz) and Associate Professor Børge Rokseth (NTNU).

The project scope is defined in collaboration with Zeabuz and parts of the presented solutions originates from discussion with Zeabuz and the supervisors. The theoretical and practical work are solely done by the author. The master thesis is an extension of the project thesis delivered in December 2021. Hence, certain parts of this thesis are expanded or edited versions of the equivalent parts of the project thesis. In addition, it has to be mentioned that parts of the theory and discussion presented are also provided as a paper written together with Nicolai Brummenæs in the course TMR06 - Marine Autonomous Systems.

Acknowledgments

I am grateful for the support from my supervisor Øyvind Smogeli og co-supervisors Erik Wilthil and Børge Rokseth. I would also express my acknowledgement to Zeabuz for defining the project scope, discussions about the proposed method and technical support. Finally, I would like to thank the entire Adaptive Stress Testing research group at NTNU/Zeabuz and specifically Nicolai Brummenæs for the collaboration.

Abstract

Maritime Autonomous Surface Ships (MASS) introduces a new level of complexity to maritime control systems. Zeabuz is a start-up company from NTNU working with MASS, aiming to deploy autonomous urban passenger ferries as an alternative mean of transport. Deployment and development of such ferries require thorough testing, verification and validation in order to argue that the autonomy system is safe. Adaptive Stress Testing (AST) as proposed in this thesis is a method that can be applied to this challenge. The method uses reinforcement learning to search for failure events while maximizing the likelihood of the failure. Previously this has shown promising results in the automotive and aerospace industry.

The master thesis contributes to improved safety and validation for autonomous passenger ferries, such that it can be argued that the autonomy system is sufficiently safe. The thesis reviews literature regarding AST and other existing methods for testing, verification and validation of autonomous systems, comparing and contrasting the method with existing approaches. Furthermore, through a set of case studies the thesis presents a discussion on how AST can be deployed for testing of the situational awareness. In order to investigate the method's potential for finding likely failure events in situational awareness, firstly an AST system using the Monte Carlo Tree Search Method (MCTS) method is designed and implemented. Secondly, this AST system is connected to an appropriate test environment, consisting of a system under test and an environment simulator. The system under test is a situational awareness system. This system is furthermore an implementation of the probabilistic data association filter (PDAF) target tracking algorithm with the M/N extension for track existence. The environment simulator is an implementation of a multi-target tracking simulator. The complete test system, consisting of the AST implementation, the system under test and the environment simulator, is developed by the author. Initial simulations are performed with the test system, providing verification through qualitative analyses that AST is able to find common failure events in target tracking. The performance of the search using MCTS is further validated by running a Monte Carlo Search (MCS) in parallel. The results yield that the success rate of the MCTS algorithm is significantly higher than when running a random search using the MCS algorithm. In addition, a new control method using seed actions to control a single process of the simulator is proposed and tested. In previous applications seed actions have been used to control all the stochastic process in the simulator through the global random number generator. However, the new method creates a random number generator designated for the stochastic process of interest, making it possible to use open loop control to control the sampling of the process without requiring access to the simulator state. The proposed method performs better than the regular open loop global control method in some of the cases. However, for some other cases the method did not perform as well as the global control option. Hence, further research is required to determine the potential of the proposed control method. Generally, the AST method yields promising results for the situational awareness application. However, there are some aspects of the method that have to be further evaluated. These aspects are mainly related to evaluating the method's ability to find unforeseen failure events and how to design generalized test metrics for that purpose.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The objective of this master thesis project is to evaluate the use of a new machine learning based method, Adaptive Stress Testing (AST) in testing, verification and validation of Zeabuz´s autonomous urban passenger ferries. The verification and validation of AST as simulation-based test method for autonomous marine systems may provide a step forward in testing, verification and validation of complex safety critical and cyber-physical systems. In order to narrow the task, this project will mainly focus on testing of situational awareness. However, the method may be applied to other parts of the autonomy system as well.

1.1 Background

Historically, cities have been built next to rivers, lakes, bays, fjords and harbors due to the water being a beneficial arena for transportation. However, the use these waterways has been left largely unused. Instead infrastructure as bridges and tunnels has been built to cross them. This has been costly, non-scalable and inflexible solutions in addition to allowing combustion engine vehicles to pollute the air of the cities. This is something the startup company Zeabuz is aiming to change. Zeabuz´s objective is to develop and deploy autonomous urban passenger ferries. The ferries are set to be emission free and to be operating in urban waters in order to provide a new mode of efficient transportation across city waterways. However, the autonomy system of the ferries is composed of complex sub-systems driven by artificial intelligence, e.g. guidance, navigation and control systems. In addition is the urban environment is unstructured with possible dense traffic of other ships and objects that may cause collisions. Therefore, in order to provide safety assurance when deploying the autonomous ferries in an urban (or any) environment, it has to be argued based on testing, verification, validation and accumulated experience, that the autonomy system of these ferries are sufficiently safe.

The AST method presented for the first time in [3] is a method that may be used in such a testing process. By collaborative research between the Norwegian Open AI Lab, NTNU and Zeabuz, the goal is to study if AST may be used as state-of-the-art safety validation of autonomous marine systems, such as Zeabuz´s autonomy system. AST is developed by researchers at NASA and NASA´s research partners. It performs falsification, meaning it searches for error. The falsification problem is modelled as a sequential decision making problem, requiring a simulator that works as a Markov process with discrete time and continuous state. AST actively adapts the sampling of paths (of simulator states) using the machine learning method Reinforcement Learning (RL) in order to optimize for finding failure events and maximizing the path likelihood. The method can be applied to both simulators with complete access to the simulator state and to simulators where some or all of the simulator states are not accessible. In AST the system under test is considered a black box. Hence, little to zero knowledge about the system is required. However, in order to efficiently design the optimal reward function for the AST agent, domain knowledge is beneficial. This will become more clear later in this thesis.

1.2 Research Question and Scope of Work

The thesis reviews methods for testing, verification and validation of autonomous marine systems. The aim is to contribute with an answer the following research question:

> *How is it possible to accumulate enough experience regarding safety and validation for autonomous passenger ferries such that it can be argued that the autonomy system is sufficiently safe?*

This will be examined through the following more specific sub-questions:

1. What is the current state-of-the-art methods used in testing, verification and validation of autonomous systems, and how does AST fit into this context?
2. Can AST be used in order to perform state-of-the-art safety validation of the situational awareness in autonomous urban passenger ferries?

These questions will be addressed through the following thesis objectives:

- Provide an introduction to autonomy, presenting a high level overview of autonomous marine systems and a more detailed overview of the system architecture of an autonomous marine surface vessel.
- Present the assurance problem of autonomous systems, including a review of related work regarding methods for testing, verification and validation of autonomous systems.
- Provide necessary background theory regarding RL, and present the AST test method as a method working in a fully and partially observable environment.
- Provide a detailed description of situational awareness using the Probabilistic Data Association (PDA) target tracking algorithm, which is implemented and acts as system under test for this project.
- Present a detailed description of the environment simulator proposed and implemented for this project.
- Propose appropriate methods for AST to control the implemented environment simulator.
- Create a complete test system by implementing and integrating the AST test method with the implemented environment simulator and system under test.
- Present an overview of the scientific approach for designing a set of case studies with the objective of verification and validation of AST as a test method for autonomous systems, with the main focus on situational awareness.
- Provide an requirements formulation for a set of case studies, including the required technical setup, tuning strategy and control objective.
- Present a set of case case studies, including case study description and setup for performing comprehensive simulations with the objective for verification and validation of AST as a test method. In addition the results of the simulations shall be presented and discussed
- Provide a discussion of the case study results in context of the thesis research question.
- Evaluate and propose further work.

1.3 Research Method

The thesis applies quantitative and qualitative research methods in order to contribute with an answer to the thesis research question. This includes a review of relevant literature, proposing an appropriate test environment, defining case studies for evaluating the proposed test method with scientific approach, requirements formulation, theoretical analyses and numerical simulations The review of literature is perform with a qualitative approach. Theory is presented on autonomy, assurance, RL, AST, situational awareness using target tracking algorithms, target tracking simulator design and simulator control methods. The case studies are defined in order to provide a context for contribution with an answer to the research question. The case studies includes a set of qualitative analyses in addition to a quantitative analysis of the AST method.

1.4 Main Contributions

This master thesis project is contributing to the search of new methods for evaluating safety of autonomous marine systems, with emphasis on the situational awareness of autonomous surface vessels. In addition to reviewing related work and presenting relevant theory regarding autonomy systems, target tracking algorithms, RL and AST, the contributions include:

- Proposing and implementing a multi-target tracking simulator.
- Proposing and testing a new control option, open loop local control, for AST to manipulate simulations in order to find likely failure events.
- Implementing the PDAF target tracking algorithm with the M/N extension for tracking multiple targets.
- Implementing the AST method using the Monte Carlo Tree Search (MCTS) algorithm.
- Applying AST in the context of autonomous marine systems, state estimation and target tracking algorithms used in situational awareness.
- Discussing how the AST method can be deployed for testing of the situational awareness of an autonomous passenger ferry through a set of case studies.
- Using simulation results to verify and validate that AST can be used to find likely failures of target tracking algorithms used in the situational awareness of autonomous marine surface vessels.

1.5 Thesis Outline

The chapters of the thesis are structured as following:

Chapter 2 provides an introduction to autonomy systems and a description of the architecture of an autonomous marine surface vessel, such as the Zeabuz system. In addition, the assurance of autonomous systems problem is presented, including related work regarding testing, verification and validation of autonomous systems.

Chapter 3 presents the background theory regarding RL, which the AST method is based on. In addition the MCTS algorithm used by AST to find likely failures of the PDAF algorithm is presented, with an added modification of progressive widening to make the algorithm perform when the action space is large.

Chapter 4 provides the fundamental theory about the AST problem formulation. Two cases are discussed, AST in a fully observable environment and AST in a partially observable environment.

Chapter 5 presents first an in depth description of the PDAF target tracking algorithm acting as system under test. Second, a description of the simulator components of the proposed and implemented simulator is presented. Lastly, the different options for AST to control the simulator is presented, including a new proposed method for control.

Chapter 6 presents first the scientific approach in this thesis for evaluating AST as a test method. Secondly, a requirements formulation for the case studies performed is presented. Thirdly, the case studies including case study design, results and discussion are presented. Finally, the results are discussed in context of the thesis objectives.

Chapter 7 provides the conclusions of the project and proposes further work.

Chapter 2

Autonomy Systems

The objective of this chapter is to introduce some background theory about autonomy, provide a high level overview of an autonomy system similar to the Zeabuz system architecture and present the assurance of autonomous systems problem. This chapter is an extended and edited version of a chapter from the project thesis, [4].

2.1 Autonomy

The word autonomous is inherited from ancient Greece combining the words "auto", meaning self, and "nomos" meaning law according to [5]. Hence, an autonomous agent is a self governing agent. In this sense an autonomous agent can be anything that is self governing, e.g. an autonomous country or an autonomous vehicle. In the industrial setting autonomy systems are considered to be deliberative control systems designed to perform complex tasks in domains with high uncertainty. According to [6] an autonomous system should be able to plan and re-plan its mission subject to mission objective, risk exposure and any operational and environmental constraints that may happen as the operation goes. In addition, the system should be learning, adapting and improving based on accumulated experience. Autonomy systems are often confused with automation systems. Automation systems are significantly less complex, and are only set to perform well-defined tasks without human intervention. This is referred to as reactive control. When an automatic system might fail, an autonomous system can still perform due to their intelligent and adaptive functionality.

Autonomous systems are expected to produce tangible benefits to the society through increased sustainability and safety, reduced costs, and improved scalability and flexibility. Autonomy systems may enable new functions and markets, make it possible to operate in complex, harsh and remote environments, in addition to remove human error as a safety concern. Operationally it is expected that autonomy systems will lead to safer and less costly operations. Unmanned systems may also be smaller and cost less to build. The business sectors that are currently driving the development of advanced autonomy systems are aerospace, robotics, marine robotics, automotive and shipping industry.

2.2 Autonomy Systems - A High Level Perspective

High level of autonomy is achieved when reactive and deliberative control are combined. Figure 2.1 shows how marine autonomy systems can be divided into three layers, mission planning, guidance and optimization, and control execution. The mission planning layer uses deliberative control to define the mission objective and use inference to perform planning. Additionally re-planning is performed if necessary when receiving new information from other autonomy control layers. The guidance and optimization layer determines a desired path and a desired velocity, rotation

and acceleration along that path. This is referred to as guidance, [7]. The guidance has to be optimized with subject to the situational awareness and energy management of the system. The control execution layer performs pure reactive control in order for the system to act.

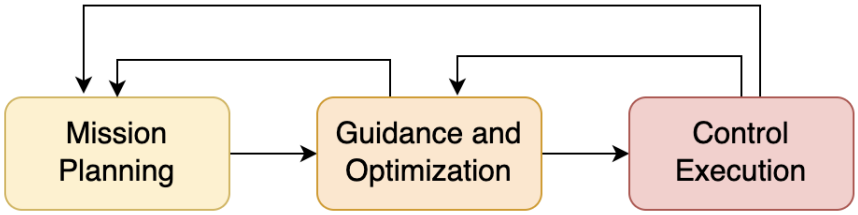


Figure 2.1: General Control System Architecture for an Autonomous Marine Vessel

Cyber-physical autonomous systems may be divided into four modules, reason and deliberate, sense, model, plan and act. The reasoning and deliberation module corresponds to the mission planning layer mentioned above. The sense, model and plan modules combined are equivalent to the guidance and optimization layer, and the act module is corresponding to the control execution layer. This is further specified for an autonomous marine surface vessel equivalent to the Zeabuz´s autonomy system in Figure 2.2.

2.3 Autonomous Surface Vessel Architecture

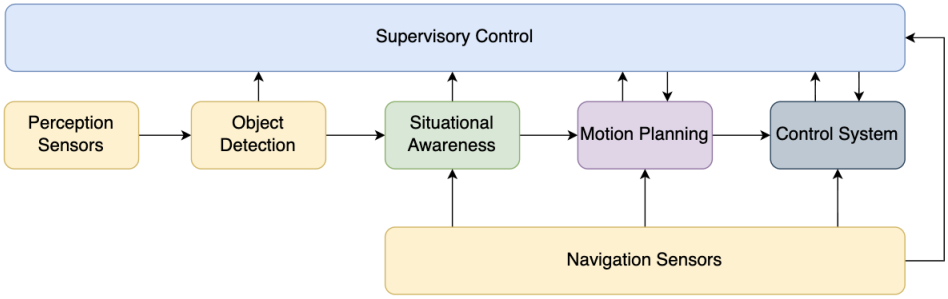


Figure 2.2: High Level Autonomy System Architecture for a Marine Surface Vessel

The modules in the Autonomous surface vessel presented in Figure 2.2 are as following:

Supervisory Control: The supervisory control system sets the mission objective, carries out planning and re-planning, performs online risk management, evaluates the integrity of the actuators and performs self diagnostics when necessary. This corresponds to the reasoning and deliberation module of the vessel.

Perception Sensors: These are sensors used by the situational awareness in order to keep track of the vessel´s surroundings. Typically, these sensors are radar, infrared (IR), images (RGB) and Lidar.

Object Detection: Combined with the perception sensors the object detection does make up the sense module for the surface vessel. The object detection analyzes all of the resolution cells of the sensor measurements in order to decide whether the measurement origins from a physical object or measurement clutter.

Situational Awareness: The situational awareness role is to keep track of the vessel´s surroundings. This is done using sensor fusion algorithms for target tracking and projection in order

to keep track of detected objects. This corresponds to the model component of the guidance and optimization layer described in the previous section.

Motion Planning: Based on the mission objective, and the situational awareness the motion planner does determine what path the vessel should follow and at what speed, rotation and acceleration the vessel should have along the path. This corresponds to guidance and is the plan module of the surface vessel.

Control System: The control system consist of the motion control and actuator control for the surface vessel. Based on the output of the motion planner, traditional sensors used for inertial navigation and online risk management done in the supervisory control, the system does carry out motion control which creates the control input for the actuator control which makes the system physically act. The control system corresponds to the act module for the surface vessel. In addition the actuator control system does provide feedback to the supervisory control in order to analyze the integrity of the actuators and perform self diagnostics.

Naviagtion Sensors: This is traditional sensors used for inertial navigation in marine control systems. These sensors could typically be Automatic Identification System (AIS), Global Navigation Satellite System (GNSS) and Inertial Measurement Unit (IMU).

2.4 Assurance of Autonomous Systems

The objective of this Section is to present the term assurance and discuss it in context of autonomous systems. Section 2.4.1 will give an introduction to assurance. The theory is gathered from lectures given in "TMR06 - Autonomous Marine Systems", which is a specialization course for master students at the Department of Marine Technology at NTNU. In addition, Section 2.4.2 presents related work regarding validation and verification, which are important concepts in assurance of complex systems.

2.4.1 Assurance

When deciding whether an autonomy system is ready for launch or not, assurance is a commonly used term. There exists different ways of defining assurance. [8] defines assurance in the following manner:

> "Grounds for justified confidence that a claim has been or will be achieved"

Whereas a claim could be about any quality which stakeholders value. This could be safety, sustainability, effectiveness or efficiency. Det Norske Veritas (DNV) provides a similar definition:

> "Grounds for justified confidence that systems, products and processes work safely, effectively and efficiently"

Considering these definitions it becomes clear that assurance consists of multiple dimensions. These dimensions could be safety, security, rules and regulations, ethics and societal impacts, environmental impacts and stakeholder expectations. Having this is mind, some challenges arises when considering assurance of autonomous systems. First, autonomous systems are complex systems made to handle unstructured environments and unpredictable scenarios, making the system less mechanical and deterministic compared to a traditional control system. Second, the control system of an autonomous vessel is a software intensive system consisting of a set of interacting components. It is important to mention that in such a system is safety an emergent system property, meaning that system failure may occur due to interactions between the system components, and not only through failure of a single component. This makes it harder to conclude that an autonomous system is safe, since it is hard to model the interactions between the components of the system and the system behaviour. Third, human operators are well trained in reacting to unpredictable

situations and may find creative solutions to problems quickly. Hence, a key problem when creating autonomous systems is to figure out how the system should act in unforeseen situations, and knowing that the system can handle the situation in a safe manner. With all these considerations in mind, building trust in a software intensive autonomous system may be a hard task to solve. The thesis will mainly focus on the safety aspect of the system. When considering the safety, there are especially three factors that has to be taken into account, technical safety, perceived safety and system performance. This will be further discussed in the next paragraphs.

Technical vs. Perceived Safety

Technical safety is related to how safe the technological system actually is, and is assessed through comprehensive testing and analysis to understand the behavior of the system in different situations. Whereas perceived safety takes into concern whether the system feels safe for the user. Reaching high level of technical safety is important to prevent losses. On the other hand, reaching high level of perceived safety is important to build public trust, and inspire to confidence in the system among all stakeholders. This dimension is especially important considering commercialization, since a potential customer will not use a system that the user does not perceive as safe. Hence, considering that achieving technical safety is a technical problem, covered by comprehensive testing, and perceived safety is a problem related to subjective experiences, achieving high perceived safety may be considered a harder problem to solve. The dilemma of technical safety vs. perceived safety is illustrated in Figure 2.3. It can be observed that Zeabuz aims to achieve both high technical and perceived safety. However, this thesis will have a technical focus, not taking perceived safety into the considerations.

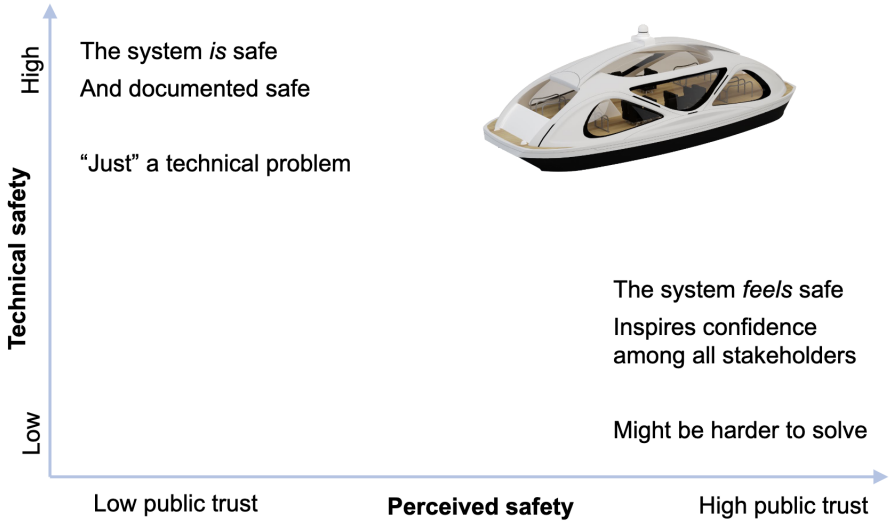


Figure 2.3: Illustration of the dilemma of technical vs. perceived safety. The figure was presented in the specialization course TMR06 at NTNU.

Balancing Safety vs. Performance

In addition to considering safety concerns regarding autonomous systems, it also important to take into account the performance dimension, such as high precision in target tracking. Depending on how a system is designed, it may score high on safety, but very low on performance, or vice versa. Both situations are unfortunate. In a worst case scenario a system may be unsafe and have poor performance. Usefulness may also be used as a term for explaining performance. Figure 2.4 presents different examples for how safety and performance have been balanced. The lower left corner shows a flamethrower drone, which would be considered as highly unsafe and with a low degree of usefulness. With increased safety, the cars imaged in the top left corner are autonomous vehicles that decides to stay parked in order to guarantee safety. Parked cars cannot be considered

useful. Hence, the cars scores low on performance. The bottom right corner shows a image of an autonomous car developed by a company called Uber. The imaged car was responsible for a deadly accident, where the car hit a pedestrian. In reports in the aftermath of the accident, it has come to light that the car was tuned for high performance, reducing it capabilities to act safely. This was done in order to avoid the car stopping all the time, by making it abstain from perceiving (for the most of the time) normal traffic situations as dangerous. Hence, finding a balance between safety and performance is a key problem with autonomous vehicles. It can be observed that Zeabuz aims to be in the top right corner, where the levels of safety and performance are high. This dilemma may be taken into further considerations when using AST to test algorithms for situational awareness using different tuning values.

Figure 2.4: Illustration of the dilemma of balancing safety and performance. The figure was presented in the specialization course TMR06 at NTNU.

2.4.2 Testing, Verfication and Validation

Assurance of autonomous systems is achieved by testing, validation , verification and certification of the system. This thesis will mainly focus on testing, verification and validation of complex control systems. Verification and validation are formulations that often are confused for each other. In order to avoid this the following definitions provided in [9] are used in this thesis:

Verification The process of evaluation software to determine whether the products of a given development phase satisfy the conditions imposed at the beginning of that phase.

Validation The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

Considering these definitions, the key difference between verification and validation is what phase of the development process the formulations covers. Verification focuses on determining if the product is being built right, whereas validation focuses on determining if the right product is being developed. Methods for verification and validation of complex embedded control systems are under development. The same methods may be customized for autonomous systems. See [10] for an overview of traditional and advanced modeling, testing, and verification techniques. In [1]

is Figure 2.5, based on a similar figure from [10], presented, providing an overview of existing methods for verification and validation of complex control systems, and where they are placed on the scales of exhaustiveness and scalability. These terms are described further in the next section. In addition is a red arrow added denoting where the AST method is positioned relative to the other methods. The AST method is as mentioned a simulation-based falsification method. Hence, the method is positioned in the same place as other falsification methods.

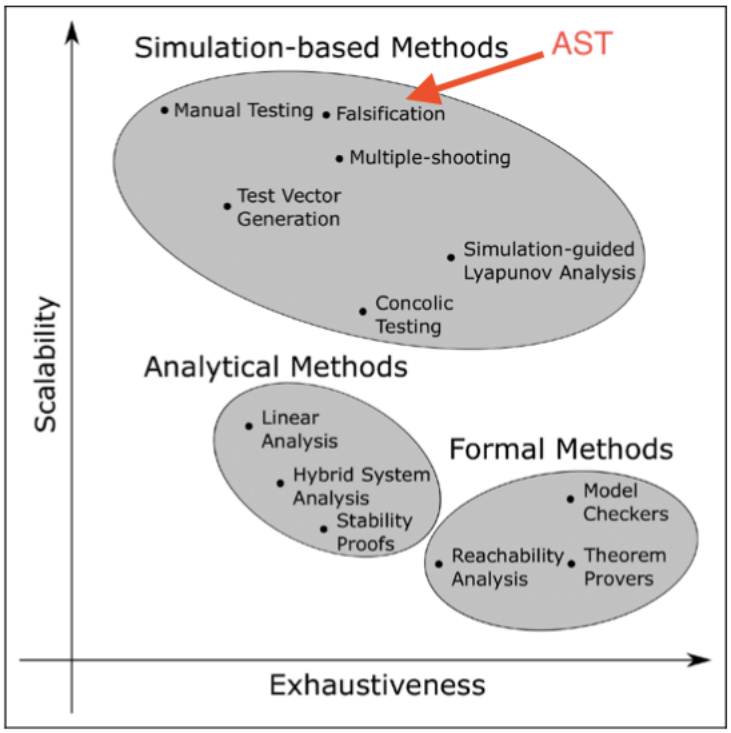


Figure 2.5: Existing methods in verification and validation of complex control systems. The x-axis shows the level of exhaustiveness and the y-axis shows the level of scalability. The red arrow denoting where the AST method is positioned on the plot. The figure is an edited version of the figure presented in [1].

It can be observed that testing, verification and validation of safety critical systems, such as Zeabuz´s autonomy system, can be separated into three broad categories. The first category is analytical methods, the second category is formal verification through formal safety proofs, and the third category is simulation-based testing. Traditionally the main focus has been on formal and simulation-based methods. Hence, these methods are main focus of the following sections.

Formal Methods

Formal verification constructs a mathematical model of the system and rigorously proves or exhaustively checks whether a safety property holds, [11]. In order for the system properties to be mathematically modeled, a formal framework for modeling has to be provided. In [12] Linear Temporal Logic was presented as a framework for formal logic in temporal computer programs. Probabilistic model checking (PMC) is a formal method used in verification. The method verifies system properties over stochastic models with discrete states, i.e. Markov Chains. Probabilistic Model Checking exhaustively evaluates properties over all states and paths with respect to probabilistic constraints. It has previously been used in verification of aircraft collision avoidance systems in [13]. The search space can be reduced by pruning infeasible paths as presented in [14]. Other formal methods as Automatic Theorem Proving (ATP) and Hybrid Systems Theorem Proving (HSTP) as presented in respectfully [15] and [16] have also been used in verification of safety critical systems before. Recently, related work regarding the use of formal methods in verification of

marine autonomous systems have been published. In [17] the marine traffic rules from the Convention on the International Regulations for Preventing Collisions at Sea (COLREGS) are formalized using temporal logic, laying the groundwork for applying formal methods in the context of collision avoidance in marine traffic. [18] investigates how Formal Methods (FMs) can be used to design and verify maritime control systems for safe and effective Maritime Autonomous Surface Ships. In [1] is research regarding automatic simulation-based testing of autonomous ships using Gaussian processes and temporal logic presented. However, a key drawback of formal methods such as PMC, ATP and HSTP is the exhaustiveness of the methods. In this context exhaustiveness means providing verification by exhausting all possibilities. This means that every possible combination of states in the system has to be evaluated through brute force. Exhaustiveness may be beneficial if the system complexity is low, but large exhaustiveness implies low scalability. Hence, the formal methods are placed in the Figure 2.5 where the Exhaustiveness is high and the scalability is low. In this context scalability refers to real systems consisting of different interconnected sub-control systems configured for the particular application, e.g. a ship. In the case of autonomous systems where the search space is vast and complex are formal safety proofs practically impossible. However, formal proofs may be used in some sub-parts of the system where the search space is smaller.

Simulation-Based Methods

Since formal verification is not feasible in the case of autonomous systems, statistical considerations have to be made in the safety argumentation. This is demonstrated well by Figure 2.6 presenting the concept of "the long tail of the probability distribution". The key idea behind this concept is that through performing operations with a system over a longer time period, the risk of an specific event occurring is decreased due to the fact that if an event will happen, then it will most likely happen at some time step of that period of time. Hence, if an autonomous passenger ferry has been in operation for billions of hours and a specific event hasn't occurred, then the risk of that event occurring is mitigated through experience. However, the risk for that event does still exist. This remaining risk is called residual risk, and is the risk represented by the long tail of the probability distribution. In this case, simulation-based testing is necessary to acquire enough data through many operational hours, such that safety is statistically demonstrated by moving the residual risk further to the right. Physical testing is not considered, since it would require a wast amount financing in order to keep the system in operation for long enough time. In addition, it wouldn't be possible to have the system in operation before it is approved to be so, which requires that it is tested, verified, validated and certificated. Simulations does also have the benefit of running scenarios representing hours of operation in just a few seconds, making the time frame for achieving low residual risk significantly shorter.

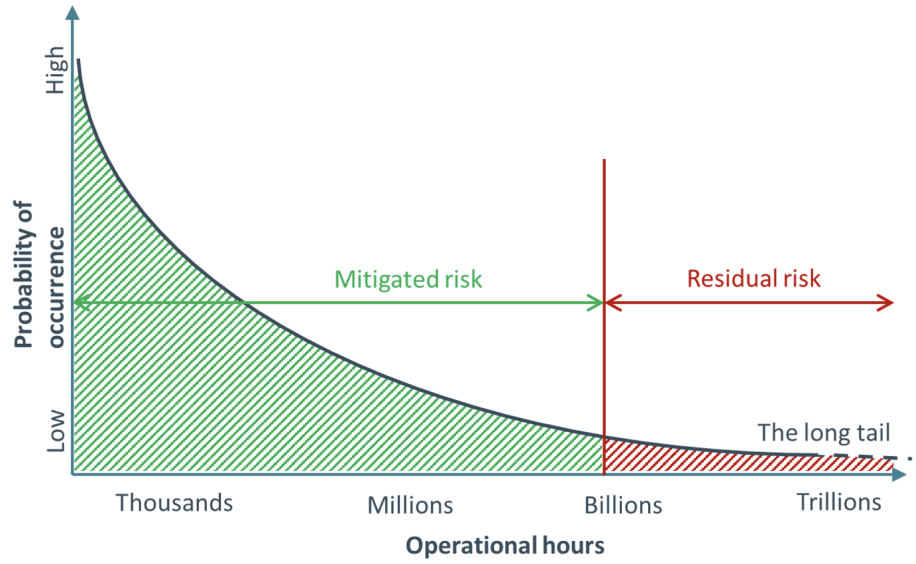
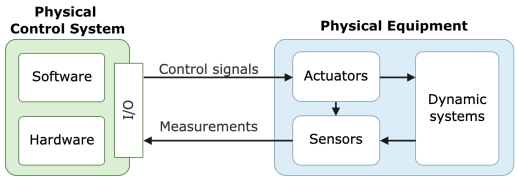
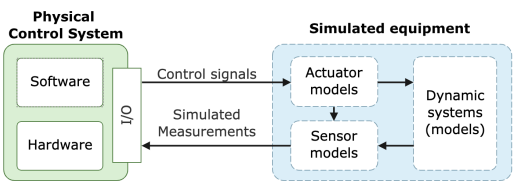


Figure 2.6: Illustration of the concept of "the long tail of the probability distribution". The x-axis represents the probability of an event occurring, and the y-axis represents the number of hours a system is in operation. The figure was presented in the specialization course TMR06 at NTNU.

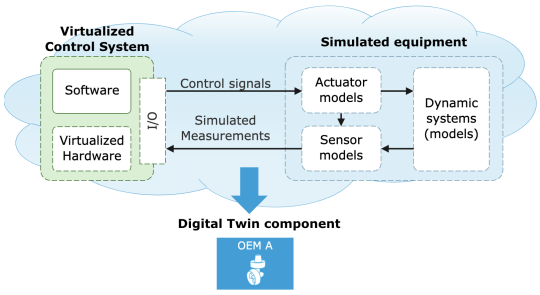
Simulation-based testing refers usually to the usage of a Digital Twin (DT) of a cyber-physical system with its operating environment. In-the-Loop testing are well know previously used approaches for simulation based testing. This includes Hardware-In-the-Loop (HIL), Software-In-the-Loop (SIL), Model-In-the-Loop (MIL) and Process-In-the-Loop testing. HIL, SIL and MIL testing are illustrated in Figure 2.7.



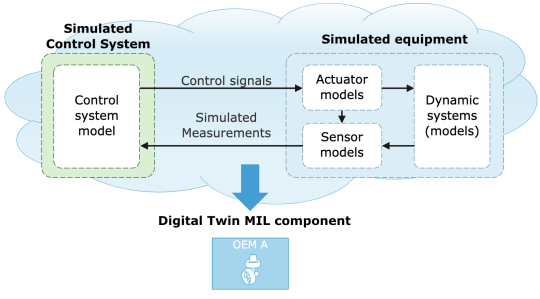
(a) Real System



(b) HIL



(c) SIL



(d) MIL

Figure 2.7: Illustration of In-the-Loop-Testing. The figure consists of sub-figures that were presented in the specialization course TMR06 at NTNU.

HIL testing is carried out by connecting simulated equipment to the actual physical control system consisting of hardware and software. The simulated equipment provides a simulated measurement to the physical control system, which produces control signals that are fed back to the simulator. In [19] a clarification of what HIL testing is and how independent third party HIL testing can be applied to safety critical control system software on drilling ships and rigs are presented. In addition

the reasons why third party HIL testing is an important contribution to technical safety, reliability and profitability of offshore operations are presented. [20] present an overview of experiences from HIL testing of dynamic positioning systems and power management systems. SIL testing works in the same fashion as HIL, but the physical control system is replaced by a virtualized control system, where the hardware is virtualized. SIL testing is equivalent to performing tests on a DT. MIL testing is quite similar to SIL testing. The difference is that instead of having to simulate the hardware of the physical control system, the entire control system is simulated as a whole using a mathematical model. MIL testing is equivalent to testing a DT where the control system behavior is mathematically modeled.

When working with In-the-Loop testing the objective is typically to design simulations scenarios that are set to be as demanding as possible for the the control system under test. Then, the objective is to make the system fail. This is known as falsification. Previously, falsification has been done manually or by sampling simulator states. However, sampling does not optimize for failures, and therefore it may take a vast number of simulations before a sequence of sampled states leads to system failure. The falsification tool S-TaLiRo presented in [21] deals with this by applying optimization algorithms such as simulated annealing and the Nelder-Mead method. The problem of such algorithms is that they does not take into account the temporal relationship between the variables that are to be optimized. Hence, global optimization algorithms struggles with finding the sequence of events that lead to failure events. Considering this, [22] presents another approach using rapidly-exploring random trees (RRT) to optimize the problem when it is considered a path planning problem. However, RRT struggles when the parameters in the state space are of mixed types or scales, since it struggles to choose a good distance metric. In addition the RRT method requires full access to the simulator state, which is not provided in all simulators.

Adaptive Stress Testing - Previous Work AST is a relatively new method, and has yet to be applied to marine systems. However, in other industries the method has demonstrated promising results. This is especially the case of the aerospace industry and to some degree the automotive industry. In [3] the method is applied in stress testing of air collision avoidance systems using the Monte Carlo Tree Search (MCTS) algorithm to identify flight trajectories that are likely to lead to near mid-air collisions (NMAC). The AST algorithm using the MCTS algorithm manages to find failure scenarios more efficiently than a regular MCTS algorithm. Introducing new software systems may add new failure modes that older robustly tested systems does not have. Then, it would be beneficial to stress test the new and the old system against each other in order to find failures in the new system. Differential AST (DAST) is an extension of AST used to find failures in a system relative to a baseline system. This was done in [23], where the next generation Airborne Collision Avoidance System (ACAS X) was tested against the system it is set to replace, Traffic Alert and Collision Avoidance System (TCAS). The DAST algorithm was successfully able to find a set of interesting NMACs. AST has also been applied in testing of autonomous vehicles. [24] presents MCTS and Deep Reinforcement Learning (DRL) AST solvers applied to a scenario where an autonomous vehicle approaches a cross walk. The DRL solver was able to find more likely failure scenarios than the MCTS solver with fewer simulator calls. However, this method did have its drawbacks. The problem was that the DRL solver used a feed-forward neural network with a discretized space of possible initial conditions. The system was not treated as a black box, since it is required to analyze the internal state of the system, leading to considerable implementation complexities. Secondly, a new instance of the solver were needed to be run for each initial condition, increasing the computational complexity and disregarding the underlying relationship between similar initial conditions. Both problems was solved in [25] by implementing a recurrent neural network taking as input a set of initial conditions from a continuous space. The approach enabled robust and efficient failure detection since the solution was able to generalize across the entire space of initial conditions. The solver demonstrated to yield solutions to problems that previously was considered intractable. AST has also been applied to testing and validation of image-based neural network controllers in [26], trajectory planning systems in [27], trajectory predictions in flight management systems in [28] and in a financial environment testing fraud detection systems in [29]. Generally, the application of AST seems to be quite flexible, being able to find likely failure events in different systems acting in different environments.

Chapter 3

Reinforcement Learning

The objective of this chapter is to provide an introduction to Reinforcement Learning (RL) and present the algorithm that will act as solver in the AST system. The theory presented is an edited version of the equivalent chapter in the project thesis, [4]. First, some background theory regarding sequential decision making problems and RL from Chapter 17 and 21 in [30] will be presented. Then, the Monte Carlo Tree Search algorithm described in [31] will be presented.

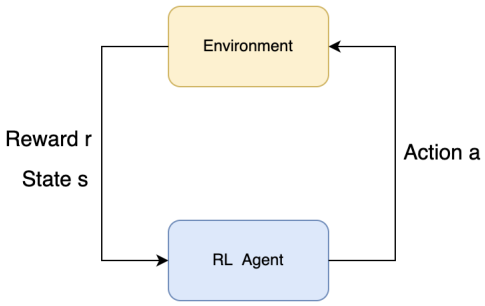


Figure 3.1: Main idea of RL methods.

RL is a branch of Machine Learning (ML) that don't fit in the typical taxonomy of ML, where all methods are divided into two main categories, supervised and unsupervised learning. Supervised and unsupervised learning algorithms do typically require large amounts of data. Hence, they are algorithms that learns aposteriori of gathering data. In addition they do not make actions in the environment in order to gather the data. RL methods are designed to learn while acting with the environment. This is well described in Figure 3.1. The RL agent takes some action a, which affects the environment, the agent observes the state s of the environment and gather some reward r for the action induced change of the environment state. This carries on as the agent tries new actions, observing the environment and gathering rewards. The most simple example to explain this, is thinking about how a puppy learns from interacting with the world. When training puppies to become obedient, RL is a common approach. E.g. giving a puppy a treat (reward) every time the puppy does something good, and punishing the puppy with not giving treats when it behaves badly. Doing this over time the puppy starts to understand what actions leads to treats and what actions does not lead to treats, and by that learns how to behave in the world. In this case the puppy is the RL agent and the rewards are treats. The next sections of this chapter will provide an introduction to how sequential decision making problems are mathematically formulated, and how RL can be used to solve such problems. This includes presenting a mathematical framework for sequential decision making problems, defining what a policy is, explaining the difference between model-free vs. model-based methods and taking a look at the dilemma of exploration vs. exploitation.

3.1 Background Theory - Sequential Decision Making

Consider the problem where the objective is to create a plan for how to reach a goal state from some initial state. For each state in-between the initial state and the goal state, including the initial state, there are some actions available. There is uncertainty associated with each action, meaning that applying an action in a state is not predictably tied to reaching another state. This is a sequential decision making problem, and can be mathematically modeled as a Markov Decision Process (MDP).

3.1.1 Markov Decision Process

In order to mathematically define a MDP it is assumed that the state s is always known. This is the case of a fully observable environment. Knowing this it is possible to model a sequential decision making problem as a MDP using the following 5-tuple of variables,

- S: $S = [s_0, s_1, ...]$, which is the set of all states
- A: $A(s) = [a_0, a_1, ...]$, which is all available actions in state s
- T: $T(s, a) = p(s\prime|s, a)$, which is the probability of reaching state $s\prime$ from state s doing action a. This is called the Transition Model.
- R: $R(s, a)$, which is a reward associated to reaching state s when executing action a. This is called the reward function.
- γ: $\gamma \in [0, 1]$, which is a discount factor to deal with the fact that an immediate reward is considered better than a later reward, and to deal with the concern regarding infinite sums of rewards when dealing with a infinite horizon.

3.1.2 Policy

The solution of an MDP is called a policy. Policy is defined as an contingency plan for what action to do in each state. A policy is denoted by π. The following expressions should give a clear picture of how a policy works:

$$\pi = \{s_0 \mapsto a_0 \mapsto s_1 \mapsto a_1 .... \mapsto s_{goal}\} \tag{3.1}$$

Where the agent starting in state s_o performs som action a_o according to the policy, and the state transitions to s_1 where the agent executes action a_1 according to the policy. This carries on until the agent reaches the goal state s_{goal}. The optimal solution of an MDP is the one that yields an optimal policy, which is the policy that has the highest expected utility. The optimal policy for an MDP is denoted by π^* and the utility of a policy is defined as the discounted sum of received rewards,

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right] \tag{3.2}$$

Then, the optimal policy starting in state s is given as the policy which maximizes the utility in state s:

$$\pi_s^* = \arg\max_{\pi} \left(U^{\pi}(s)\right) \tag{3.3}$$

Then, the optimal action in state s is given by:

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} p(s'|s,a) U^\pi(s') \tag{3.4}$$

3.1.3 Model-Free vs. Model-Based Methods

There are two categories of methods that can be used in the search of an optimal policy. One category is the model-based solution, which is used when the transition model and the rewards are known a priori. Then, the solution can be found using dynamic programming (DP) algorithms, such as Value Iteration or Policy Iteration. However, the transition model and the rewards are not always known. Then, a model-free solution has to be found. This can be done using indirect RL where the agent indirectly solves the problem by systematic sampling of the environment creating an estimate of the transition model and the rewards, before applying DP algorithms to find the optimal policy. This is typically done by maximizing the utility function expressed by the Bellman equation:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} p(s'|s,a) U^\pi(s') \tag{3.5}$$

All the equation variables are mentioned in the previous section. Direct RL is another approach where the optimal policy is found by directly estimating utilities for actions without estimating the transition model. This is typically done by learning a utility function of actions and states rather than a utility function of only states. Then, the optimal policy is the path of actions which yields the highest estimated utility. The action-utility function are directly related to the state utility function:

$$U(s) = \max_{a \in A(s)} Q(s,a) \tag{3.6}$$

The difference between using a Q-value function rather than the regular Bellman equation, is that by using a method called Temporal Difference (TD) learning the agent does not need the transition model. The Q-value function is expressed by:

$$Q(s,a) = R(s) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} Q(s',a') \tag{3.7}$$

The transition model is still present, but using TD learning the Q-function can be estimated with the following update rule:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{3.8}$$

Where $\alpha \in [0,1]$ is the learning rate, which determines how much the Q-values are updated. Using this TD update rule is referred to as Q-learning. There exists other variations of this update rule. State-Action-Reward-State-Action (SARSA) is a method which uses TD learning, but does not include the max operator that Q-learning uses.

3.1.4 Exploration vs. Exploitation

RL algorithms are as previously mentioned model-free methods that can be used in the search of an optimal policy. They rely on performing random actions exploring the environment such that they can learn the true model of the environment. However, doing only this would require the agent to search a wast space containing all combinations of all possible actions. This would be equivalent to a brute force approach, and would not be very efficient. To make the search faster, the agent may narrow down the search by following a greedy policy to some chosen point in the search. The greedy policy would be the current optimal policy. After the chosen point, the agent starts exploring new actions that have not been executed along that path before. This way the agent exploits locally optimal policies while exploring the search space. It is important that the agent is able to balance exploration, performing random actions, and exploitation, performing actions in the action space that currently yields the highest utility. The trade-off between exploration and exploitation is typically done by performing random actions some fraction ϵ of the time, and the

rest of the time $1 - \epsilon$ executing the greedy policy. It is also possible to use a exploration function designated to determine what actions to perform next. This will be further discussed in the Section 3.2 regarding RL using MCTS.

3.1.5 Partially Observable Environment

To this point it has been assumed that the environment state is always known, but there are cases where it is not know. This is in the case of a Partially Observable Markov Decision Process (POMDP). In a POMDP a sensor model may to be introduced. The sensor model is a measure for how likely the observation z of the state s are

$$P(z|s) \tag{3.9}$$

Using the sensor model combined with the observations, state estimation/filtering methods may be used deal with the uncertainty. Other methods dealing with partial observability will be further discussed in Section 4.2.

3.2 Reinforcement Learning using Monte Carlo Tree Search

MCTS is a heuristic search algorithm used in RL. The description of MCTS in this section is based on [31] unless otherwise is specified. Principles of RL are combined with a classic tree search implementation to create a probabilistic driven search algorithm. The algorithm being heuristic means it uses a heuristic function pointing it in the right direction making it faster and able to handle greater problems than a normal tree search algorithm such as Depth-first-search. MCTS builds a search tree incrementally while focusing on the most important areas, causing the tree to be asymmetrical. The nodes of the tree represents the MDP states. The edges between the parent and child nodes represents the actions performed in the parent node to reach the child node.

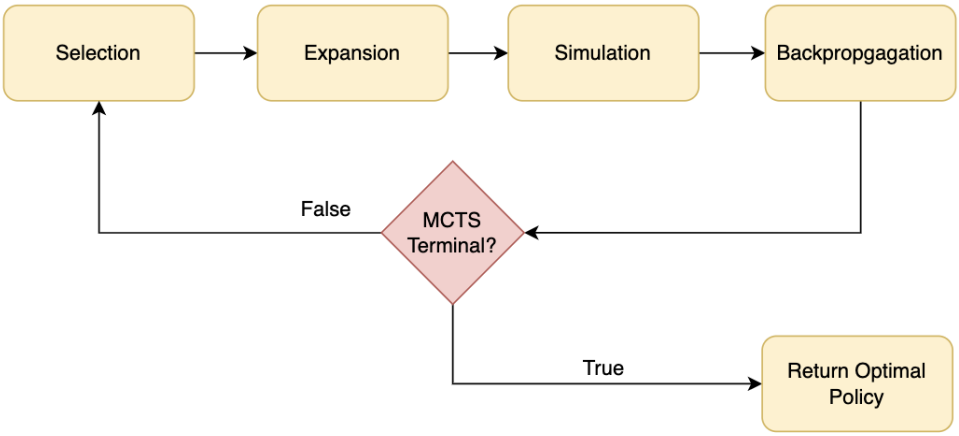


Figure 3.2: Generalized MCTS flow diagram.

The algorithm starts in the root node, which is the initial state s_0 of the MDP. From there on the the four core steps of MCTS is performed, *Selection*, *Expansion*, *Simulation* and *Backpropagation*. The flow of the algorithm using these steps are illustrated Figure 3.2. Additionally it is possible to implement a method called Progressive Widening (PW), to handle large and even continuous action spaces.

3.2.1 Selection

Selection is the step where the algorithm chooses which child node to go to next. Starting in with only s_0 the first child node chosen is s_0. When the tree has grown the next child node is chosen according to two criteria, is the node a leaf and what is the Upper Confidence Bound (UCB). Generally, a node is a leaf if all actions available for the state that the node represents has been executed previously. Leaf nodes are chosen preferably to all other nodes in order to evaluate all immediate available actions as they are considered more important than later actions. When there are no leaf nodes at the current level of the tree, the next child node s' is selected by choosing the node yielding highest UCB value. According to [32] the UCB value may be calculated for all nodes while running the MCTS algorithm according to:

$$UCB = \frac{w_i}{n_i} + c\sqrt{\frac{lnN_i}{n_i}} \tag{3.10}$$

Where UCB stands for Upper Confidence Bound, w_i is the number of times a "win" is achieved from the node, n_i is the number of times the node has been considered in simulations, N_i is the number of times the parent node has been considered in simulations, and c is the exploration parameter. Subscript i is used to denote simulations after the i'th "move". The term "move" comes from the application of MCTS in games, where a complete MCTS search is performed for each move made. The same can be said for the term "win", where the objective of the search is finding a game winning strategy. The UCB handles the exploration vs. exploitation trade-off. The first term of the expression is the success rate of the node implying that nodes with high rate of success is preferred. Hence, the algorithm is encouraged exploitation making greedy choices. The second term is related to how often the node has been visited, decreasing for each time the node has been visited. This term is added to make ensure that the algorithm performs exploration in order to not get stuck in a local optimum. The c value is chosen according to how much the exploration should be weighted in the search. The selection step maximizing UCB values at each level of the tree is performed until a leaf node is reached. The selection step is illustrated in Figure 3.3

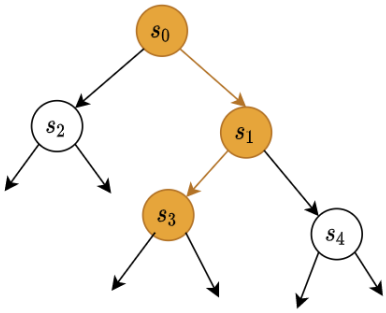


Figure 3.3: Example of the selection step in a search tree where there are 2 available actions in all nodes

3.2.2 Expansion

The expansion step is performed when the selection step is done and a leaf node has been reached. If a child of the leaf node has been visited before, the algorithm performs a complete set of random actions until a goal/terminal state is reached. These actions may lead to the same state as they previously has done or new unexplored states due to the stochastic properties of MDPs. If none of the child nodes has been visited before, the algorithm executes randomly one of the available actions, reaching a new child node s_*. The Value of the node is initialized and set to zero, Value$(s_*) = 0$, and the count of number of visits is set to $n_i = 1$. Figure 3.4 illustrates how the expansion step is performed.

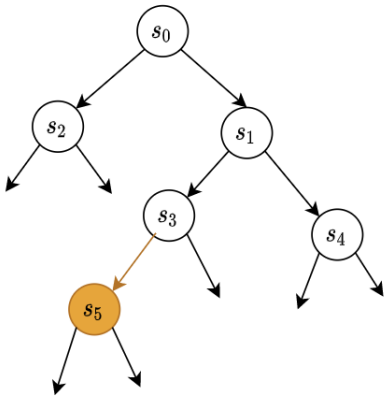


Figure 3.4: Example of the expansion step in a search tree where there are 2 available actions in all nodes

3.2.3 Simulation

The simulation step is executed by continuing to perform random actions until a terminal/goal state is reached. In the goal state a specified reward is collected. The reward may be given by a reward function designed for the specific problem that MCTS is applied to. Reward function design for AST is presented in Chapter 4. Figure 3.5 provides a illustration for how the simulation step is executed.

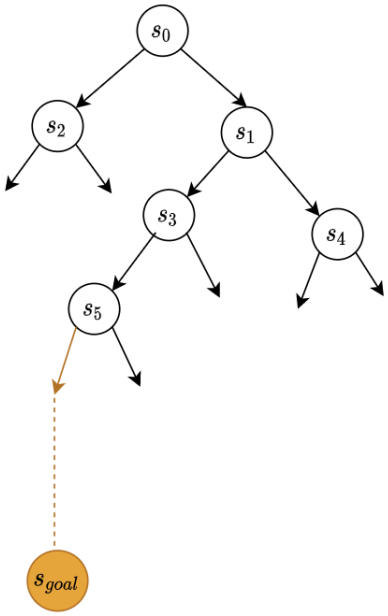


Figure 3.5: Example of the simulation step in a search tree where there are 2 available actions in all nodes

3.2.4 Backpropagation

The backpropagation step is done by following reversing the path following in the tree and updating the visit count $n_i(s_p) = n_i(s_p) + 1$ for each parent node s_p, and by updating the value of the parent

nodes by adding the reward r collected in the terminal state:

$$\text{Value}(s_p) = \text{Value}(s_p) + r \tag{3.11}$$

The backpropgation step is illustrated in Figure 3.6

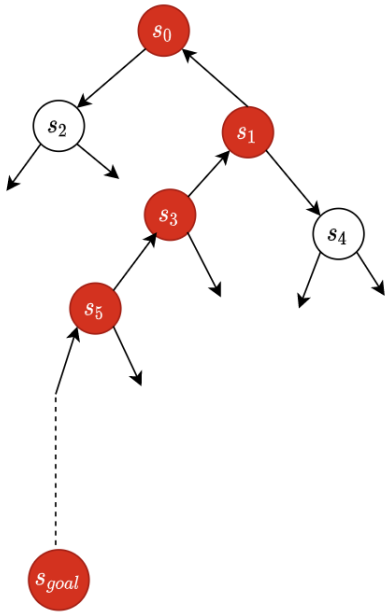


Figure 3.6: Example of the backpropagation step in a search tree where there are 2 available actions in all nodes

This value update is the simplest way to represent values in the nodes. Another approach is by estimating Q-values as explained in Section 3.1.3. When the backpropagation step is finished, the current state is the initial state s_0. Then, the algorithm may continue searching for better policies if the predetermined terminal time of MCTS is not yet reached. If, the terminal time is reached, the algorithm returns the optimal policy found, and depending on the implementation it may execute the policy. In a game the agent would execute the first action or move of the policy and then start a running the MCTS search again when receiving information about the opponents actions.

3.2.5 Progressive Widening

In some cases the action space may be continuous or very large, possibly infinite. When action spaces are large, random sampling is not sufficient when trying to revisit states, making the quality of the value estimates poor. In those cases modifications has to be done to the MCTS algorithm. For AST [2] suggest that Progressive Widening (PW) may be used to deal with this. PW forces the search to revisit existing nodes and allows for exploration creating new nodes when the number of visits increases, progressively widening the tree. It is used to avoid exploding branching factor of the tree. Asymptotically the MCTS search will converge to the optimal solution using PW. The key idea is to limit the number of actions available from each node. Consider the case where there are N available actions from a state s:

$$A(s) = [a_1, a_2, ..., a_N] \tag{3.12}$$

Using PW the number of actions is limited to $K < N$ actions

$$A(s) = [a_1, a_2, ..., a_K] \tag{3.13}$$

where actions in range of $K + 1$ to N are left out. The K is calculated according to

$$K = Ct^{\alpha} \tag{3.14}$$

Where t is the current simulation time step, and C and α are tuning parameters.

Chapter 4

Test Method - Adaptive Stress Testing

This chapter will present how finding the likely path to failure events using Adaptive Stress Testing is modeled as a MDP. The entire chapter is based on the work presented in [2]. The theory presented in this chapter is an edited version of the theory presented in the equivalent chapter in the project thesis, [4].

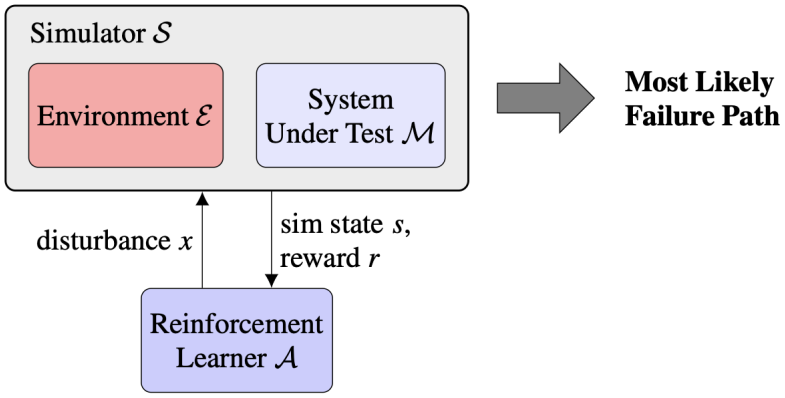


Figure 4.1: High level sketch of the problem formulation. The search for the most likely failure path is formulated as a RL problem. The agent chooses likely disturbances such that the simulated environment is as challenging as possible for the system under test. This sketch is from [2].

In order to formulate the problem consider the sketch in Figure 4.1. The simulator S contains a system under test M and a simulated environment ϵ that the system is acting in. The RL agent A applies some disturbance x based on the simulator state s and a reward r. The disturbance x is sampled from the likelihood distribution $p(x|s)$ and is chosen in order to make the environment ϵ as challenging as possible for the system under test. Then, the AST objective is for the RL agent to find the most likely path to an failure event. Assuming that the simulation is episodic it is set to terminate at t_{end} when the system under test reaches a failure state or when the maximum simulation time t_{max} has passed. The solution to this problem is expressed by multiplying the maximum likelihood at each time step under the constraint that a failure event occurs during the simulation:

$$\max_{x_{0:t_{end}}} \prod_{t=0}^{t_{end}-1} p(x_t|s_t) \tag{4.1}$$

under the constraint of:

$$s_{t_{end}} \in E \tag{4.2}$$

Where $s_{t_{end}}$ is a terminal state and E is the set of failure events. Put into words this means that the solution is the set of disturbances that maximizes the likelihood with the constraint that the terminal state is an failure event.

4.1 Fully Observable Environment

In order to find a solution as the one expressed by (4.1) and (4.2), the problem has to be modeled as an MDP. In this section the case of an fully observable environment is considered. The modeling is done as following:

- The MDP state of the problem is set to the simulator state s.
- The MDP action of the problem is applying a disturbance x in state s.
- The MDP transition model is to be found by the RL agent when optimizing (4.1) under the constraint of (4.2).
- The MDP reward function is designed in order to help the RL agent with the optimization.
- The MDP γ is chosen to be $\gamma = 1$, since the number of time steps is finite.

The reward function is the key to an correct and efficient solution to the AST problem. The function must be designed such that the RL agent chooses likely disturbances and is encouraged to look for failure events. Having this in mind [2] presented the following reward function:

$$R(s,x) = \begin{cases} R_E, \text{ if s is terminal and } s \in \text{E} \\ -d, \text{ if s is terminal and } s \notin \text{E} \\ log(p(x|s)), \text{ Otherwise} \end{cases} \tag{4.3}$$

The RL agent receives a positive reward R_E for reaching a failure state, encouraging it to seek for failure events. When the agent reaches a terminal state that is not a failure state the agent receives a negative reward $-d$. This can either be some distance metric describing how far away the terminal state was from a failure state, or some random number set high enough for the algorithm to converge. When the RL algorithm haven´t reached a terminal state yet, it receives a reward according to the log likelihood of the disturbance. In order to show that this is an optimal reward function, the sum of the received rewards in a single simulation run is calculated:

$$R = \sum_{t=0}^{t_{end}-1} r_t = \sum_{t=0}^{t_{end}-1} R(s_t, x_t) \tag{4.4}$$

Where subscript t denotes at time t. Then, inserting the rewards yields:

$$\begin{aligned} R &= \textstyle\sum_{t=0}^{t_{end}-1} log(p(x|s)) + \begin{cases} R_E, \text{ if s is terminal and } s \in \text{E} \\ -d, \text{ if s is terminal and } s \notin \text{E} \end{cases} \\ &= \textstyle\prod_{t=0}^{t_{end}-1} p(x|s) + \begin{cases} R_E, \text{ if s is terminal and } s \in \text{E} \\ -d, \text{ if s is terminal and } s \notin \text{E} \end{cases} \end{aligned} \tag{4.5}$$

Then, the sum is maximized:

$$\max_{x_{0:t_{end}}} R = \max_{x_{0:t_{end}}} \prod_{t=0}^{t_{end}-1} p(x|s) + R_E \tag{4.6}$$

Equation (4.6) yields a solution on the same form as the desired solution expressed by (4.1) under the constraint of (4.2). Hence, maximizing the accumulated rewards using the proposed reward function will yield a solution to the initial MDP problem, where the objective was finding the path to likely failure events.

4.2 Partially Observable Environment

The AST formulation presented in Section 4.1 assumes that the RL agent has complete access to the simulator state, meaning that the MDP environment is fully observable. However, not all simulators provides full access or access at all to the simulator state. Hence, an AST formulation for partially observable environments must be defined. Previously in Section 3.1.5 the use of state estimation methods were proposed for dealing with POMDPs. However, when the simulator provides no access to the simulator state at all or a sensor model doesn't exist, state estimation cannot be applied. Instead new solutions has to be sought for. In the case of AST this has previously been done by the use of seed-actions. Rather than sampling a disturbance directly, the agent sample a pseudorandom seed $\bar{x}$ that are used in initialization of all random processes in the simulator. The seed is a integer vector used in initialization of a pseudorandom number generator. Assuming that all random processes in the simulator are derived from the same pseudorandom number generator and seed, the agent is able to manipulate all random processes in the simulation by setting the seed. It is assumed that the simulator samples disturbances x in state s by itself according to the likelihood distribution:

$$x \sim p(x|s) \tag{4.7}$$

Assuming all random processes is deterministic given the seed, the likelihood $p(x|s)$ of the sampled disturbance x is then also deterministic given the seed. Then, the sample x is deterministically tied to the seed $\bar{x}$. However, the agent still requires the distance metric d and the log-likelihood ρ in the reward function presented in (4.8). In addition the simulator must notify the agent when a failure event e and terminal state t are reached. This must be provided by the simulator. How this is done, is further explained in Chapter 5 and Chapter 6.

$$R(\rho, e, d, \tau) = \begin{cases} R_E, \text{ if } \tau \wedge e \\ -d, \text{ if } \tau \wedge \neg e \\ \rho, \text{ Otherwise} \end{cases} \tag{4.8}$$

Chapter 5

Test Environment - Situational Awareness Algorithms and Simulator Setup

The objective of this chapter is to is to present the test environment where AST´s capability to find likely failure events in situational awareness will be evaluated. Situational awareness is generated by keeping track of the environment surrounding the autonomous vessel. This is as mentioned achieved by performing target tracking and projection. The test environment consists of the system under test and the simulated environment. Hence, in this case the test environment is a target tracking algorithm connected to a target tracking simulator. The first section will present the target tracking algorithm acting as system under test in this thesis. The second section will describe the simulator design, and different options for AST to control the simulations. The theory regarding the tracking algorithm presented and the simulator design is an edited and expanded version of the theory presented in the project thesis, [4].

5.1 Situational Awareness - Target Tracking Algorithms

Target tracking is related to traditional state estimation/filtering algorithms, but with one main difference. When using state estimators such as the Kalman filter (KF) for linear problems and Extended Kalman Filter (EKF) for non-linear problems, the system only receives one measurement at each time step, and that measurement does only belong to one object. E.g. the GPS measurement for a ship does only belong to that ship. In target tracking, the system may receive several measurements. Therefore, it is important that the algorithm is able to determine which measurement can be associated with an object or target, and hence, should be used. In addition, there may exist multiple targets that the algorithm have to detect and track. Therefore, it is common to distinguish between single-target and multi-target tracking. The typical standard assumption for target tracking is that the target of interest generates at most one measurement at each timestep. If the target does not generate a measurement, a misdetection has occurred in the autonomy system´s object detection algorithm. Additional measurements may be generated by other targets or clutter. Clutter is generated by false alarms in the object detection algorithm. This section will present the underlying assumptions of target tracking. In addition, the two tracking algorithms that will act as system under test, will be presented. This entire section, including the paragraph above, is based on Chapter 7 in [33]. For further details about the theory presented, [33] is highly recommended reading material.

5.1.1 Target Tracking Assumptions

When designing target tracking algorithms, a set of assumptions has to be made in order to make the problem manageable. This thesis will focus mainly on single-target assumptions, and what modifications is necessary to make these work for tracking multiple targets.

Single-target Assumptions

The main objective of a Bayesian single-target tracking algorithm is to estimate the predicted probability distribution $p_{k|k-1}(x_k) = p(x_k|Z_{1:k-1})$, which represents the prediction of the target state x_k given all previously received measurements $Z_{1:k-1}$. In addition, the goal is to estimate the posterior distribution $p_k(x_k|Z_{1:k})$, which is used to estimate the target state x_k by updating the prediction based on information provided by the current measurement Z_k. In order to create models for these distributions, the following assumptions are made:

1. At time step $k-1$ one and only one target exists in the surveillance region, with state vector x_{k-1}.
2. The prior density of x_{k-1} is given as $p_{k-1}(x_k)$.
3. The state vector of the target evolves from time step $k-1$ to k according to a Markov model of the form $f_x(x_k|x_{k-1})$.
4. A measurement from the target is detected by probability P_D
5. If a measurement from a target exists, then it is related to x_k according to the likelihood of the form $f_z(z_k|x_k)$.
6. An unkown number ϕ_k measurements originate from clutter, where the discrete-valued random variable ϕ_k is distributed according to $\mu(\phi)$.
7. If z is a clutter measurement, then it is distributed according to a pdf $c(z)$, independently of all other measurements.

5.1.2 Probabilistic Data Association Filter (PDAF)

The purpose of this section is to become familiar with one of the systems under test by providing a description of the PDAF target tracking algorithm. It is further assumed that the reader is familiar with linear algebra, e.g. the transpose or the identity matrix. Figure 5.1 provides a block diagram representing the flow of the PDAF algorithm. It can be observed that the algorithm consists of seven main steps, EKF State Prediction, EKF Measurement Prediction, Measurement Gate, Calculate Association Probabilities, Event Conditional Measurement Update, Gaussian Mixture Model and Mixture Reduction.

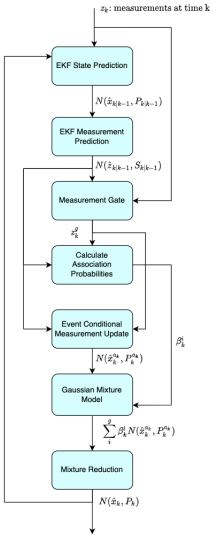


Figure 5.1: Block diagram for the PDAF algorithm.

EKF State Prediction

The EKF State Prediction step is the same prediction step performed by the EKF state estimation algorithm. The prediction done by a

regular KF algorithm is performed by assuming that the process dynamics may be represented by a linear model such as the one presented in (5.1). The EKF does not assume that the process model linear is linear, and must therefore handle this by linearization. However, for the sake of simplicity the process model is assumed to be linear, such that the EKF State Prediction step is actually the KF State Prediction step.

$$x_k = Fx_{k-1} + v_k, \text{ where } v_k \sim N(0, Q) \tag{5.1}$$

Where $x \in \mathcal{R}^4$ is the state vector, $F \in \mathcal{R}^4 \times \mathcal{R}^4$ is the transition matrix, $v \in \mathcal{R}^4$ the process noise vector and $Q \in \mathcal{R}^4 \times \mathcal{R}^4$ is the process noise covariance matrix. Subscript is used to express the time. Assuming that the targets maintains constant velocity F and Q may be represented by the expression in (5.2). The choice of the F and Q matrix is furthered explained in Section 5.2, where the simulator design is presented.

$$F = \begin{bmatrix} I_{2x2} & TI_{2x2} \\ 0_{2x2} & I_{2x2} \end{bmatrix} \text{ and } Q = \begin{bmatrix} \frac{T^3}{3}I_{2x2} & \frac{T^2}{2}I_{2x2} \\ \frac{T^2}{2}I_{2x2} & TI_{2x2} \end{bmatrix} \sigma_a^2 \tag{5.2}$$

Where T is the time step and σ_a is a measure for how much acceleration the target is expected to undergo. The state prediction $\hat{x}_{k|k-1}$ is then given by:

$$\hat{x}_{k|k-1} = F\hat{x}_{k-1} \tag{5.3}$$

Where $\hat{x}_{k-1} \in \mathcal{R}^4$ is the estimated state at previous time step. The output of the prediction step is a multivariate Gaussian distribution. Hence, the predicted covariance has to be caluclated as well. The covariance prediction $P_{k|k-1}$ is given by the following equation:

$$P_{k|k-1} = FP_{k-1}F^T + Q \tag{5.4}$$

Where $P_{k-1} \in \mathcal{R}^4 \times \mathcal{R}^4$ is the estimated covariance from the previous time step. Then, the output of the EKF State Prediction step is given by the following multivariate Gaussian distribution:

$$N(\hat{x}_{k|k-1}, P_{k|k-1}) \tag{5.5}$$

EKF Measurement Prediction

The EKF Measurement Prediction step is carried out by predicting what the measurement should be based on the predicted state. The same assumption about linearity that was made for the EKF State Prediction step is made for this step. Then, the measurement prediction can be based on the following linear model:

$$z_k = Hx_k + w \text{ where } w_k \sim N(0, R) \tag{5.6}$$

Where $z_k \in \mathcal{R}^2$ is the measurement vector, H is the measurement matrix, x_k is as for the prediction step the state vector, $w \in \mathcal{R}^2$ is the measurement noise vector and $R \in \mathcal{R}^2 \times \mathcal{R}^2$ is the measurement noise covariance. The H and R matrices are typically given as:

$$H = \begin{bmatrix} I_{2x2} & 0_{2x2} \end{bmatrix}$$

and

$$R = \sigma_z^2 I_{2x2} \tag{5.7}$$

Where σ_z is the expected measurement noise standard deviation. The measurement prediction is then given by the following equation:

$$\hat{z}_{k|k-1} = H\hat{x}_{k|k-1} \tag{5.8}$$

Where $\hat{z}_{k|k-1}$ is the measurement prediction and $\hat{x}_{k|k-1}$ is the expected value of the EKF State Prediction. The output of the measurement prediction is a multivariate Gaussian. Hence, the measurement covariance has to be predicted as well:

$$S_{k|k-1} = HP_{k|k-1}H^T + R \tag{5.9}$$

Where $S_{k|k-1} \in \mathcal{R}^2$ is the predicted measurement covariance and $P_{k|k-1} \in \mathcal{R}^2 \times \mathcal{R}^2$ is the process covariance predicted by the EKF State Prediction step. The resulting multivariate Gaussian is then:

$$N(\hat{z}_{k|k-1}, S_{k|k-1}) \tag{5.10}$$

Measurement Gate

The Measurement Gate step is where the algorithm determines which measurements at the given time step should be considered as target measurements or clutter measurements. This is done by creating a validation gate G which only accepts measurements within g standard deviations around the measurement prediction $\hat{z}_{k|k-1}$ according to the predicted measurement covariance $S_{k|k-1}$. The resulting gated measurements are then represented by:

$$z_k^g = \left\{ \forall z \in z_k \text{ which satisfies } (z - z_{k|k-1})^T S_{k|k-1}^{-1} (z - z_{k|k-1}) \leq g^2 \right\} \tag{5.11}$$

Calculate Association Probabilities

There exist several models for calculating the association probabilities. The PDAF used in this study use the diffuse model, where the probability of the cardinality of number of clutter measurements is assumed to be uniformly distributed. This means that the number of clutter measurements is given equal probability for all numbers. It is generally hard to find the exact association probabilities for the gated measurements, but it is possible to find values proportional to the association probabilities. Hence, instead of probabilities, association weights are calculated. The association weight for the i'th measurement in z_k is given by:

$$\beta_k^i = Pr\{a_k = i | z_{1:k}\} \propto \begin{cases} \frac{m_k}{V_k} \frac{1 - P_D P_G}{P_D}, \text{ if } i = 0 \\ N(z_k^i; \hat{z}_{k|k-1}, S_{k|k-1}), \text{ if } i > 0 \end{cases} \tag{5.12}$$

Where $a_k = 0$ means that no measurement is generated by the target, and $a_k = m_k$ indicates that measurement m_k originates form the target. Hence, $i = 0$ indicates that the algorithm gates none of the measurements believing no measurement originates from the target, and $i > 0$ indicates that the algorithm believe one of the measurements originates form the target. m_k is the total number of gated measurements at time k, and V_k is referred to as the gate volume. m_k/V_k takes the role of the clutter density, when it is not known a priori. P_G is the probability that the true measurement is inside the gate and P_D the detection probability.

Event Conditional Measurement Update

In order to update the estimate with information gathered from the measurements, the event conditional posterior distribution has to be found. Measurement $z_k^{a_k}$ represents the a_k'th measurement that is considered by the algorithm to possibly originate from the target. The event conditional posterior distribution for each such measurement is given by the following multivariate Gaussian distribution:

$$P(\hat{x}_k | a_k, z_{1:k}) = N(x_k; \hat{x}_k^{a_k}, P_k^{a_k}) \tag{5.13}$$

where the mean vector and covariance matrix is given by the following expressions:

$$\hat{x}_k^{a_k} = \hat{x}_{k|k-1} + W_k (z_k^{a_k} - \hat{x}_{k|k-1}) \tag{5.14}$$

$$P_k^{a_k} = (I - W_k H) P_{k|k-1} \tag{5.15}$$

$$W_k = P_{k|k-1} H^T S_{k|k-1}^{-1} \tag{5.16}$$

Where $\hat{x}_{k|k-1} \in \mathcal{R}^4$ and $P_{k|k-1} \in \mathcal{R}^4 \times \mathcal{R}^4$ are the predicted state and process covariance given from the EKF State Prediction step. $\hat{x}_{k|k-1}$ and $S_{k|k-1}$ are the predicted measurement and measurement covariance from the EKF Measurement Prediction step. $W_k \in \mathcal{R}^4 \times \mathcal{R}^4$ is known as the Kalman Gain. The case where $a_k = 0$ does also have to be considered, then the posterior is given by:

$$p(\hat{x}_k | a_k, z_{1:k}) = p_{k|k-1}(x_{k-1}) \tag{5.17}$$

This is the same distribution outputted from the EKF State Prediction step, meaning that no measurement update is performed. If this carry on for multiple time steps, it is equivalent to dead reckoning, which means that only the process model predictor is used in estimation.

Gaussian Mixture Model

The Gaussian found in the Event Conditional Measurement Update step are in this step combined into a mixture model:

$$\sum_{i=0}^{m_k} \beta_k^i N(\hat{x}_k^{a_k}, P_k^{a_k}) \tag{5.18}$$

At this point the Gaussian mixture model act as the prior for the next time step of the algorithm.

Mixture Reduction

The algorithm assumes that the prior distribution is a single Gaussian. Hence, the Gaussian mixture model violates this assumption. This is handled by reducing the mixture to a single Gaussian using moment matching. Moment matching means that a new Gaussian with the same mean and covariance as the mixture is created. Hence, the expectation value and the covariance matrix of the mixture have to be calculated. The expectation is easily calculated using the linearity of expectations:

$$\hat{x}_k = \beta_k^0 x_{k|k-1} + \sum_{i=1}^{m_k} \beta_k^i \hat{x}_k^i \tag{5.19}$$

The covariance matrix calculation is trickier. First, the "spread-of-innovations" or external covariance term has to be calculated:

$$\tilde{P}_k = W_k \left[\sum_{i=1}^{m_k} \beta_k^i \nu_k^i (\nu_k^i)^T - \nu_k \nu_k^T \right] W_k^T \tag{5.20}$$

Where the innovations are given as $\nu_k = z_k^i - H\hat{x}_k$ and $\nu_k^i = z_k^i - H\hat{x}_k^i$, all other parameters are previously mentioned. The internal covariance is given by:

$$P_{k,internal} = P_{k|k-1} - (1 - \beta_k^0) W_k S_{k|k-1} W_k^T \tag{5.21}$$

The total covariance of the Gaussian mixture is calculated by adding the internal and external covariance:

$$P_k = P_{k,internal} + \tilde{P}_k \tag{5.22}$$

The output of the PDAF filter is represented by the following single mulitvariate Gaussian distribution:

$$N(\hat{x}_k, P_k) \tag{5.23}$$

This distribution is also the prior distribution that acts as input for the EKF State Prediction in the first step of the algorithm, meaning that the algorithm is a recursive algorithm.

5.1.3 Extension to Track Existence: M/N Logic

The PDA algorithm does not take into account that tracks are not initialized *a priori*. Generally, the tracking algorithm should autonomously determine whether a sequence of measurements are likely to originate from a target or clutter. Then, if the measurements are considered to belong to a target, the algorithm should establish a track. There exist different ways of handling this. The simplest approach would be to count the number of times the validation gate contains a detection over a limited time interval. If measurements are repeatedly gated over the interval, it is likely that they originate from a target.

This approach is the core idea behind the M/N Logic track initialization method. The tracker initiates a tentative track when receiving two measurements that are close to each other. How close the measurements have to be in order to create such a track could be determined by calculations using expected maximal speed and measurement noise. The track is then propagated through the tracking algorithm (in this case the PDA) for a maximum N time steps. Then, if the tracker gates M of these N steps, the track is confirmed and established, otherwise the track is terminated.

The methodology described so far handles initiation of tracks, but not termination. The target of a confirmed track may disappear for the tracker. This could be due to the target moving outside the region of interest, because it dissolves, because it becomes invisible or simply due to poor tracking resulting in track loss. Then, it is beneficial to terminate the track, since multiple invalid tracks may confuse the algorithm. This confusion might rise due to the validation gate of a lost track that may quickly grow, making a simple single-target scenario become a complex multi-target scenario. Hence, termination should be handled by the tracking system.

The simplest approach in track termination is the M/N Logic. Then, the algorithm keeps track of the number of time steps where measurements are gated similar to when initiating the track. If the algorithm gates measurements in less than M of the last N time steps, the track is terminated.

5.2 Simulator Design and Setup - Simulating Moving Targets

The objective of this section is to provide a general description of the simulator setup used in the case studies presented later in this thesis. The first section will provide a general description of the simulator components of a multi-target simulator. What specific components and modifications that are included in the different case studies will be presented in the respective case study sections. In the second section the control options for the AST algorithm is presented. Using different ways to control the simulations, AST can modify the simulated environment such that the target tracking algorithms struggles to detect and track targets.

5.2.1 Simulator Components

Chapter 2 provides a general description of an autonomous marine surface vessel. The situational awareness uses sensor fusion algorithms for target tracking and projection in order to keep track of the vessels surroundings. In the previous section a detailed description of underlying assumptions in target tracking in addition to the PDAF target tracking algorithm used in this project is presented. Tracking algorithms use input from perception sensors and traditional navigation sensors. The input are measurements that may consist of noise and clutter. In order to use AST to find likely failure events in the situational awareness, these measurements have to be simulated. Hence, the simulator must create targets and simulate their dynamics. In addition the simulator must be able to add noise to the target states and clutter measurements in order to mimic real measurements realistically. For this thesis an open sea simulator with a fixed frame is considered. This is done in order to illuminate considerations that has to be made when land has to be considered a part of the measurements and to avoid taking into consideration that the surface vessel is moving while tracking. Figure 5.2 presents the simulator setup used in this thesis. This section will present the different components of the simulator setup. The entire section is based on theory presented in [33].

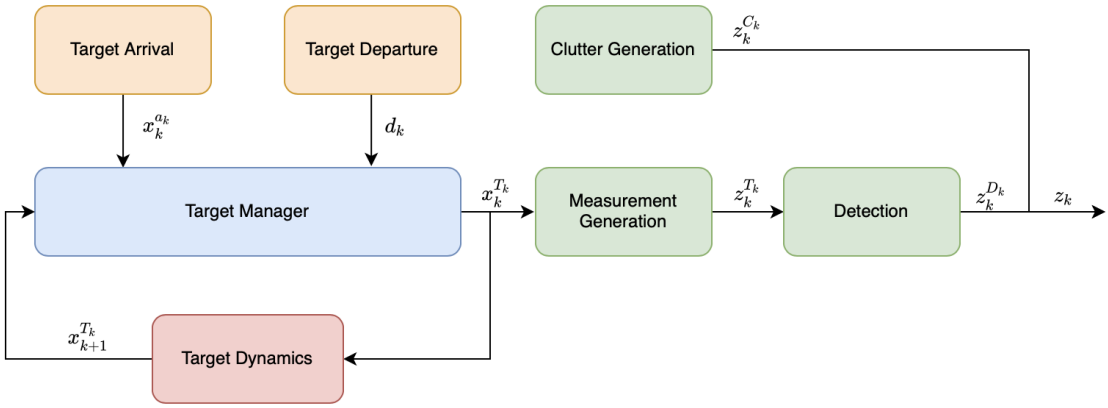


Figure 5.2: Proposed simulator setup for complete multi-target simulation.

Target Arrival Process

Targets are modeled to arrive in the simulation frame according to a Poisson Point Process. First, the number of targets at a given time k is sampled from the following distribution:

$$P(N = n) = \frac{\Lambda_a^n}{n!} e^{-\Lambda_a} \tag{5.24}$$

Where N is the random variable and n is the realization of n. Λ_a is the expected value of N. When the number of targets have been chosen, the initial position for each target is drawn i.i.d from a uniform distribution over the simulation frame. Then, the output of the target arrival component is the set of initial states $x_k^{a_k}$ for arriving targets at time k. Where superscript a_k denotes the set of arriving targets.

Target Derparture Process

Targets may departure the simulation frame or disappear from the measurements. This selection is done as a Bernoulli trial, which is a trial where the outcome is either "success" or "failure" with the same probability of "success" each time. The Bernoulli trial i performed by sampling from a Bernoulli distribution. The sampling yields a boolean value deciding whether a target should be killed or not. The Bernoulli distribution where r is the probability for a target to be killed is presented in [33]:

$$p(X = x) = \begin{cases} 1 - r & , \text{if } x = 0 \\ r & , \text{if } x = 1 \end{cases} \tag{5.25}$$

The Bernoulli trial may in practice be done by sampling a value p from the uniform distribution over the range of [0, 1]:

$$p \sim U_{[0,1]} \tag{5.26}$$

Then, if the sampled value p is above a decision threshold p_d, which is the probability of departure, the target is deleted. The output of the departure component is then the set of departing targets d_k.

Continuous Time Target Dynamics

The continuous time target dynamics are modeled according as a nearly constant velocity (CV) model:

$$\dot{x} = Ax + Gn \tag{5.27}$$

The A and G matrices are defined as:

$$A = \begin{bmatrix} 0_{2x2} & I_{2x2} \\ 0_{2x2} & 0_{2x2} \end{bmatrix} \text{ and } G = \begin{bmatrix} 0_{2x2} \\ I_{2x2} \end{bmatrix} \tag{5.28}$$

x is the target state vector consisting of position and velocity, n is the process noise given by

$$n \sim N(0, D\delta(t-\tau)), \text{ where } D = \begin{bmatrix} \sigma_a^2 & 0 \\ 0 & \sigma_a^2 \end{bmatrix} \tag{5.29}$$

Where σ_a is a measure for how much acceleration the target is expected to undergo. $\delta(t-\tau)$ is the Dirac delta function, where τ is a short time interval $\tau = t_k - t_{k-1}$.

Discrete Time Target Dynamics

To practicably be able to implement the CV dynamics, the CV model has to be discretized. This is done according to the following expression:

$$x_{k+1} = Fx_k + v_{k+1} \tag{5.30}$$

Where x is the state vector, F is the transition matrix and v the process noise vector. Subscript is used to express the time. F and v_k is given by the following equations:

$$F = e^{A(t_{k+1}-t_k)} \tag{5.31}$$

$$v_{k+1} = \int_{t_k}^{t_{k+1}} e^{A(t_k-\tau)} Gn(\tau)\, d\tau \tag{5.32}$$

The process noise inherits whiteness from the continuous time model, such that $v_k \sim N(0, Q)$. Hence, v_k is properly represented by its covariance matrix Q. Using fixed discretization time $T = t_{k+1} - t_k$ results in the following expression for the covariance matrix Q:

$$Q = \int_0^T e^{(T-\tau)A} GDGe^{(T-\tau)A^T}\, d\tau \tag{5.33}$$

This integral may be evaluated using approximations or using a closed-form solution when techniques for evalutating the matrix exponential is available. The closed form solution is given by Van Loan's formula:

$$exp\left(\begin{bmatrix} -A & GDG^T \\ 0 & A^T \end{bmatrix} T\right) = \begin{bmatrix} \times & v_2 \\ 0 & v_1 \end{bmatrix} \text{ and } Q = v_1 v_2 \tag{5.34}$$

The matrix exponential indicates that a complete solution is not yet achieved. The evaluation of matrix exponentials is not necessarily supported by all programming languages, and generally the evaluation would be slow. Hence, if possible, it would be favorable to evaluate the matrix exponential in terms of elementary functions. This is the case for the CV model, where all terms of order 4 and higher becomes zero in the series expansion. Hence, it can be represented by 3rd order expansion. The solutions for F and Q is given by:

$$F = \begin{bmatrix} I_{2x2} & TI_{2x2} \\ 0_{2x2} & I_{2x2} \end{bmatrix} \text{ and } Q = \begin{bmatrix} \frac{T^3}{3} I_{2x2} & \frac{T^2}{2} I_{2x2} \\ \frac{T^2}{2} I_{2x2} & TI_{2x2} \end{bmatrix} \sigma_a^2 \tag{5.35}$$

The target state vector x_k is then sampled according to:

$$x_{k+1} \sim N(Fx_k, Q) \tag{5.36}$$

Let superscript T_k denote the set of existing targets at time k, then the resulting output from the target dynamics component is the set of true target states $x_{k+1}^{T_k}$ at time step $t = k+1$.

Target Manager

The target manager is the component that manages the targets at each time step. The main objective of this component is to merge the set of existing target states $x_k^{T_{k-1}}$ from the previous time step with the set of initial target states $x_k^{a_k}$ for the arriving targets at the current time step. In addition the target manager removes the set of departing targets d_k from the set of targets. The output of the target manager component is then the updated set of target states $x_k^{T_k}$ at time step k.

Generating Measurements

When the simulator for the targets and their behaviour are finished, the measurements originating from the targets must be generated. This is done by transforming the target states into measurements using a linear measurement model:

$$z_k = Hx_k + w_k, \text{ where } w_k \sim N(0, R) \tag{5.37}$$

The R matrix is the measurement covariance matrix, and is set by:

$$R = \sigma_z^2 I_{2x2} \tag{5.38}$$

Where σ_z is the expected measurement standard deviation. The target state vector is x, and the measurement matrix is given as:

$$H = \begin{bmatrix} I_{2x2} & 0_{2x2} \end{bmatrix}$$

The measurement z_k then sampled according to:

$$z_k \sim N(Hx_k, R) \tag{5.39}$$

Let superscript T_k denote the set of targets at time k, then the resulting output from the measurement generation component is the set of measurements $z_k^{T_k}$ belonging to existing targets.

Target Mis-detection:

In reality the radar (or any other measurement device) may not detect all existing targets. Hence, there is some probability assigned to detection of targets. This probability is considered to be constant for every time step. Then, only a certain number measurements of all targets should appear in the measurements. This is done by randomly selecting targets to not include, such that the number of target measurements divided by total number of targets is equal to the detection probability, P_D. This selection is done in the same manner as target departure is handled. Through a Bernoulli trial it is determined whether a target should be deleted generating a mis-detection or kept generating a detection. Hence, the output of the detection component is the set of detected measurements $z_k^{D_k}$ at time k. Superscript D_k denotes the reduced set of measurements where all elements are detected measurements.

Generating Clutter Measurements

Clutter measurements are modeled to arrive in the simulation frame according to a Poisson Point Process. First, the number of clutter measurements n at a given time k is sampled from the Poisson distribution:

$$n \sim poisson(\Lambda) = \frac{\Lambda_c^n}{n!} e^{-\Lambda_c} \tag{5.40}$$

Where Λ_c is the expected number of clutter measurements at each time step. When the number of targets have been chosen, the initial position for each target is drawn i.i.d from a uniform distribution over the surveillance area:

$$z_k^{c_k,i} \sim U_{[x_{min}, x_{max}]} \tag{5.41}$$

Then, the output from the clutter generation component is the set of all clutter measurements $z_k^{C_k}$ at time k, where C_k denotes the set of all clutter measurements at the time step. When the sets clutter measurements and detection measurements are generated, the input to the target tracker is the two sets merged into a single set of measurements z_k.

The log-likelihood

In addition to generate the simulations, the simulator must return the likelihood or log-likelihood of each simulation step, since this is a simulator requirement for AST. The whole simulator step from target dynamics to clutter measurement generation should be manipulated by the AST agent in order to disturb the PDAF algorithm. Hence, the simulator should return the log-likelihood of all the samples done in the simulator. Assuming independence between all the distributions used in a single time step, the log-likelihood for each simulator step is then given by the sum of the log-likelihood for each sample:

$$\begin{aligned}
&log\rho = log(poisson(N = |a_k|, \Lambda_{|a_k|})) + \textstyle\sum_{i=1}^{|a_k|} logU_{(x_k^{a_k,i}=x_k^{a_k,i};[x_{low},x_{high}])} \\
&+ \textstyle\sum_{i=1}^{|x_k^T|} logU_{(R_a^i=r_a^i;[0,1])} \\
&+ \textstyle\sum_{i=1}^{|x_k^T|} logN(X_k^{T,i} = x_k^{T,i}; Fx_{k-1}^{T,i}, Q^{T,i}) \\
&+ \textstyle\sum_{i=1}^{|z_k^T|} logN(Z_k^i = z_k^i; Hx_k^i, R^i) \\
&+ \textstyle\sum_{i=1}^{|z_k^T|} logU_{(R_d^i=r_d^i;[0,1])} \\
&+ log(poisson(N = |z_k^{C_k}|; \Lambda_{|C_k|})) + \textstyle\sum_{i=1}^{|z_k^{C_k}|} logU_{(Z_k^{c,i}=z_k^{c,i};[x_k-\Delta,x_k+\Delta])}
\end{aligned} \tag{5.42}$$

Where the first two sums are related to target arrival, the second to target departure, the third to target dynamics, the fourth to measurement generation, the fifth to detection and the last two are related to generation of clutter measurements.

5.2.2 Control Options

In previous applications of AST using the MCTS algorithm there have been two main options for how AST is allowed to control the simulated environment. These options are open loop global control and closed loop local control. In this thesis the terms local and global refers to whether AST controls a single stochastic process or all stochastic processes in the simulator. In [34] closed loop control is defined as when control is injected into the simulation in real time. For open loop control the control actions has to be set ahead of the simulation run, or indirectly in real time through seed-actions. The last option of open loop control will be explained further in the next sections. The selection of control approach defines the action space for the RL agent. However, in this project a third approach for control is proposed. This new approach benefits from the local control option which closed loop local control exploits, and from the black box property of open loop global control. Hence, the third method of control is named open loop local control. The local control approach may be beneficial since controlling a single parameter reduces the search space. Combining local control with open loop control, this can be done in a less intrusive manner than when using closed loop control, since open loop does not require access to the simulator state. Local control is also interesting scientifically and educationally, since it allows for freezing all the processes that are not controlled, making it possible to study how different processes affects performance of the system under test. E.g. it is possible to set the target trajectory to be the same through the entire search, while AST determines how the target measurements are distributed along the path. In the next sections the different control options will be presented in detail. What specific methods of control for each specific case study is presented in the respective case studies.

Closed Loop Local Control

For closed loop control the AST agent may apply disturbances to the system under test for each time step of the simulation. The RL learning agent can then adapt its strategy in real time, such that failure events are found fast and efficient. This requires direct access to the simulator state. Hence, in this case the system under test is only considered as a semi black box. When testing target tracking algorithms this must be made possible by letting the AST agent manipulate and create new measurements. Figure 5.3 presents an illustration for how this can be done by allowing the agent to perform the sampling of the stochastic processes in the simulator. The following processes may be sampled:

1. Manipulating the arrival process by:
    - Sampling the number of targets to arrive.
    - Sampling the initial position of the arrived targets.
2. Manipulating the departure process by sampling whether to kill or not to kill a target.
3. Manipulating the target dynamics by sampling of the process noise affecting the ground truth state transitions, $[x_{k-1} \longrightarrow x_k]$.
4. Manipulating the measurements z_k by sampling of measurement noise.
5. Manipulating the detection of targets by sampling whether to include or not include a measurement in the input to the tracker.
6. Manipulate the clutter generation process by:
    - Sampling the number of clutter measurements to generate.
    - Sampling the position of the clutter measurements.

When using AST the agent may be allowed to perform all the six mentioned control options at the same time, or one by one, or combinations of them. However, not all search RL algorithms have support for selecting multiple actions at a single time step, which is the case of the MCTS algorithm used in this project. Having this in mind, the search in this project is limited to only use one of the mentioned options at the same time. Hence, the name local control. The key benefit of this type of control when using AST is that manipulation of specific parts of the simulated environment may isolate and uncover specific failure modes. This is beneficial when the system under test is complex and the number of possible failures is large. Having access to the simulator state will also allow for using more advanced RL algorithms, such as solvers using Deep Neural Networks to estimate the Q-value function. In addition as mentioned earlier, using local control makes it possible to freeze certain parts of the simulation, making it possible to examine how different processes influences the performance of the system under test. When the control strategy has been chosen, the agent would have to balance maximizing the likelihood of each sample, by sampling values close to the mean of each distribution, and creating disturbances the system under test find hard by sampling further away from the the mean.

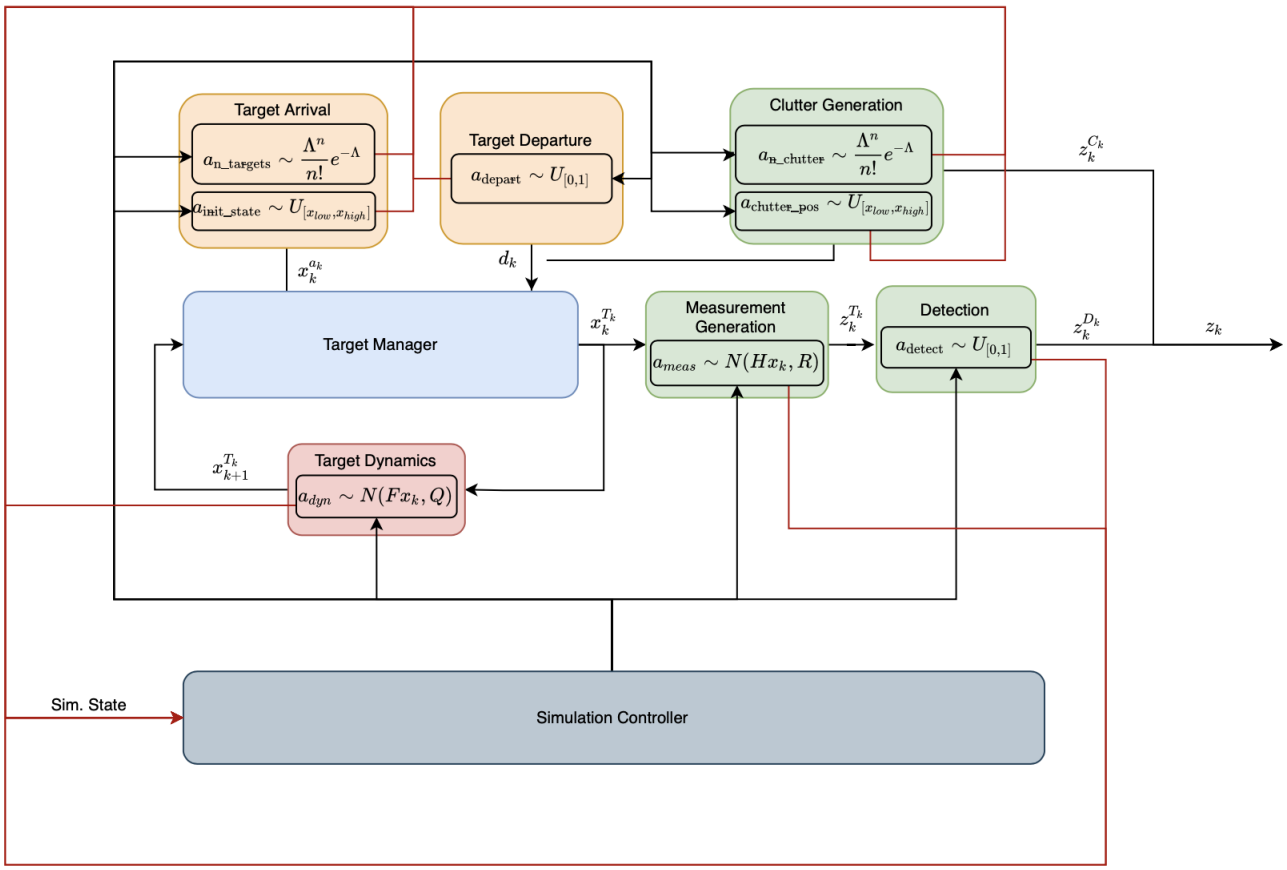


Figure 5.3: Proposed simulator setup for using closed loop control in a complete multi-target simulation. The red lines represents the outputted states of the different simulator components which together makes up the simulator state.

Open Loop Global Control

Open loop control is typically used in the case of a partially observable environment, where the agent does not have access to the simulator state at each time step or any time step. The lack of access to the simulator state may be due to how the system is designed and/or even due to the simulator state being classified information. Hence, the agent must then do necessary modifications to the simulator before each simulation run or indirectly during the simulation run. This works typically well when the simulator run is based on a configuration file. Then, the approach used for partially observable environments in [2] presented in Section 4.2 should in theory work. The AST agent does then only partially have access to the simulator, and instead of applying disturbances directly at each time step, the agent set the pseudo-random seed that is used to initialize the global pseudo-random number generator (RNG) of the simulator. The RNG is used in all the sampling done of the random processes of the simulator. Hence, setting the seed at each time step would re-initialize the RNG and continuously change the sampling, creating disturbances without accessing the simulator state. How the open loop strategy can be used with the proposed simulator is presented in Figure 5.4. Considering a very complex simulator with many different stochastic functions, seed action disturbances would work well. Then, it wouldn't be necessary to create a interface for every function such that the AST agent can manipulate it. This is a key argument for using open loop control with seed action, since it would make the leap from testing only a the situational awareness module of a system to testing multiple modules combined easier. However, the open loop option does not benefit from the isolation of different failure modes that the closed loop option is able to do.

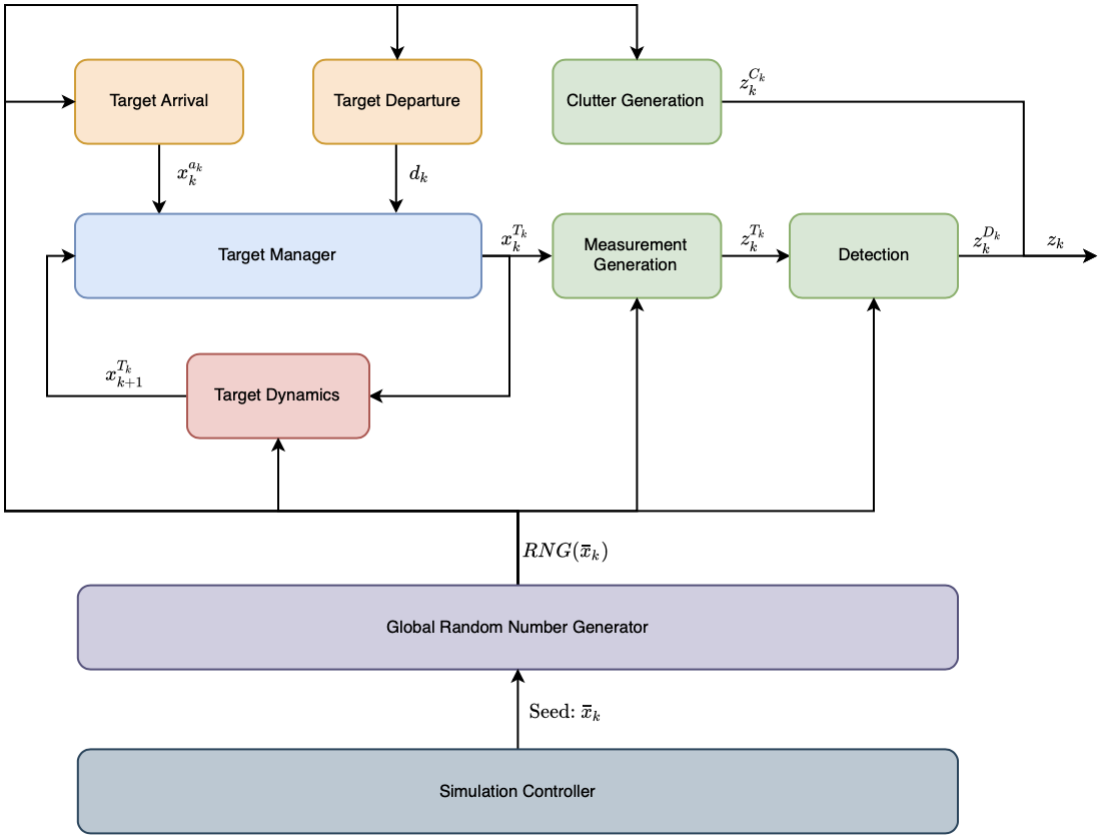


Figure 5.4: Proposed simulator setup for using open loop control in a complete multi-target simulation.

Open Loop Local Control

The key idea of the open loop local control option proposed for this master thesis project is to combine the benefits of the regular closed loop and open loop options. This option would then have the same ability to manipulate specific processes of the simulator isolating different failure modes, and at the same time maintain the black box property of the open loop option. The only requirement for this method of control, is a interface giving access to a separate RNG for the specific process that is to be controlled. This RNG would then have to be used rather than the global one when sampling is done in the isolated process. Figure 5.5 provides an illustration of the open loop local control option for the simulator used in this case. The local control option for the described simulator would allow AST to be able to:

1. Manipulate the arrival process by:
   - Setting a new seed for the RNG used when sampling the number of targets to arrive.
   - Setting a new seed for the RNG used when sampling the initial position of the arrived targets.
2. Manipulate the departure process by setting a new seed for the RNG used when sampling whether to kill or not to kill a target.
3. Manipulate the target dynamics by setting a new seed for the RNG used when sampling of the process noise affecting the ground truth state transitions, $[x_{k-1} \longrightarrow x_k]$.
4. Manipulate the measurements z_k by setting a new seed for the RNG used when sampling of measurement noise.

5. Manipulate the detection of targets by setting a new seed for the RNG used when sampling whether to include or not include a measurement in the input to the tracker.

6. Manipulate the clutter generation process by:

   - Setting a new seed for the RNG used when sampling the number of clutter measurements to generate.
   - Setting a new seed for the RNG used when sampling the position of the clutter measurements.

The AST agent may be allowed to perform all the six mentioned open loop local control options at the same time with separate RNGs or a shared one. Using a shared RNG would be equivalent to the open loop option. The AST agent may also be able to use one by one, or combinations of local control options. This would mean that the Agent is able to do manipulations equivalent to what the agent could do with the closed loop option without accessing the simulator state. When the control strategy has been chosen, the agent would as for closed loop control have to balance maximizing the likelihood of each sample and creating disturbances for the system under test.

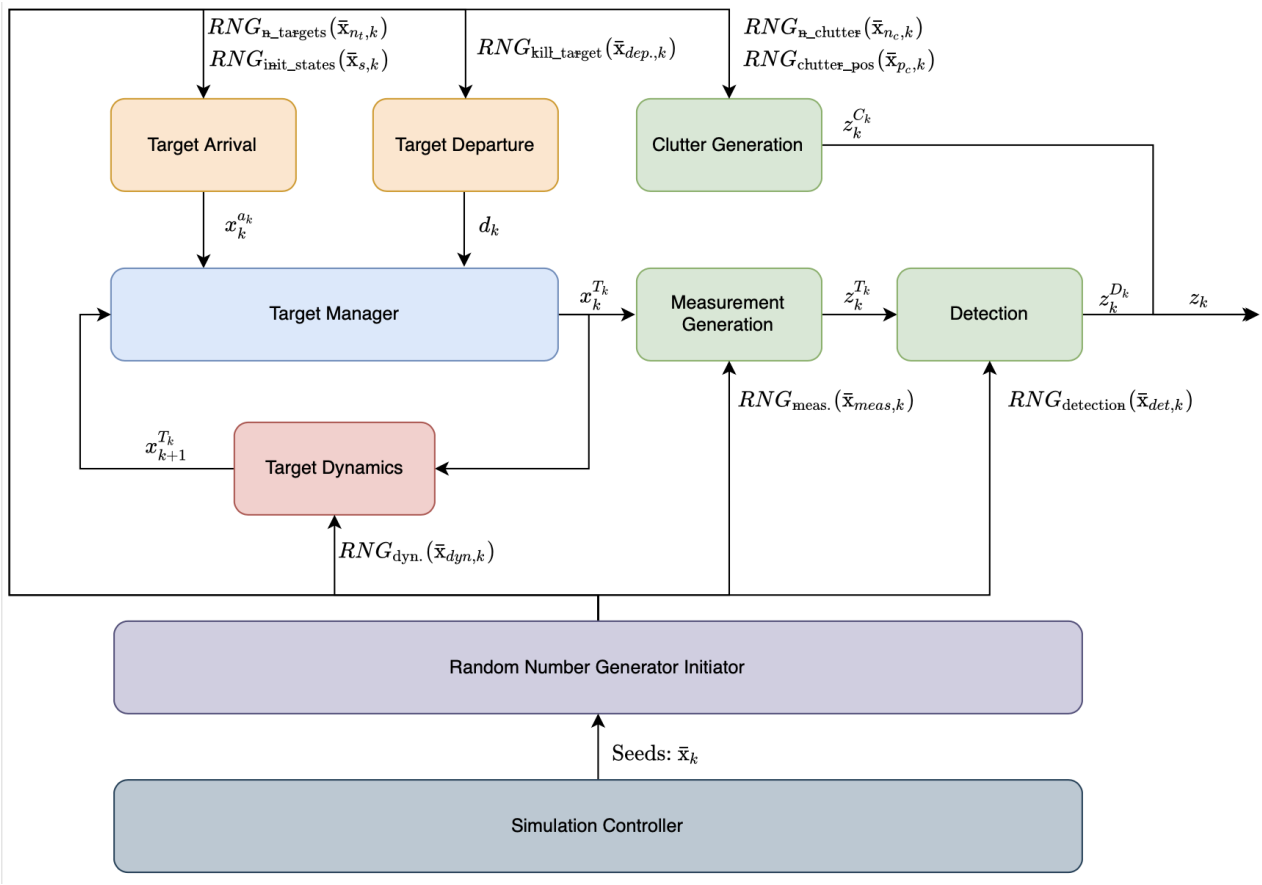


Figure 5.5: Proposed simulator setup for using open loop local control in a complete multi-target simulation.

Chapter 6

Case Studies - Adaptive Stress Testing of Situational Awareness

The objective of this chapter is to perform a set of case studies in order to evaluate AST as a method for finding likely failure scenarios in the situational awareness of an autonomous marine surface vessel. In addition the new open loop local control method proposed Section 5.2 will be tested and evaluated. First, the scientific approach for evaluating the test method is presented. Second, the requirements formulations for the case studies are presented. Then, the specific case studies and the results of the case studies are presented.

6.1 Scientific Approach - Verification and Validation of the Test Method

With the goal of performing effective and thorough evaluation of AST as test method, the scientific approach in this master thesis project has evolved throughout the project while accumulating experience. The core idea behind the final approach is that AST as a test method have to go through the same procedures of verification and validation as the system that AST is intended to test. Generally, before using the method as a tool for verification and validation of an autonomy system, it is important to verify that the method does what it is intended to do and validate that it does so effectively. Before diving into the approach for doing this, a set of definitions have to be presented:

Test Method The test method is the method that will be used to test the system under test, and is the method that is to be tested and evaluated in this context. This term was briefly introduced when presenting AST as a test method in Chapter 4.

System Under Test The system under test is the system which the test method is to evaluate. In this case, the system under test is the PDAF target tracking algorithm presented in Chapter 5.

Environment Simulator The environment simulator is the component which has the role of generating input for the system under test. In this case, the environment simulator is a target tracking simulator, which was presented in Chapter 5.

Test Environment The test environment is the combination of the system under test and the environment simulator. The test environment with system under test and environment simulator is a mentioned described in detail in Chapter 5.

Test System The test system is the complete test setup consisting of both the test method and the test environment.

Having these definitions in mind, it was considered strategical to evaluate the test method through two main phases, comprehensive testing in a low fidelity test environment and further testing in a high fidelity test environment. The complete evaluation process, consisting of phase 1 and 2, is presented in Figure 6.1. This thesis will cover work related to the first phase. In the next sections the two phases will be further explained.

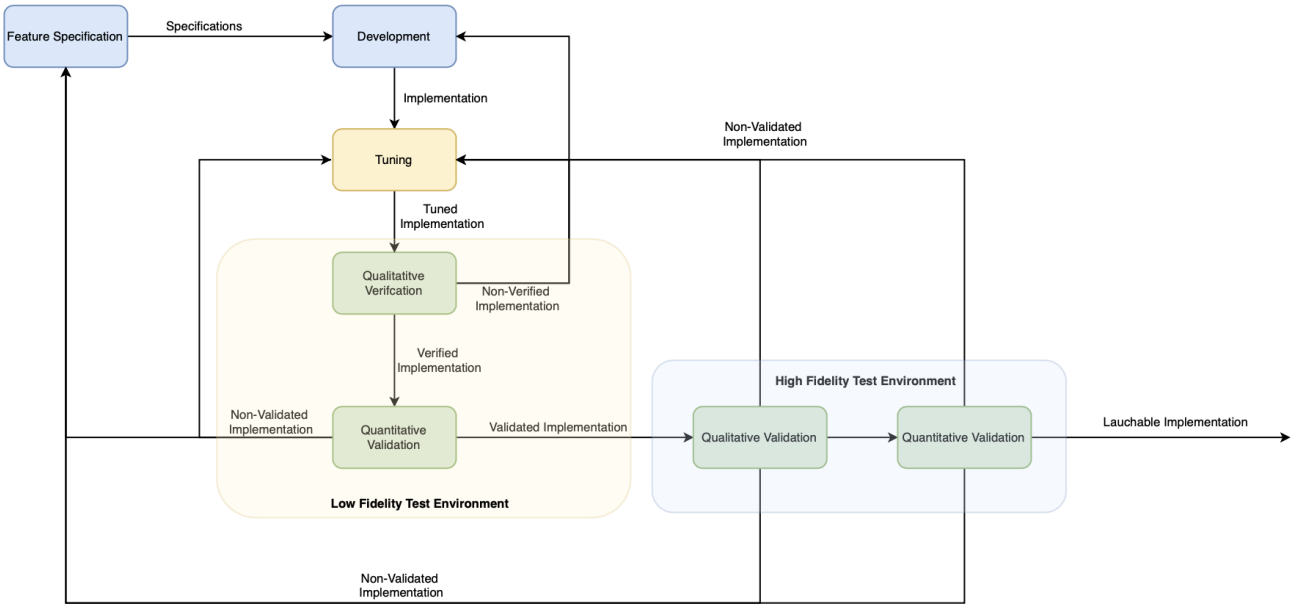


Figure 6.1: Complete development cycle with phase 1, validation and verification in a low fidelity test environment, and phase 2, validation and verification in a high fidelity test environment.

6.1.1 Phase 1 - Low Fidelity Test Environment

In the first phase, the test method is as mentioned to be tested and evaluated in a low fidelity test environment. The test environment is developed and managed by the same developer (the author) which implements the test method. This was considered beneficial at this stage since the developer would have complete control over and insight into the test environment, making thorough testing more manageable than when using a high fidelity test environment, which the test method developer is not familiar with. Having complete control of the complete test system would also make rapid prototyping easier.

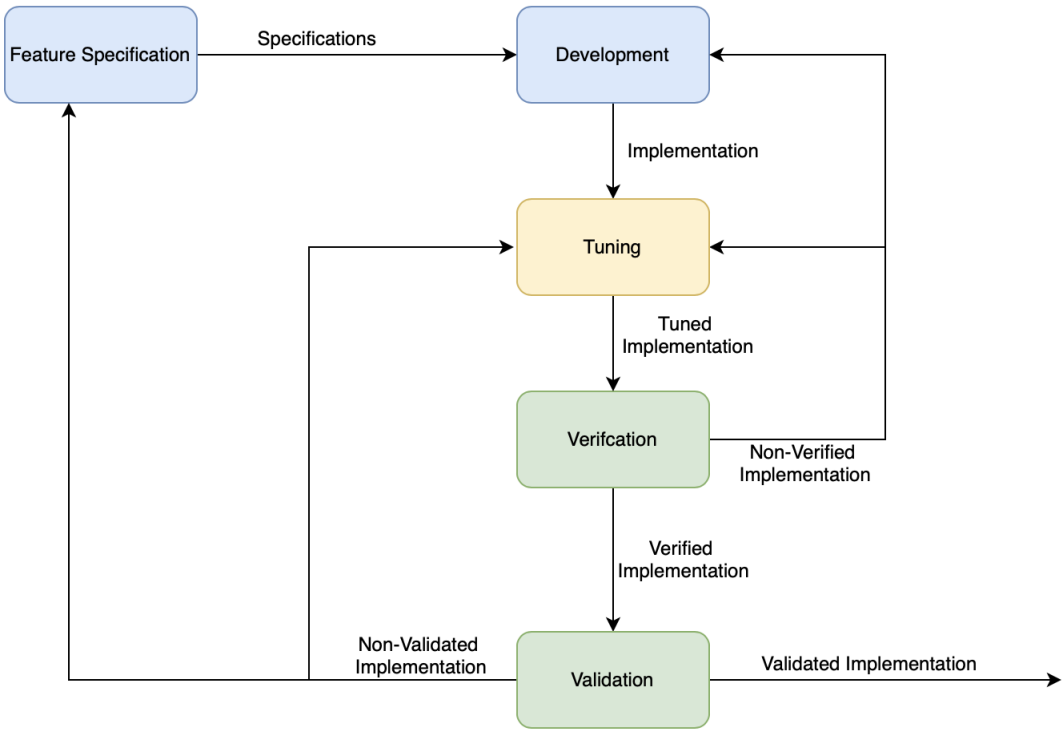


Figure 6.2: Development cycle

The general development strategy for this phase consists of cycles with feature specification, implementation, tuning, verification and validation. This is illustrated in Figure 6.2. More specifically the process is as following:

1. **Feature Specification:** Key features of the product are specified.
2. **Development:** With specifications as input, an implementation is developed.
3. **Tuning:** If applicable, values for the tuning parameters of the implementation are set.
4. **Verification:** Through a qualitative analysis the implementation is verified if it meets its specification requirements.
5. **Validation:** Through a quantitative analysis the implementation is validated if it satisfies its performance requirements.

The steps above gives an description for what the next step is in the development process if the implementation passes all tests. However, it is also important to determine what to do if it does not pass one of the tests. Hence, if the implementation fails in the verification step, the developer should go back and develop a new implementation if it is considered likely that the implementation is erroneous. It is also possible to change the values of the tuning parameters of the implementation. In addition, if the implementation fails in the validation step, new features for improved performance could be determined. Then, the implementation has to go through the same steps again before it can be validated. The second option if the implementation is not validated, is to change the values of the tuning parameters in order to improve performance. In addition it may be considered beneficial to go through the entire loop of specification, development, tuning, verification and validation several times in order to verify and validate the implementation in iterations as it becomes more and more sophisticated. It is important to distinguish between the development of the test method and the test environment, when going through this cycle. In this project, the test environment was developed simultaneously with the test method when extensions of the test environment was required.

The presented development cycle is inspired by Agile development strategies, which focuses on iterative development and early continuous delivery of valuable software. This means in general that having a functional product early, which can be improved in later iterations of the development cycle, is considered better than developing the complete envisioned product from the start. This cycle of development and testing, verification and validation, may be considered to be an iterative cycle of test driven development, which is an Agile development strategy. The key benefit of this strategy is the possibility of continuous evaluation of the product. In test driven development the developer creates tests which the implementation has to pass before the developer starts implementing the product. The key idea behind such a development strategy is that after each cycle, if all tests has been passed, the product is at a stage where it reaches some minimum requirements for being ready for launch. This is referred to as a *Minimum Viable Product*. For further theory regarding Agile development see [35].

6.1.2 Phase 2 - High Fidelity Test Environment

The second proposed phase has a more industrial focus, where the testing and evaluation of the method is done in a high fidelity test environment, validating that the method is useful for testing of an industrial system. In this context, there are a set of variations for how the test environment may be constructed. In order to clarify this see Figure 6.3 for an illustration of different ways the test method can be applied to test an industrial system. The test system developed in phase 1 is the system where the test method developer has implemented the complete test environment consisting of simulator and system under test, which is on the top of the figure. The test method and environment are in that case integrated without need for an interface. The next approach is the test method applied in a industrial test environment, where it must exist an interface between the test environment and the test method. The third approach is using the test method developers environment simulator to test the industrial system under test. Then, there must exist an interface connecting the simulator with the system under test. The last approach is when an independent third party simulator is used. Then, the test method must be connected to the third party simulator through an interface, and the simulator must be connected to the industrial system under test through an additional interface. The three last approaches may be considered to be using a high fidelity test environment compared to the first approach, since for two of the cases the simulator is of higher fidelity than in the first case and the system under test is of higher fidelity in all of the cases.

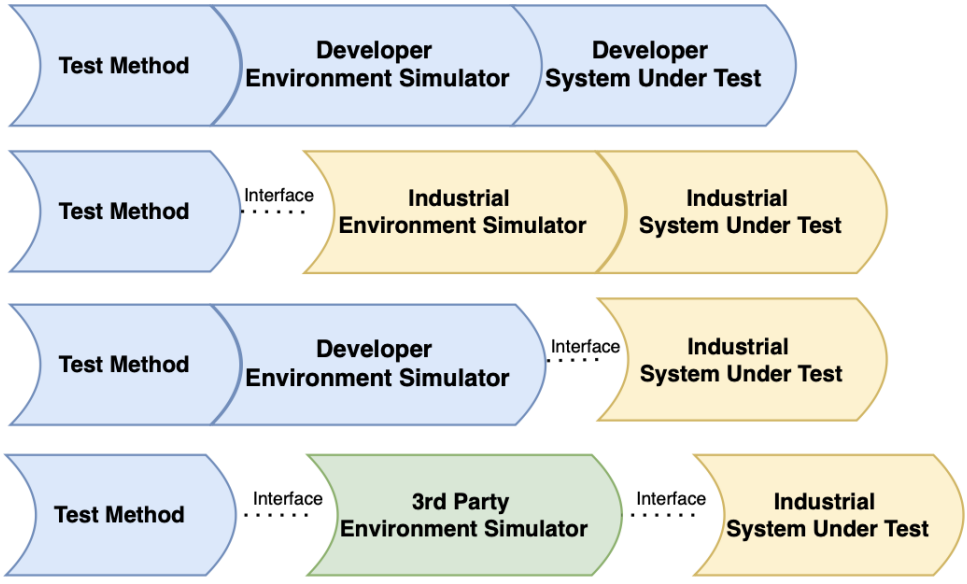


Figure 6.3: Different ways of applying the test method. Blue represents the test method developers systems, yellow represents the industrial systems, and green represents third parties.

The validation of the implementation is done as a two step process. The first step is the qualitative validation in the high fidelity test environment. This means that the implementation is tested for its qualities in the new test environment, which is the same qualities that were verified in the qualitative verification in the low fidelity test environment in phase 1. In addition the end user (or customer) may specify new qualities that the implementation has to be tested for. If the end user is satisfied with the results of the qualitative analysis, the implementation is ready for the second validation step of phase 2, which is a quantitative analysis of the implementation. In this step the implementation is validated if it meets its performance requirements in terms of key performance indicators. This could be comparing the performance of the implementation with existing methods and baseline values. In the case that the implementation is not validated in one of the two validation steps, new specifications may be formulated in order to improve the performance of the method, or the parameter tuning may be improved. In both cases the implementation should go through the same evaluation steps as previously before being validated in the phase 2 validation steps. This is considered beneficial since initial testing and integration of the implementation in a low fidelity test environment would reduce the complexity of the task significantly. In the case that the method is validated in both of the phase 2 validation steps, the validation process may additionally have produced interesting failure scenarios, which may be useful knowledge for the end user.

6.2 Requirements Formulation

The objective of this Section is to provide a formulation of the requirements for applying AST to find likely failure events in the situational awareness of an autonomous surface vessel. The specifications of how the requirements are fulfilled are partially introduced in this section for the components which are common for all of the case studies. How the remaining requirements are met are specified in each case study.

6.2.1 Environment Simulator

Having a environment simulator is a key requirement when applying a simulation-based test method. The environment simulator is as previously explained the environment which the test method is to manipulate in order to evaluate the system under test.

Design

Chapter 5 provided a description of the different components of the multi-target simulator designed and developed for this project. In the different cases presented in next section, different components of the simulator are used to search for specified scenarios. Hence, the simulator setup may differ from case to case. Therefore, the simulator setup will be presented in each individual case.

Implementation

The simulator developed for this project, consisting of all components described in Chapter 5, is implemented in Python. Toolkits such as Scipy, see [36], and Numpy, see [37], was used to handle the stochastic processes and calculations in the simulator.

AST Simulator Requirements

In addition to generating target tracking scenarios, the simulator must fulfill a set of requirements in order to make it possible for the AST method function as specified in Chapter 4. These requirements are related to the reward function. More specifically the requirements are as following:

- Detection of failure events e

- Calculation of distance d from failure
- Notifying when a terminal state τ is reached
- Calculation of the log-likelihood ρ of the simulation scenario

Calculation of the log-likelihood was discussed in Chapter 5 when describing the simulator design. The rest of the requirements are fulfilled in different manners in the different case studies. Hence, this is presented in each individual case study.

6.2.2 System Under Test

The system under test is as mentioned the system which the test method is to manipulate. Hence, having a system under test is crucial for evaluating the test method. In this case the system under test is a target tracking algorithm which may be used in the situational awareness of an autonomous surface vessel.

Design

The system under test in this master thesis project is the PDAF algorithm described in detail in Chapter 5. During the different case studies the different qualities of the algorithm is tested. While doing this, certain components of the system under test are not used. How this is done is explained in each respective case study.

Implementation

In this project the PDAF target tracking algorithm is implemented in Python, similar to the simulator implementation. For the PDAF implementation, the Scipy, see [36], and Numpy, see [37], toolkits are used in the same manner as in the simulator to handle calculations and stochastic processes.

6.2.3 Test Method

Design

The test method is in this case is the AST algorithm, which is to be verified and validated. The AST method implemented in this project follows the design presented in Chapter 4. By running a MCTS implementation using progressive widening, such as the method presented in Section 3.2, AST searches for likely failure events in target tracking.

Implementation

There exist different libraries with AST implementations. This includes two Julia implementations, the POMDPStressTesting.jl toolbox, see [38], and the AdaStress.jl package developed by NASA, see [39]. In addition, there exists an Python implementation by the Stanford Intelligent Systems Lab, see [40]. However, in this project it was considered beneficial to create an implementation which would not require an interface to connect the AST method to the test environment. In addition, creating an implementation would provide insight to the inner workings of the method and facilitate for rapid prototyping. Hence, an AST implementation integrated with the test environment was developed. The implementation was created using a MCTS implementation by [41].

6.2.4 Test System Integration

According to the high level AST setup described in Figure 4.1 in Chapter 4, the AST setup for this case study should be equivalent to the setup presented in Figure 6.4. With the control strategy presented in the previous section, the AST agent sets the simulator seed $\bar{X}_k$ at time step k. Then, a simulator step is performed, and the log-likelihood ρ_k and the distance from failure d_k at time k are returned in addition to two boolean values defining if a failure or terminal event has occurred.

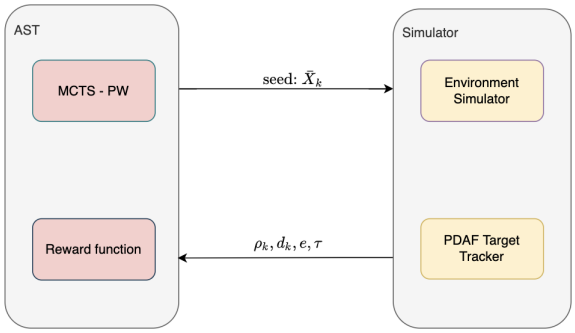


Figure 6.4: Proposed AST setup using seed-actions in open loop control to test the PDAF target tracking algorithm.

However, this AST integration does not provide a description of how the Environment Simulator and the PDAF Target Tracker interacts, or how the required parameters, (ρ_k, d_k, e, τ) for the reward function at time k are derived. How failure events e, distance metrics d and terminal events τ may be defined is as mentioned presented in each respective case study. Introducing a module named Critic, the outputs from the Environment Simulator and the PDAF Target Tracker are mapped to values for the reward function parameters. The log-likelihood ρ is calculated by the environment simulator. The complete AST integration is presented in Figure 6.5.

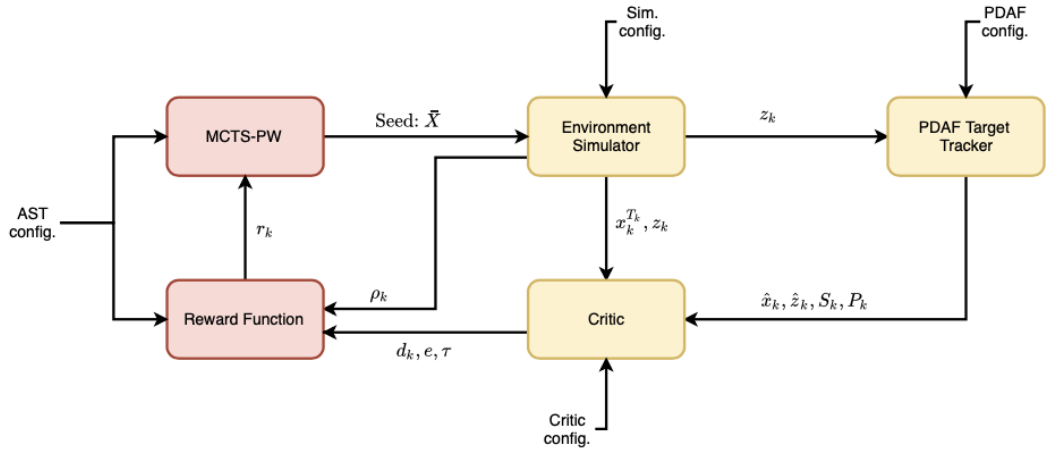


Figure 6.5: Proposed AST setup using seed-actions in open loop control to test the PDAF target tracking algorithm, including the critic module and the interactions between each component.

Each module is initialized with their respective configuration. The configuration includes initial values and tuning values. How these are set will be presented in each respective case studies. When all modules have been initialized, the MCTS search can begin. The MCTS-PW algorithm selects a seed $\bar{X}_k$ at the current time step k. The seed initializes all random processes in the environment simulator in the case of the global control option and a specific stochastic process in the case of the local control option. The environment simulator generates the ground truth states $x_k^{T_k}$ for

all targets in the simulation, the measurements z_k and the log-likelihood of the measurements ρ_k for time step k according to the simulator described in Chapter 5. The measurements are fed to the PDAF Target Tracking algorithm, which calculates the state estimates vector $\hat{x}_k$, the predicted measurement $\hat{z}_k$, the estimated covariance P_k and the predicted innovation covariance S_k for each track. The Critic takes the output from the PDAF module, and the ground truth states $x_k^{T_k}$ and the true target measurements $z_k^{T_k}$ from the environment simulator, calculating the distance d_k from error at time k. In addition the Critic determines if a failure event or terminal event has occurred, returning respectively the booleans e and τ. The reward function receives the output from the Critic in addition to the log-likelihood ρ_k of the simulation at time k from the Environment Simulator and returns the reward for time step k. Then, the MCTS-PW algorithm selects a new seed $\bar{X}_{k+1}$ for time $k+1$ and the entire process is repeated. This is done for a fixed amount of steps, before the AST algorithm returns the most likely failure event.

6.2.5 Parameter Tuning

In the complete test system consisting of environment simulator, system under test and test method, there are several parameters which have to be tuned when running the different case studies. The general tuning strategy for the complete test system is as following:

- Tuning of corresponding simulator and tracker parameters is done through the following procedure:
    1. First, the corresponding parameters are set to be equal and realistic values, being the case of perfect tuning.
    2. Second, if the AST algorithm struggles to find failures with equal values in the simulator and tracker, the parameter values for the simulator are adjusted slightly. Then, the tuning of the tracker shouldn't be perfect anymore, and the probability of error should increase.
- Tuning of not corresponding simulator and tracker parameters is done through the following procedure:
    1. Initially, the parameters of the tracker that does not correspond to the parameters of the simulator will be tuned for maximization of performance. This is done by running different simulations while tuning the tracker.
    2. Second, if the AST algorithm struggles to find failures, the tuning of the tracker may be adjusted slightly, such that the probability of error increases.
- The AST implementation is tuned in order to maximize the performance of the search.
- The Critic parameters was set by deciding what kind of failure event is tested, and for that failure event deciding the appropriate failure threshold and distance metric.

In addition, it is important to mention that the implemented test environment, including system under test and environment simulator, is designed such that it handles metrics of all scales. This means that if the x-position is given as a number, then the number may represent centimetres, metres, kilometres etc. This is the case for all of the parameters of the system. Time is handled in the same manner, where a number may represent the increment of any time interval. E.g. time $t = 1$ may represent a second, a minute or an hour. The key point is that the metrics does not matter in the case studies as long as they all follows the same scale. Hence, results are not denoted with metrics in the case studies.

6.2.6 Method of Control

Before running the simulations of the case studies, it is necessary decide which control option to the AST implementation should use. In Chapter 5 the different control options were explained in detail.

The options were open loop global or local control, and closed loop local control. Each control option has its benefits, making the selection of which to use a qualitative choice. In this master thesis project a key objective is to test, verify and validate the open loop local control option, in addition to the main objective of testing, verifying and validating the AST method applied to target tracking. The local control option has it´s benefit that it makes it possible to isolate a single process of the simulator, reducing the search space and making it possible to study the influence of a single process on the system under test. However, as mentioned in Chapter 2, safety is an emergent system property, meaning that the interaction between the different components of the system may itself introduce errors. Hence, manipulating a single process, while keeping all other processes frozen, does not test for such errors. It is possible to control all processes in the simulator using local control. This would require selecting a seed-action for every process at each time step. However, the MCTS algorithm used as RL solver in this project is only able select one action at every time step while maintaining its performance. In multi-action scenarios, the branching factor of the algorithm becomes intractable large, making the search inefficient. Hence, using MCTS and local control require only controlling a single process. It is possible to apply algorithms which has support for multi-action control. Such a method could be Evolutionary MCTS (EMCTS) as presented in [42], or the extension of EMCTS, Flexible Horizon EMCTS (FH-EMCTS) as presented in [43]. However, it was considered beneficial to keep the main focus on the global control option with regular MCTS when verifying and validating AST as a test method. For each AST search with global control a parallel search is done with local control such that the local control option may be evaluated against the global control option in the quantitative validation case. The verification of the open loop local control method is done in the first case study together with the verification of the open loop global control method.

6.3 Case Studies

In this section are the design and results for six case studies presented. The cases are as following:

1. Verification Cases:
   (a) Qualitative analysis of seed-action control.
   (b) Qualitative analysis of adaptive stress testing of state estimation.
   (c) Qualitative analysis of adaptive stress testing of single-target tracking.
   (d) Qualitative analysis of adaptive stress testing of track initiation.
   (e) Qualitative analysis of adaptive stress testing of track termination.
   (f) Qualitative analysis of adaptive stress testing of tracking multiple targets.
2. Validation Case: Quantitative Analysis of Adaptive Stress Testing of Situational Awareness

The first five of the cases are verification cases where qualitative analyses are performed with the goal to verify that AST is able to find common errors in target tracking algorithms. For each case the complexity of the simulation increases as the AST implementation is further developed in order to handle more complex simulation scenarios produced by the test environment which increases in complexity during development. In addition to the verification cases, a last validation case is performed, where the AST results using both open loop control options are compared to a Monte Carlo Search (MCS), which is a search performing random actions. This is done to validate the efficiency and usefulness of the search. Hence, the cases represents a set of iterations through the development cycle presented in Figure 6.2.

6.3.1 Verification Case 1: Qualitative Analysis of Seed-Action Control

The objective of this case is to provide verification that using open loop control with seed-actions works as described in Chapter 5. First a set of feature specifications and verification requirements are presented. Then, the simulator setup and parameter values of the case are presented. Lastly, the results of the verification procedure are presented.

Feature Specifications and Verification Requirements

The goal is to evaluate both local and global closed loop control through a qualitative analysis. First the following feature specifications are made:

1. Seed-Action Open Loop Global Control should be able to manipulate target tracking simulations in a deterministic manner.
2. Seed-Action Open Loop Local Control should be able to manipulate target tracking simulations in a deterministic manner.

In order to simplify this verification process, the simulator used is a state estimation simulator. This is a target tracking simulator limited to generating a single target trajectory and measurements which exclusively belong to the target. Setting up such a simulation is simply done by tuning. This will be further discussed later in this Section. After the simulator is set up accordingly, the objective is to demonstrate that the following conditions are met:

1. Open Loop Global Control:
   (a) Running a simulation with the same seed initialization of the global random number generator should return the same trajectory and measurement distribution every time in order to ensure determinism.
   (b) Resetting the global random number generator with a new seed after n time steps should return a trajectory and measurement distribution equal to the initial simulation, but with a different trajectory and measurement distribution after the n´th time step.
2. Open Loop Local Control:
   (a) Running a simulation with the same seed initialization of the Local random number generator for the target dynamics should return the same trajectory and measurement distribution every time in order to ensure determinism.
   (b) Resetting the Local random number generator for the target dynamics with a new seed after n time steps should return a trajectory and measurement distribution equal to the initial simulation, but with a different trajectory and measurement distribution after the n´th time step.

Simulator Setup

The simulator in this case is as mentioned set up to produce state estimation simulations consisting of only a single target and measurements originating from that target. How this is done is illustrated in Figure 6.6. The target arrival and departure modules are tuned such that they do not create new targets or delete existing ones. In addition the detection component is tuned such that all measurements are detected. The clutter generation component is limited such that it does not create a single clutter measurement. This is also done by tuning. All the components which are limited by the tuning are faded. Hence, the resulting output of the simulator is the output created by the simulator components enclosed by the box denoted State Estimation Simulator. The parameter values for the simulator are presented in the next section.

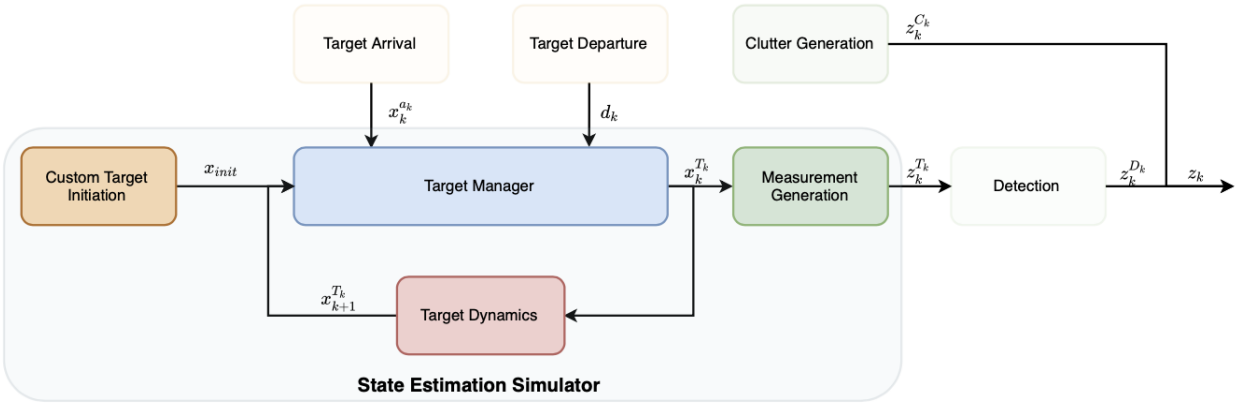


Figure 6.6: Illustration of the state estimation simulator used in this case.

Parameter Values

The parameter values relevant for this case study are presented in Table 6.1. In this case only the simulator parameters are relevant, since the tracking algorithm and the AST implementation were not used. By setting the detection probability $P_D = 1.0$, the expected number of clutter measurements $\Lambda_c = 0.0$ and arriving targets $\Lambda_a = 0.0$, and the probability of departure to be $P_d = 0.0$, the simulator was limited to produce state estimation scenarios in the manner described in the previous section. The other parameters are tuned such that the simulator produces realistic state estimation scenarios.

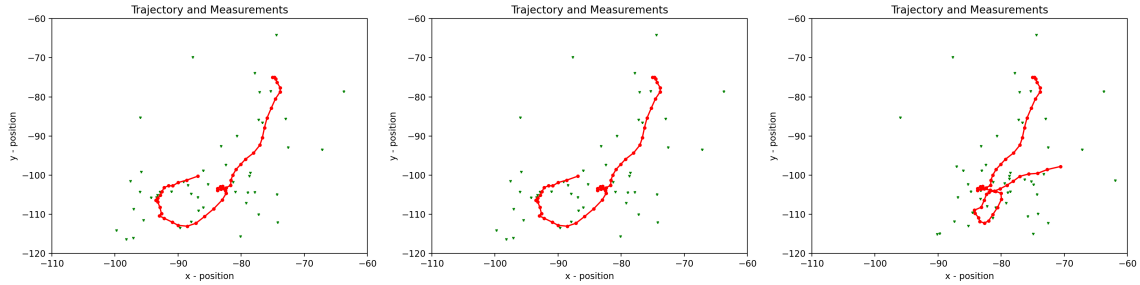
$t_{terminal}$	x_{init}	Ts	σ_a	σ_z	P_D	Λ_c	Λ_a	P_d	Surveillance Area
100	$[0, 0, 0, 0]$	2	0.2	5	1.0	0	0.0	0.0	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$

Table 6.1: Simulator parameter values for the Seed-Action Control verification case. The parameters are explained in Chapter 5.

Results and Discussion

In the next paragraphs the results of verification of respectively open loop global and local control are presented.

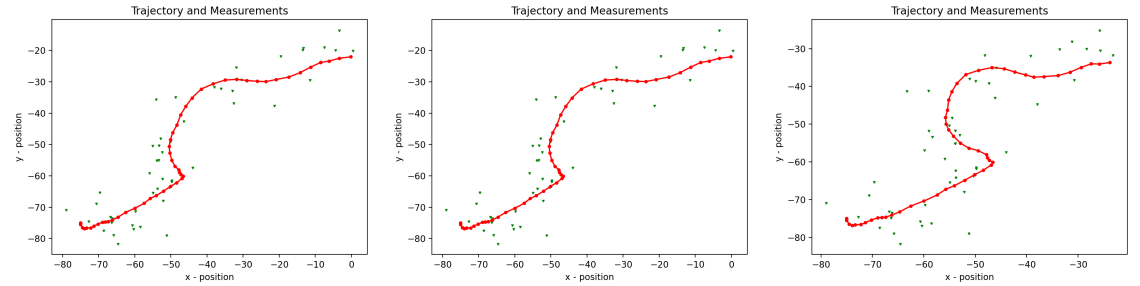
Verification of Open Loop Global Control In Figure 6.7 presents the results of testing the two conditions for verification of Open Loop Global Control. It can be observed that running simulations with the same seed initialization returns the same target trajectory and measurement distribution. Hence, the first condition related to the determinism of the control option is met. Through visual inspection of the track where the seed was changed in the middle of the simulation, it can be observed that the trajectory of the target was the same until the re-seeding, and different after. Evaluating the distribution of the measurements are harder, since the measurements of the new trajectory mixes with the old. However, through close examination it is observed that they were the same in the first half of the simulation. Hence, the second condition is met, which implies that the open loop global control option is verified according to its specifications.



(a) (b) (c)

Figure 6.7: The target trajectory and measurements presented in (a) and (b) originates from running two simulations after each other, both setting the seed of the global random number generator to be 91. The target trajectory and measurement presented in (c) originates from a simulation simular to (a) and (b), but re-seeding the global random number generator at time $t = 50$ with $seed = 120000$.

Verification of Open Loop Local Control Performing the same procedure for the local random number generator for the target dynamics as was done for the global resulted in the results presented in Figure 6.8. Running simulations with the same seed initialization resulted in the same result every time. Changing the seed in the middle of the simulation altered the trajectory slightly after the time step of re-seeding. Hence, the open loop local control option was verified according to its specifications.



(a) (b) (c)

Figure 6.8: The target trajectory and measurements presented in (a) and (b) originates from running two simulations after each other, both setting the seed of the Local random number generator for the target dynamics to be $seed = 33$. The target trajectory and measurement presented in (c) originates from a simulation similar to (a) and (b), but re-seeding the Local random number generator for the target dynamics at time $t = 50$ with $seed = 1$.

6.3.2 Verification Case 2: Qualitative Analysis of Adaptive Stress Testing of State Estimation

Feature Specifications and Verification Requirements

This case has as primary goal to verify that AST is able to find failure events related to some common errors in state estimation, which is a key component of target tracking. The errors of interest in this case are related to filtering consistency, which is presented in detail in the next section. The filtering consistency of a tracker is determined by examining five conditions, where two of these conditions will be tested in this case. For this case study the feature specification is that AST should be able to find common failure events in state estimation related to poor tuning of the PDA filter, which is further specified by evaluating whether AST is able to find events where:

1. The estimation error of the filter is large.
2. The filter show signs of being overconfident.
3. The filter show signs of being underconfident.

The specifications are verified through the following meeting the following conditions:

1. When manipulating a state estimation simulation the AST implementation should be able to find scenarios where the PDA filter yields relatively large estimation error.
2. When manipulating a state estimation simulation the AST implementation should be able to find scenarios where the PDA filter show tendencies where the estimates is drifting away from the true target state, which is the case of being overconfident.
3. When manipulating a state estimation simulation the AST implementation should be able to find scenarios where the PDA filter show tendencies where the estimates are noisy and mostly follows the measurements, which is the case of being underconfident.

Simulator Setup

In order to isolate the state estimation task form the other key components of target tracking, the simulations are run without clutter measurements, mis-detections, target arrival and target departure, which is equivalent to the state estimation simulator described in the previous case study. The simulations are set up in the same manner, by adjusting the tuning parameters. See Figure 6.6 in the previous case for an illustration of the simulator setup.

AST Simulator Requirements

The objective of this section is to discuss how the simulator for this case study should fulfill the AST requirements mentioned in Section 6.2.1. The case study presented in this section is similar to the case study presented in the project thesis. Hence, the theory presented in this section is based on the work presented in the project thesis, [4]. This will include definition of failure events and suitable distance metrics, in addition to defining what a terminal state is. Table 6.2 provides a compact description of possible failure events in the state estimation component of the tracking algorithm. For each failure event a test and distance metric are proposed. In order to detect a failure event it is proposed that the test metric may be compared to some failure threshold. The failure threshold may be set to be an upper, lower or both upper and lower threshold. In order for the AST agent to know when a failure event has occurred, the simulator must implement one of the proposed test metrics and return e if the metric indicates failure and $\neg e$ otherwise. It is also worth to mention that it may not make sense to test for failure in every time step of the tracking algorithm. The result over time is more interesting, since the algorithm may use multiple time steps to correct for errors. Hence, the evaluation of the results should be done at given time intervals or even only at the end of the simulation. The terminal event column defines when a terminal event has occurred. Termination is set to the happen when the simulation has reached its end $t = t_{end}$ for all events.

Failure Event	Test Metric	Failure Detection	Distance (d) From Failure	Terminal Event
Biased Estimation	RMSE	RMSE $\geq RMSE_{thr}$	$\max(\text{RMSE}_{thr} - RMSE, 0)$	$t = t_{end}$
Estimation Error Magnitude	$\bar{\epsilon}$	$\bar{\epsilon}_{l,thr} \geq \bar{\epsilon}$ or $\bar{\epsilon} \geq \bar{\epsilon}_{u,thr}$	$\max(\bar{\epsilon} - \bar{\epsilon}_{l,thr}, 0)$ or $max(\bar{\epsilon}_{u,thr} - \bar{\epsilon}, 0)$	
	$\epsilon_{\%above}$	$\epsilon_{\%above} \geq \epsilon_{\%above,thr}$	$\max(\epsilon_{\%above,thr} - \epsilon_{\%above}, 0)$	$t = t_{end}$
	$\epsilon_{\%below}$	$\epsilon_{\%below} \geq \epsilon_{\%below,thr}$	$\max(\epsilon_{\%below,thr} - \epsilon_{\%below}, 0)$	
Biased Innovation	$\bar{\nu}$	$\bar{\nu} \geq \bar{\nu}_{thr}$	$\max(\bar{\nu}_{thr} - \bar{\nu}, 0)$	$t = t_{end}$
Innovation Magnitude	$\overline{\epsilon^{\nu}}$	$\overline{\epsilon^{\nu}}_{l,thr} \geq \overline{\epsilon^{\nu}}$ or $\overline{\epsilon^{\nu}} \geq \overline{\epsilon^{\nu}}_{u,thr}$	$\max(\overline{\epsilon^{\nu}} - \overline{\epsilon^{\nu}}_{l,thr}, 0)$ or $max(\overline{\epsilon^{\nu}}_{u,thr} - \overline{\epsilon^{\nu}}, 0)$	
	$\epsilon^{\nu}_{\%above}$	$\epsilon^{\nu}_{\%above} \geq \epsilon^{\nu}_{\%above,thr}$	$\max(\nu_{\%above,thr} - \nu_{\%above}, 0)$	$t = t_{end}$
	$\epsilon^{\nu}_{\%below}$	$\epsilon^{\nu}_{\%below} \geq \epsilon^{\nu}_{\%below,thr}$	$\max(\nu_{\%below,thr} - \nu_{\%below}, 0)$	

Table 6.2: In addition to test metrics and failure thresholds for failure detection, the table present possible failure events in target tracking.

The four failure events in the table are related to the filtering consistency of the target tracking algorithm. Filtering in this context is referred to as state estimation. According to [33] filtering consistency is determined by the following five conditions:

1. The state errors should be acceptable as zero mean.
2. The state errors should have magnitude commensurate with the state covariance yielded by the filter.
3. The innovations should be acceptable as zero mean.
4. The innovations should have magnitude commensurate with the innovation covariance yielded by the filter.
5. The innovations should be acceptable as white.

If one of the conditions is not met, the tracker is considered inconsistent. This case is setup such that AST will search for failures related to condition 1 and 2. The following paragraphs will present different test metrics which are related to the conditions.

Filter Consistency Condition 1: The state errors may be tested by comparing the Root Mean Square Error, $RMSE$, of the position component of the state estimate to a threshold $RMSE_{thr}$. The position error at time k, δx_k, is the difference between estimated target position $\hat{x}_k$ and the true target position x_k:

$$\delta x_k = \hat{x}_k - x_k \tag{6.1}$$

Where $\delta x_k, \hat{x}_k, x_k \in \mathbb{R}^2$. For a simulation run with N timesteps, starting at $t = 0$, terminating at $t = t_{end}$, with ground truth state x_k and state estimate $\hat{x}_k$ at time k, the RMSE is calculated according to:

$$RMSE = \sqrt{\frac{1}{N} \sum_{t=0}^{t=t_{end}} \delta x_k^T \delta x_k} \tag{6.2}$$

Filter Consistency Condition 2: Using the estimation error for position δx_t and the estimated covariance P_t, condition 2 can be evaluated using the Normalized Estimation Error Squared (NEES) for postion. The NEES value for position at time k is given by:

$$\epsilon_k = \delta x_k^T P_k^{-1} \delta x_k \tag{6.3}$$

Suppose that $x \in \mathbb{R}^d$, then ϵ_k will be a χ^2 distributed random variable with d degrees of freedom. In the case of NEES for position $d = 2$. Adding together a set of N χ^2 distributed random variables such as ϵ_k, a new χ^2 distributed random variable with Nd degrees of freedom is created. This implies that the average NEES should behave as a scaled χ^2 distribution equivalent to a Gamma distribution:

$$f(x) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-\frac{x}{\theta}} \tag{6.4}$$

Where the scale parameter is $\theta = 2/N$ and the shape parameter is $k = Nd/2$. For such a distribution it is possible to construct a confidence interval between the α and $1-\alpha$ quantile. Typically a 95% confidence interval is created with $\alpha = 0.025$. The upper and lower limit of the confidence interval might be used as a test metric, to evaluate if the average NEES (ANEES) $\bar{\epsilon}$ values are reasonable. This is the first proposed test metric for condition 2 in Table 6.2. Then, a failure event may be defined as when the ANEES value is outside the confidence interval. The same limits may be used as a guideline to evaluate the time series of NEES values. By counting the number of time steps the NEES values were below, inside or above the confidence interval, it is possible to calculate the fraction of NEES values below, inside and above. These fractions may serve as test metrics for condition 2 as well. In Table 6.2 it is proposed that the fraction above $\epsilon_{\%above}$ and below $\epsilon_{\%above}$ the confidence interval compared to some threshold might serve as test metrics. The use of ANEES and NEES values when interpreting filtering results are quite useful. Values above the confidence interval indicates a overconfident filter, trusting its process model too much. When the ANEES and NEES values are below the confidence interval, the filter relies on the measurements to much, creating a jagged estimate.

Filter Consistency Condition 3: The innovation may be tested by evaluating the innovation mean $\bar{\nu}$. The innovation ν_k at time k is the difference between the predicted measurement $z_{pred,k}$ and the actual measurement z_k:

$$\nu_k = z_k - z_{pred,k} \tag{6.5}$$

Where $\nu_k, z_k, z_{pred,k} \in \mathbb{R}^2$.

Filter Consistency Condition 4: The innovation magnitude can be tested in a quite similar manner as condition 2. Instead of calculating NEES, the Normalized Innovation Squared (NIS) may be calculated. Which at time k using the innovation ν_k and the innovation covariance S_k^{-1} yielded by the filter, is calculated by:

$$\epsilon_k^{\nu} = \nu_k^T S_k^{-1} \nu_k \tag{6.6}$$

Then, a confidence interval for the Average NIS (ANIS) may be calculated in the same manner as for ANEES. The confidence interval as a test metric is included in Table 6.2, comparing the ANIS value $\overline{\epsilon^{\nu}}$ against the upper $\overline{\epsilon^{\nu}}_{u,thr}$ and the lower $\overline{\epsilon^{\nu}}_{u,thr}$ limit. In the same way as for the NEES values, the fraction of NEES value below, inside and above the confidence interval may be calculated and used as test metrics. This possibility is also included in Table 6.2.

Filter Consistency Condition 5: The whiteness of the innovations may be tested by applying whiteness tests. However, this will not be covered.

Parameter Values

Environment Simulator Parameters The simulator parameters for this case presented in Table 6.3 are set to the same values as in the previous case, since both cases were state estimation cases.

$t_{terminal}$	x_{init}	Ts	σ_a	σ_z	P_D	Λ_c	Λ_a	P_d	Surveillance Area
100	$[0,0,0,0]$	2	0.2	5	1.0	0	0.0	0.0	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$

Table 6.3: Simulator parameter values for the state estimation verification case. The parameters are explained in Chapter 5.

System Under Test Parameters The parameters of the PDAF algorithm were according to the tuning strategy set to correspond with the parameter values of the environment simulator. The probability of detection P_D and the gating probability P_G were both set to one, making the algorithm gate all measurements. Since no clutter measurements are generated, only a single

measurement will be gated at each time step. This would make Gaussian mixture reduction component of the algorithm neglected, making the PDAF implementation correspond to a Kalman Filter, which is a pure state estimation algorithm. The parameter values of the tracker can be observed in Table. 6.4

σ_a	σ_z	P_D	P_G	Clutter Density	Ts
0.2	5	1.0	1.0	0	2

Table 6.4: Target tracker parameter values for the state estimation verification case. The parameters are explained in Chapter 5.

The M/N logic track initiation parameters are not included in this case, since the track is initiated manually in the beginning of each simulation run to start the track from the initial state of the target, and the simulator is set up to not produce or terminate targets.

Test Method Parameters The AST parameters were as mentioned set to maximize the performance of the MCTS search. This was done through trial and error, resulting in the values presented in Table 6.13. The number of iterations was chosen to be a large number in order to give AST sufficient time to make the MCTS search converge. In addition running a large amount of iterations were necessary to provide statistical proof for the validation case. The UCB weight was set balancing exploration and exploitation, which is a dilemma described in Chapter 3. The same consideration was necessary to be made when selecting the constants for progressive widening, which determines how many actions are available for the MCTS algorithm. By lowering the UCB constant and lowering the number of available actions slightly in a set of iterations, the algorithm reached a balancing point where it was able to find failure events while still allowing for exploration. In addition to all the mentioned parameters two new parameters for weighting of the likelihood and the failure distance in the reward function were created. This was done in order to ensure that the order of magnitude of the two metrics were comparable, making the reward function balance the search of likely events and failure events. The search for failure was considered most important. Hence, the failure weight was set high enough to make the distance from failure d the most dominant component.

N.O. Iterations	UCB Constant c	PW Constant c	PW Constant α	Likelihood Weight	Failure Weight
1000	0.1	2	0.25	1	1e4

Table 6.5: AST parameter values for the state estimation verification case. The parameters are explained in Chapter 3 and 4.

Critic Parameters The Critic parameters were set based on the definitions of failure, RMSE failure and NEES failure, which were described in previous section. First, the PDAF tracker is tested for RMSE failure according to Table 6.2 and the value for the position RMSE threshold is presented in Table 6.14. Second, the PDAF algorithm is tested for NEES failure in to cases by comparing the fraction of NEES values above and below the confidence interval to threshold values presented in Table 6.14. See Section 6.2.1 for further details about the parameters.

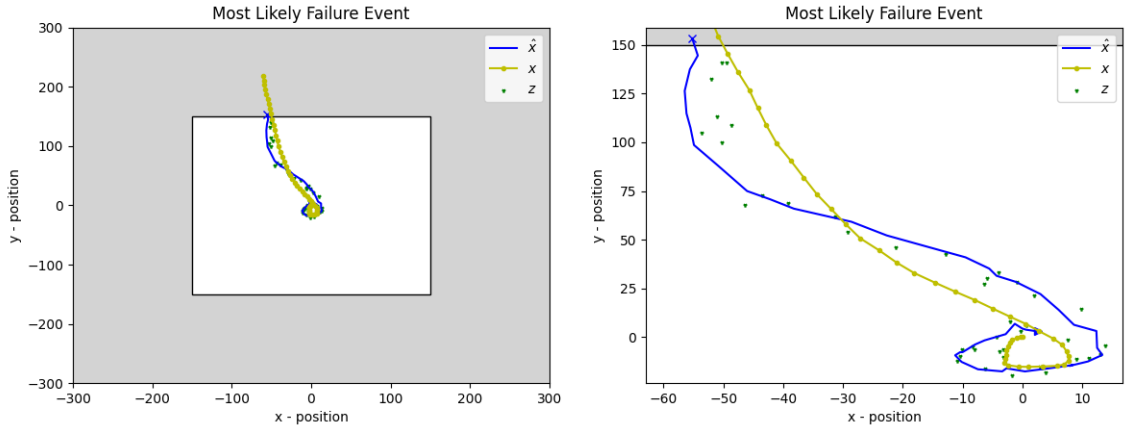
Case	Failure Threshold	Distance Metric
RMSE Failure	$\text{RMSE}_{thr} = 9$	d = max(9 - RMSE, 0)
NEES Failure 1	$\epsilon_{\%above} = 40\%$	d = max(0.4 - $\epsilon_{\%above}$, 0)
NEES Failure 2	$\epsilon_{\%below} = 40\%$	d = max(0.4 - $\epsilon_{\%below}$, 0)

Table 6.6: Critic Configuration

Results and Discussion

In the next paragraphs the are the results of verification of AST´s ability to find some specified failure events in state estimation presented.

Filtering Consistency Condition 1 The first AST run was done in order to find likely errors related to filtering consistency condition 1, which is that the state errors should be acceptable as zero mean. Hence, as mentioned RMSE for position was chosen as test metric. Initially, a search using RMSE threshold of $RMSE > 10$ was performed. However, the method didn't find any failure events with that high RMSE. The highest RMSE returned was slightly above 9. Hence, the threshold was lowered to 9 in order to push the limits of the AST search. After 1000 MCTS iterations the track presented in Figure 6.9 was outputted from the algorithm:



(a) (b)

Figure 6.9: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for state estimation case 1. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track.

It can be observed that the target performs a sharp turn in the beginning of the simulation before the path straightens out for the rest of the simulation. The measurements are in the beginning mostly distributed on the outside of the curve, before changing side of the target trajectory when the target starts to make a slight turn in the middle of the simulation. This makes the PDAF track relatively smooth, with the track staying at the same side of the target trajectory as the measurements through the whole simulation. This is a quite realistic scenario, since the track follows the measurements pretty closely, and the variance of the measurements seems to be quite low, making the track smooth. Such a smooth track is typical for a overconfident filter, which relies more on its process model rather than measurements.

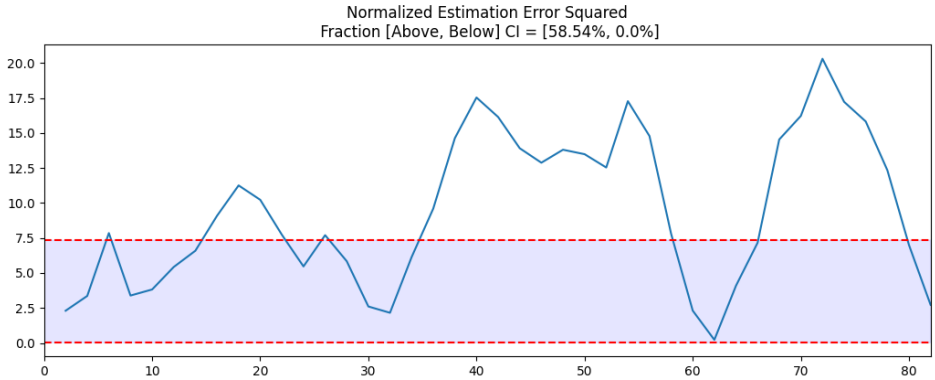


Figure 6.10: NEES time series plot with the 95% confidence interval for state estimation case 1.

Looking at the NEES plot in Figure 6.10 the suspicion of the filter being overconfident is strengthened. The NEES values are above the confidence interval in a large fraction of the simulation, with 58.54% of the values being above. Taking a look at the results in Table 6.7, it can be observed that the ANEES value was above the confidence interval strengthening the hypothesis of the filter being overconfident. However, since the track mostly follows the measurements, this is most likely not the case. The true cause of error in this case is most likely the distribution of measurements, and if the target continued to move inside the surveillance area, the tracker would most likely not struggle to estimate the state with NEES values of such a magnitude.

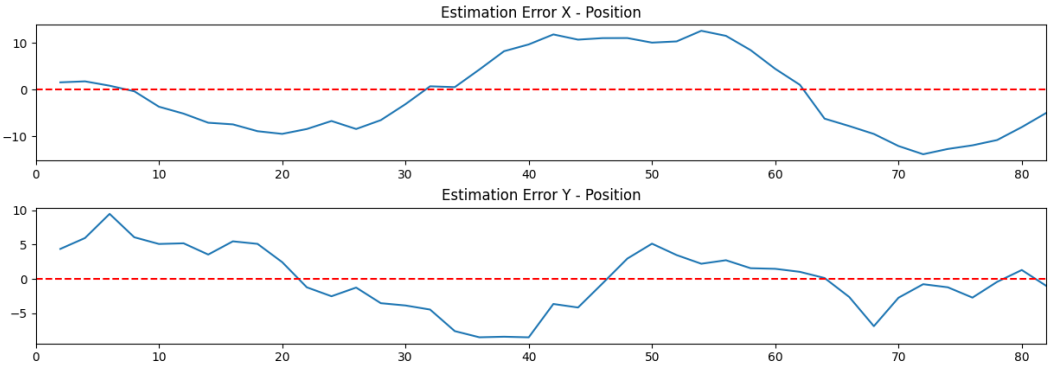


Figure 6.11: Estimation error time series for respectively x and y-position for state estimation case 1.

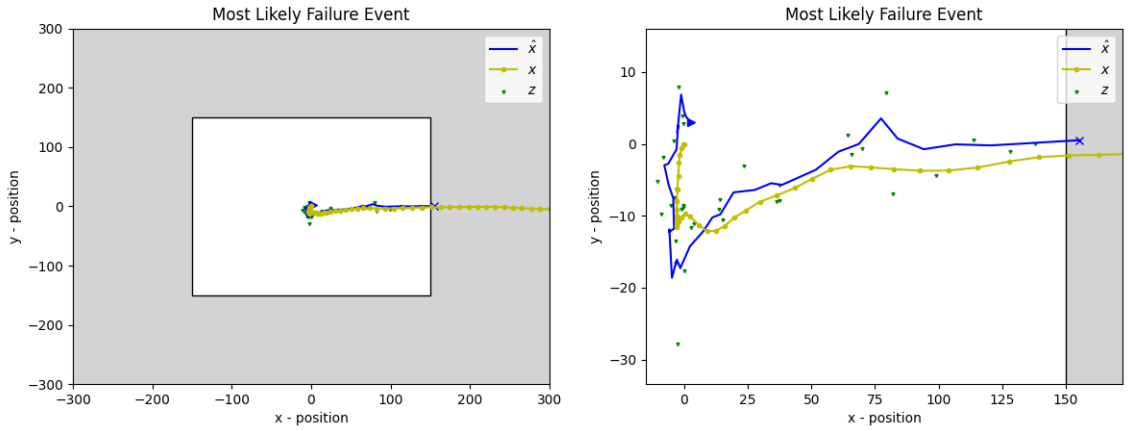
By examining Figure 6.11 it can be observed that the estimation error for the x-position acts quite similar to a sinusoidal curve, meaning that the errors would cancel each other out when calculating the mean error. Hence, this is in line with the first filtering consistency condition. In the plot of the estimation error for y-position it can be observed that the errors would cancel each other out in the same manner when calculating the mean error as for the x-position. Hence, it can't be said that the tracker failed the first filtering condition in this case, since the estimation error on average is approximately zero. This is most likely due to using RMSE as test metric. When computing the RMSE the error is squared, making the minus sign for negative error disappear. Hence, using RMSE as test metric may not have been the best choice for testing this condition. Using the estimation errors or the mean estimation error would probably be a better choice. The position RMSE for track can be observed in Table 6.7. Even though the event found does not make the filter fail according to the condition, the algorithm was able to effectively find a likely event related to high position RMSE, verifying that the algorithm can find such events. Hence, further testing is required to test the first filtering consistency condition, but the algorithm shows promising results implying that this should be possible. Regarding the requirement specified in the beginning of the case section, which stated that AST should be able to find events where the

estimation error is relatively large, the method is verified.

Metric	Value
RMSE Position	9.25
RMSE Velocity	1.35
ANEES	9.4
ANEES 95% Confidence Interval	[0.051, 7.378]
NEES Fractions	
Above CI	0.59
Below CI	0.0
Inside CI	0.41

Table 6.7: State Estimation Performance Measures for state estimation case 1.

Filtering Consistency Condition 2: Overconfident State Estimation The second AST run was done in order to find likely errors related to filtering consistency condition 2, which is that the state errors should have magnitude commensurate with the state covariance yielded by the filter. Hence, as mentioned the NEES fraction above the 95% confidence interval for position was chosen as test metric. Initially, a search using a threshold with a fraction of 20% above the confidence interval was performed. However, when running the Monte Carlo Search in parallel it became clear that this was an error which was too easy to find. Hence, the threshold was increased to 40%. After 1000 MCTS iterations the track presented in Figure 6.12 was outputted from the algorithm:



(a) (b)

Figure 6.12: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for state estimation case 2. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track.

Similar to the previous case it can be observed that the track stayed on the same side of the target trajectory as the majority of the measurements. The main difference which can be observed in this case, was that the variance of the measurement distribution was much higher, making the estimate more jagged. After the turn made in the beginning of the trajectory, the track seems to stay on one side of the trajectory and slightly moving farther away from it. This may be an indication of drift. However, considering that the track seems to follow the measurements it may not be the case.

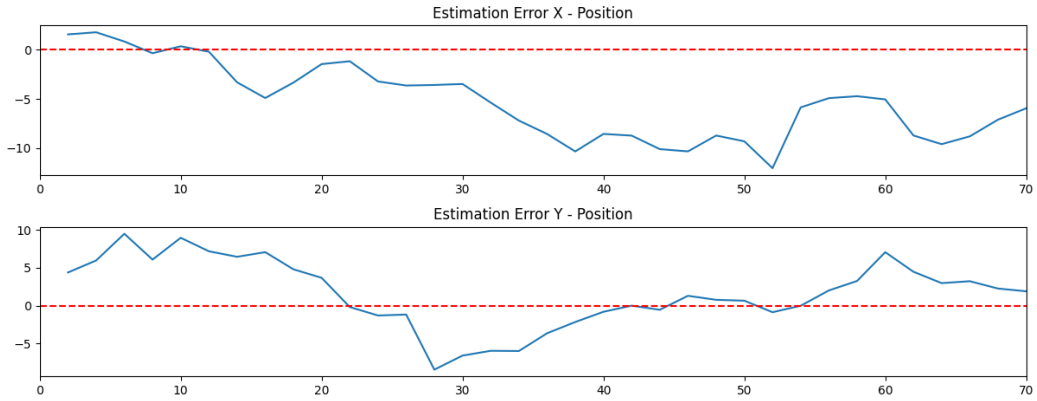


Figure 6.13: Estimation error time series for respectively x and y-position for state estimation case 2.

Studying the estimation error plot in Figure 6.13 The drift tendency can also be observed in the estimation error of the x-position. In the period after the turn is made, the estimation error for x-position seems to grow throughout the simulation.

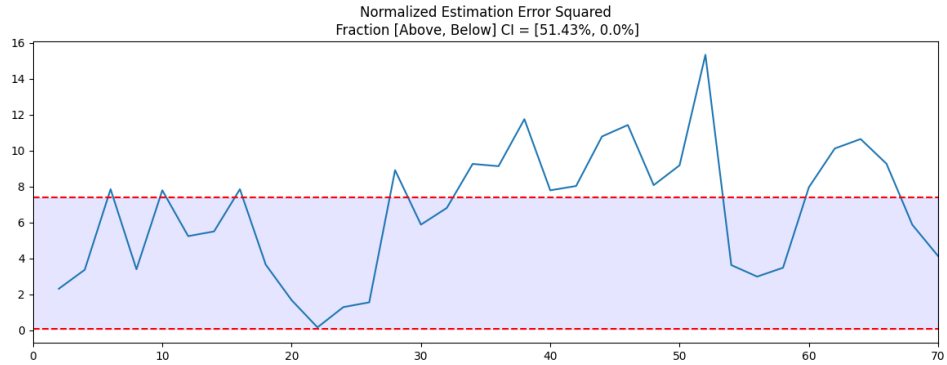


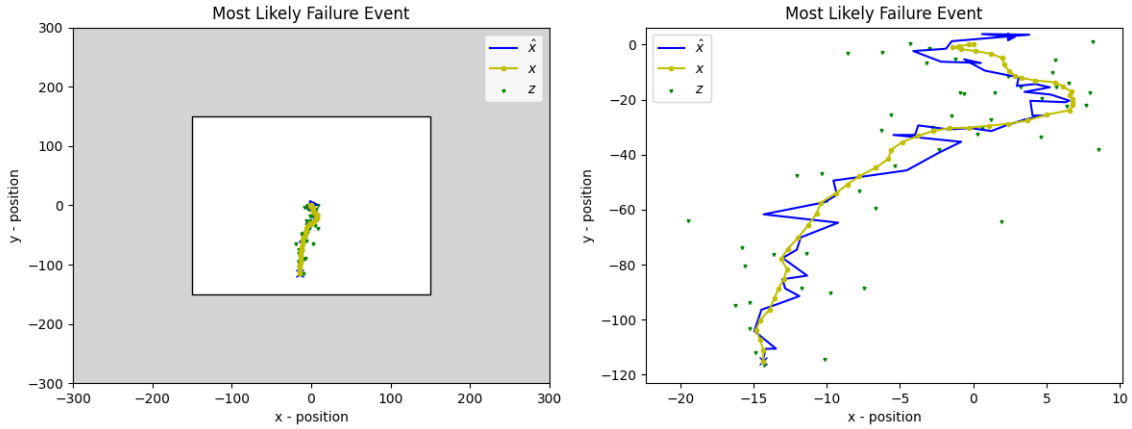
Figure 6.14: NEES time series plot with the 95% confidence interval for state estimation case 2.

In Figure 6.14 the NEES values are plottet together with the 95% confidence interval. It can be observed that from the middle of the simulation and out, a significant amount of the NEES values are above the confidence interval resulting in a fraction above of 51.43 %. However, in the end of the simulation the NEES values becomes lower making more of them be inside the confidence interval. Table 6.8 presents som key performance metrics, including ANEES result, the limits of the 95% confidence interval, RMSE for position and velocity. Observing the ANEES value and comparing it to the limits of the confidence interval, it can be observed that the ANEES value is inside the limits. This may indicate that the event found may not be the result of an overconfident filter. However, the AST method did successfully find an event with the specified test metric and threshold, which may be used as an argument to say that the method is verified for this case. The requirements initially made in the beginning of this case section, did state that AST should be able to find events where the PDA estimate shows indication of drift, which AST was able to. Hence, for that requirement the implementation is verified.

Metric	Value
RMSE Position	7.31
RMSE Velocity	1.18
ANEES	5.81
ANEES 95% Confidence Interval	[0.051, 7.378]
NEES Fractions	
Above CI	0.31
Below CI	0.0
Inside CI	0.69

Table 6.8: State Estimation Performance Measures for state estimation case 2.

Filtering Consistency Condition 2: Underconfident State Estimation The third AST run was done in order to find likely errors related to filtering consistency condition 2, which is that the state errors should have magnitude commensurate with the state covariance yielded by the filter. Hence, as mentioned the NEES fraction below the 95% confidence interval for position was chosen as test metric. Initially, a search using a threshold with a fraction of 20% below the confidence interval was performed. However, examining the results, it became clear that the magnitude of that error was not large enough for the AST search to perform at its best. Hence, the threshold was increased to 40%. After 1000 MCTS iterations the track presented in Figure 6.15 was outputted from the algorithm:



(a) (b)

Figure 6.15: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for state estimation case 3. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track.

The result of the final simulation in this case yielded a significant more jagged track than in the two previous simulation runs. The measurements was distributed with pretty large variance around the target trajectory, which may be a more realistic scenario than the two scenarios found by AST in the previous sections. The filter follows the measurements closest to the target trajectory, implying that the state estimate is not too bad. However, the estimate follows the measurement a bit too closely, implying that the process noise covariance could have been lowered a bit making the estimate more smooth.

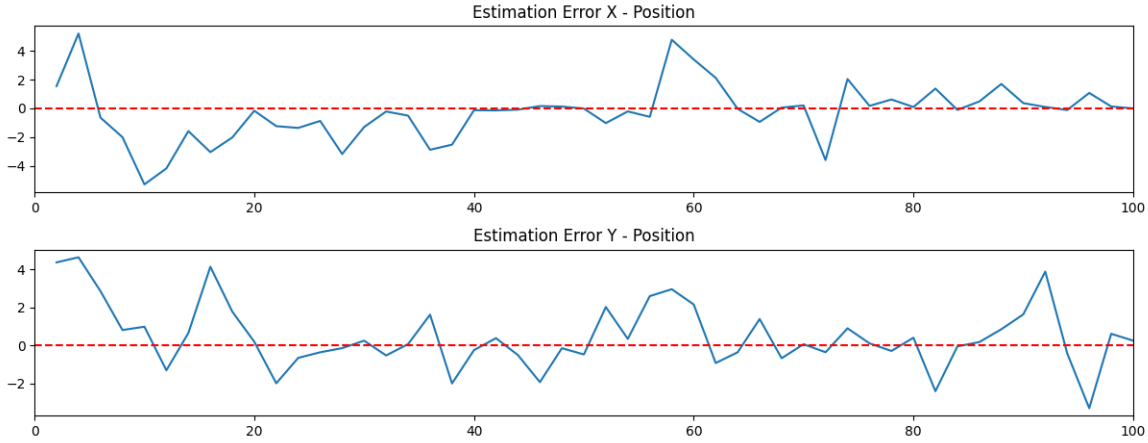


Figure 6.16: Estimation error time series for respectively x and y-position for state estimation case 3.

In Figure 6.16 it can be observed that the estimation error mostly oscillated around zero with a relatively high frequency and low magnitude for both x and x-position. Which is typical for low NEES values.

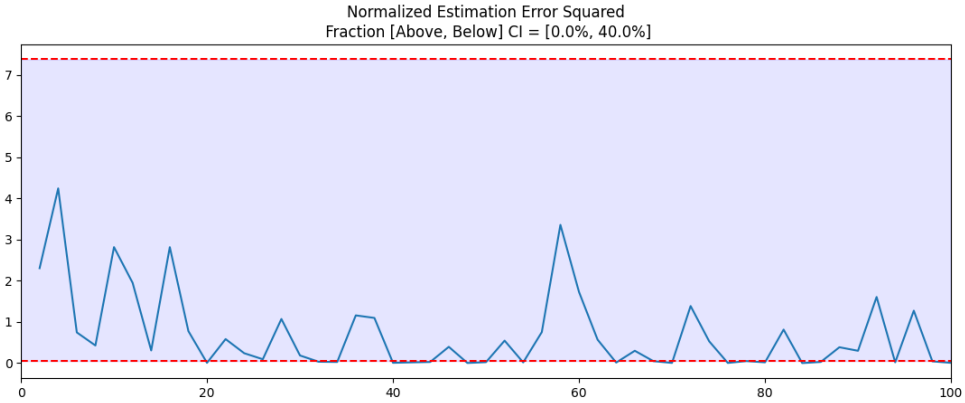


Figure 6.17: NEES time series plot with the 95% confidence interval for state estimation case 3.

Figure 6.17 presents the plot of the NEES values through the simulation. It may be hard to see visually in this plot, but when zooming in it can be observed that the NEES values was pretty low throughout the most of the simulation with a fraction of 40% under the confidence interval. However, looking at the ANEES value in Table 6.9, the higher NEES values of the simulation seems to compensate for the low ones, making the ANEES value stay inside the confidence interval. Considering these results and the third requirement for verification of the test method in this case, which was that AST should find events where the estimate is noisy and mostly follows the measurements, the method is verified. Hence, the method was considered verified for all of the state estimation requirements in tested for in this case study. Further evaluation is recommended to ensure that the method is able to find all failure event described in this case.

Metric	Value
RMSE Position	2.6
RMSE Velocity	0.573
ANEES	0.70
ANEES 95% Confidence Interval	[0.051, 7.378]
NEES Fractions	
Above CI	0.0
Below CI	0.4
Inside CI	0.6

Table 6.9: State estimation performance measures for state estimation case 3.

6.3.3 Verification Case 3: Qualitative Analysis of Adaptive Stress Testing of Single-target Tracking

The objective of this case study is to provide verification that AST may be used to find some specified common failure events in single-target tracking. This is done by testing a set of conditions presented in the first paragraph. The choice of test metrics and other AST requirements are presented in the second paragraph. In addition are the simulator setup and parameter values of the case presented.

Feature Specifications and Verification Requirements

This case is an extension of the state estimation case to a single-target case, where tracker has to deal with clutter and mis-detections. Since the state estimation case covers some elements of filter consistency, it is considered beneficial to focus on track loss in this case. Track loss is when the tracking algorithm loses track of target and the estimation drift away from the target state. This is a more common failure event in single(and multi)-target tracking than in state estimation due to the presence of clutter measurements and mis-detections. Hence, the formal specifications, which is to be verified is related to ASTs ability to find failure events in single-target tracking where track loss occurs. More specifically the goal is to demonstrate that the method can find scenarios where:

1. The estimation error continues to grow from a time step and through the rest of the simulation.
2. The estimation error exceeds a threshold at a specific time step and never goes below that threshold throughout the rest of the simulation.

These specifications are verified through the following conditions:

1. When manipulating a single-target simulation the AST implementation should be able to find scenarios where the true target state is not inside the PDAF validation gate through n consecutive steps.
2. When manipulating a single-target simulation the AST implementation should be able to find scenarios where the estimation error exceeds a minimum distance d through n time consecutive steps.

AST Simulator Requirements

The objective of this paragraph is to discuss how the simulator for this case study should fulfill the AST requirements mentioned in section 6.2.1. This will include definition of failure events and suitable distance metrics and defining what a terminal state is. The events searched for in this

case are related to the concept of track loss, which was mentioned in the previous section. Table 6.10 presents the failure definitions, test metrics and the terminal event condition used in this case. The termination is set to happen when the simulation has reached its end $t = t_{end}$. The failure detection conditions and test metrics are further specified in the next paragraphs.

Failure Event	Test Metric	Failure Detection	Distance (d) From Failure	Terminal Event
Track Loss	Validation Gate Condition	$\mathrm{n}_{steps,d_m>g^2} \geq n_{thresh}$	$\max(\mathrm{n}_{thresh} - n_{steps,d_m>g^2}, 0)$	$t = t_{end}$
	Euclidean Distance Condition	$\mathrm{n}_{steps,d_e>g^2} \geq n_{thresh}$	$\max(\mathrm{n}_{thresh} - n_{steps,d_e>g^2}, 0)$	$t = t_{end}$

Table 6.10: test metrics and failure thresholds for failure detection, when searching for events related to track loss.

Validation Gate Condition: Track Loss failure has occurred when the true target state is not inside the validation gate for a sequence of consecutive n_{thresh} steps. Mathematically this is formalized as the Mahalanobis distance between the position component of the target state x_k^{pos} and the predicted measurement Gaussian $N(\hat{z}_{k|k-1}, S_{k|k-1})$ being below the validation gate size squared g over n_{thresh} consecutive time steps:

$$e = \left\{ \text{if } d_m = (x_k^{pos} - \hat{z}_{k|k-1})^T S_{k|k-1}^{-1} (x_k^{pos} - \hat{z}_{k|k-1}) > g^2 \text{ for } n_{thresh} \text{ consecutive time steps} \right\} \tag{6.7}$$

The distance from failure is then determined by:

$$d = max(n_{thresh} - n_{steps,d_m>g^2}, 0) \tag{6.8}$$

Where $n_{steps,d_m>g^2}$ is the the number of consecutive steps where the condition does not hold.

Euclidean Distance Condition: Track Loss failure has occurred when the euclidean distance between the target position and the predicted measurement exceeds a specified threshold g^2 for a sequence of consecutive n_{thresh} steps:

$$e = \left\{ \text{if } d_e = (x_k^{pos} - \hat{z}_{k|k-1})^T (x_k^{pos} - \hat{z}_{k|k-1}) \leq g^2 \text{ for } n_{thresh} \text{ consecutive time steps} \right\} \tag{6.9}$$

The distance from failure is then determined by:

$$d = max(n_{thresh} - n_{steps,d_2>g^2}, 0) \tag{6.10}$$

Where $n_{steps,d_2>g^2}$ is the the number of consecutive steps where the condition does not hold.

Simulator Setup

The simulator in this case is as mentioned set up to produce single-target simulations consisting of only a single target, detected measurements originating from that target and clutter measurements. How this is done is illustrated in Figure 6.18. The target arrival and departure modules are tuned such that they do not create new targets or delete existing ones. All the components which are limited by the tuning are faded. Hence, the resulting output of the simulator is the output created by the simulator components enclosed by the box denoted Single-target Simulator. The parameter values for the simulator are presented in the next section.

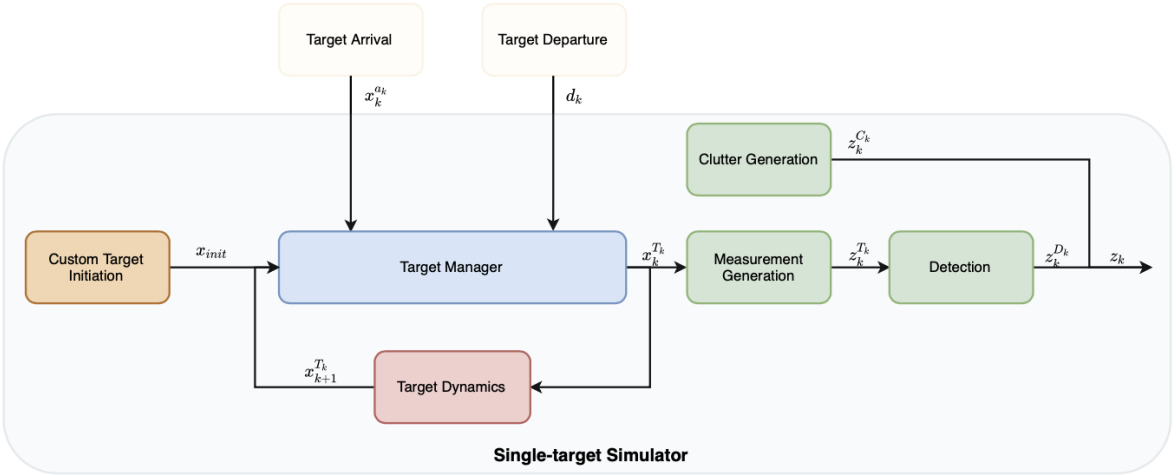


Figure 6.18: Illustration of the single-target simulator used in this case.

Parameter Values

Environment Simulator Parameters The simulator parameters for this case presented in Table 6.11 are set to the same values as in the previous case for the parameters related to state estimation. The parameters related to target tracking are adjusted in order for the simulator to generate realistic single-target scenarios. This is done by adjusting the detection probability to be $P_D = 0.8$ and the expected number of clutter measurements at each time step to be $\Lambda_c = 5$. This number could probably be higher, but it was sufficient enough for AST to find some interesting scenarios. The expected number of arriving targets is still set to zero, in addition to the departure probability being set to zero. This is done since the goal is to evaluate a single-target scenario without initiation and termination. In addition the starting point is changed for no specific reason.

$t_{terminal}$	x_{init}	Ts	σ_a	σ_z	P_D	Λ_c	Λ_a	P_d	Surveillance Area
100	$[-75, -75, 0, 0]$	2	0.2	5	0.8	5	0.0	0.0	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$

Table 6.11: Simulator parameter values for the single-target verification case. The parameters are explained in Chapter 5.

System Under Test Parameters The parameters of the PDAF algorithm was according to the tuning strategy set to correspond with the parameter values of the environment simulator, similar to in the previous case. The main difference from the previous case is that the probability of detection is set to $P_D = 0.8$ and the gating probability is set to $P_G = 0.8$. The clutter density was set to $5/(300^2)$, which corresponds to the number of expected clutter measurements created by the simulator at each time step divided by the surveillance area. The parameter values of the tracker can be observed in Table 6.29

σ_a	σ_z	P_D	P_G	Clutter Density	Ts
0.2	5	0.8	0.8	$5/(300^2)$	2

Table 6.12: Target tracker parameter values for the single-target verification case. The parameters are explained in Chapter 5.

Test Method Parameters The AST parameters was set to be equal to the parameters of the previous case, since the performance of the implementation seemed to work well in this case as well.

N.O. Iterations	UCB Constant c	PW Constant c	PW Constant α	Likelihood Weight	Failure Weight
1000	0.1	2	0.25	1	1e4

Table 6.13: Simulator parameter values for the single-target verification case. The parameters are explained in Chapter 5.

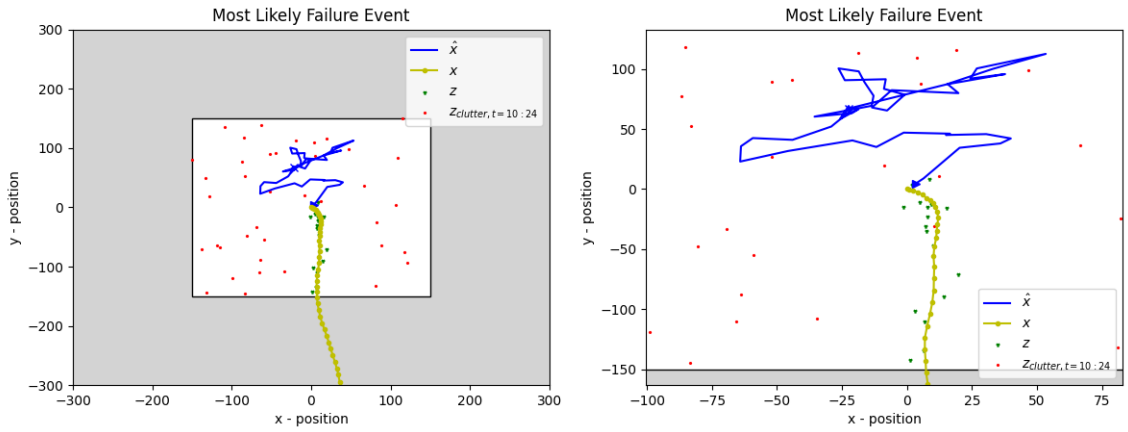
Critic Parameters In this case the Critic parameters is set based on the failures that is searched for, which is determined track loss condition 1 and 2. First, the PDAF tracker is tested for track loss using the validation gate condition, which is condition 1. Second, the tracker is tested for track loss according to the condition based on euclidean distance, which is condition 2. See Section 6.2.1 for further details about the parameters.

Case	Failure Threshold	Distance Metric
Validation Gate Loss Condition	$n_{steps} = 9$	d = max(9 - N_{steps}, 0)
Euclidean Distance Loss Condition	$n_{steps} = 15$	d = max(15 - N_{steps}, 0)

Table 6.14: Critic Configuration, N_{steps} denotes the number of consecutive steps where the loss condition is not met, and n_{steps} denotes limit of the consecutive number of steps for the scenario to be considered a failure event e.

Results and Discussion

Validation Gate Loss Condition The first AST run in this case was done in order to find likely errors related to the first track loss condition using the validation gate. This means that if the PDAF does not gate the true target state in n_{steps} consecutive steps a failure event has occurred. After 1000 MCTS iterations the track presented in Figure 6.19 was outputted from the algorithm:



(a) (b)

Figure 6.19: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for single-target tracking case 1. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step where the true target state is not inside the validation gate.

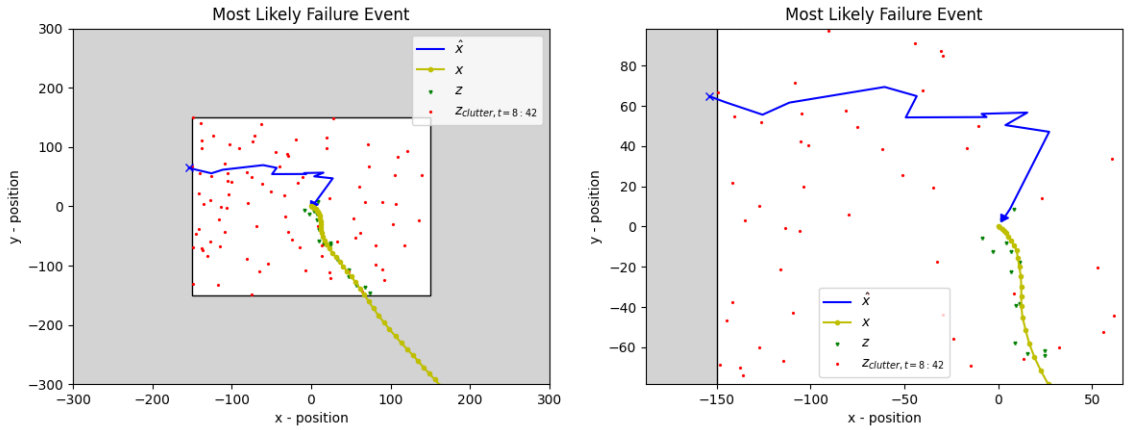
It can be observed that the track drifts off at the in the beginning of the simulation, and never follows the true target trajectory. The reason for this happening might be that in the beginning of the simulation clutter measurements are generated close to the true target measurements, making

the track drift off and gate more clutter measurements. In Table 6.15 some key performance indicators are presented. It can be observed that the RMSE was significantly higher than when running the state estimation case. The same can be said for the ANEES value and the fraction of NEES values above the confidence interval. According to the validation gate loss condition the validation gate failed to gate the true target state in 14 consecutive steps. Observing the track, it would be expected that the number of steps with track loss was more than 14, since the track is off the entire simulation and never leaves the surveillance area. In Chapter 5 when describing the validation gate, it is mentioned that the Mahalanobis distance between the measurement and the predicted measurement was used in the gating condition, see (5.11). However, the trouble of using this distance metric is that when the covariance estimate for the predicted measurement is poor, the covariance matrix may cancel out large differences between the predicted measurement and for this case the target state. Hence, the filter gates the target state even though the estimate is far off. Noticing this effect is the main reason for using the euclidean distance in the next AST run. However, even though the validation gate condition doesn't capture the complete essence of track loss, it was a good enough test metric to find a track loss scenario. The test method did pass the first requirement for verification of the test method´s ability to find scenarios related to track loss, where the true target state is not inside the PDAF validation gate through n consecutive time steps. Hence, the method is verified in this case.

Metric	**Value**
RMSE Position	75.6
RMSE Velocity	3.36
GOSPA	49.03
Loss	14
ANEES	16.88
ANEES 95% Confidence Interval	[0.051, 7.378]
NEES Fractions	
Above CI	1.0
Below CI	0.0
Inside CI	0.0

Table 6.15: State estimation performance metrics for single-target tracking case 1

Euclidean Distance Loss Condition The second AST run in this case was done in order to find likely errors related to the second track loss condition using the euclidean distance condition. This means that if euclidean distance between the PDAF estimate and the true target state is larger than a specified threshold for n_{steps} consecutive steps, a failure event has occurred. After 1000 MCTS iterations the track presented in Figure 6.20 was outputted from the algorithm:



(a) (b)

Figure 6.20: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for single-target tracking case 2. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step where track loss is detected.

The result of this case is similar to the one of the previous, where it can be observed that the track drifts off at the in the beginning of the simulation, and never follows the true target trajectory. In the same manner as for the validation gate condition the reason for this happening might be that in the beginning of the simulation clutter measurements are generated close to the true target measurements, making the track drift off and gate more clutter measurements. However, the main difference in this case is that the number of steps recorded with track loss in this case was 34, which was significantly higher than for the previous case. However, examining the track it can be observed that the number of steps with loss should be approximate the same as for the previous case, meaning that the Euclidean distance condition did capture this in a better manner than the validation gate condition. The reason why it is not recorded track loss after 42 time steps is that the track is terminated due to leaving the surveillance area. For this case the RMSE and ANEES values even higher than for the previous case. This can be observed in Table 6.16. Hence, using the second track loss condition may be better when searching for track loss scenarios than using the first condition. Considering the results the test method meets the second verification requirement for this case study. Hence, the method's ability to find likely events with track loss is verified.

Metric	**Value**
RMSE Position	117.46
RMSE Velocity	5.14
GOSPA	40.1
Loss	18
ANEES	211.67
ANEES 95% Confidence Interval	[0.051, 7.378]
NEES Fractions	
Above CI	1.0
Below CI	0.0
Inside CI	0.0

Table 6.16: State estimation performance metric for single-target tracking case 2

6.3.4 Verification Case 4: Qualitative Analysis of Adaptive Stress Testing of Track Initiation

The objective of this case study is to provide verification that AST is able to find failure events in target tracking related to track initiation. This is done by defining a set of specifications which is verified by a set of presented conditions.

Feature Specifications and Verification Requirements

The feature specifications to be verified in this case study are related to track initiation, which is a key component of target tracking. The simulation is set up by not initializing the PDAF algorithm with the prior distribution of the target state, and by using the M/N logic presented in Chapter 5 to handle the track initiation. More specifically the specifications are that AST should be able to find common failure events in track initiation related to:

1. Slow initiation of tracks.
2. Failing to initiate tracks.

The specifications are verified in this case study through the following conditions:

1. When manipulating a single-target simulation the AST implementation should be able to find scenarios the PDAF algorithm with the M/N extension for track initiation struggles to initiate the track quickly.
2. When manipulating a single-target simulation the AST implementation should be able to find scenarios the PDAF algorithm with the M/N extension for track initiation fails to initiate the track.

AST Simulator Requirements

The objective of this section is to present how the simulator for this case study should fulfill the AST requirements mentioned in Section 6.2.1. This will include definition of failure events and suitable distance metrics and defining what a terminal state is. The events searched for in this case are related to the concept of track initiation, which was mentioned in the paragraph above. Table 6.17 presents the failure events search for in this case, in addition to failure detection conditions and test metrics. The events and metrics are further specified in the next paragraphs.

Failure Event	Test Metric	Failure Detection	Distance (d) From Failure	Terminal Event
Slow Initiation	Steps Before Initiation	$\mathrm{n}_{steps} \geq n_{thresh}$	$\max(\mathrm{n}_{thresh} - n_{steps}, 0)$	$t = t_{end}$
Initiation Failure	N.O. Targets vs. Tracks ($\mathrm{t} = \mathrm{t}_{end}$)	$\mathrm{n}_{targets} \geq n_{tracks}$ and $x_t^i \in [-150, 150]$	$\max(\mathrm{n}_{tracks}, 0)$	$t = t_{end}$

Table 6.17: Test metrics and failure thresholds for failure detection, when searching for events related to track initiation.

Slow Initiation: Slow track initiation failure has occurred when the number of steps n_{steps} from the target is initiated to the track is initiated exceeds a specified threshold n_{tresh}:

$$e = \{n_{steps} > n_{thresh}\} \tag{6.11}$$

The distance from failure is then determined by:

$$d = max(n_{thresh} - n_{steps}, 0) \tag{6.12}$$

Initiation Failure: Failing to initiate has occurred when the number of tracks n at the simulation end is less than the number of targets N inside the surveillance area $[-150, 150]$ at the simulation

end:

$$e = \left\{ n_{targets} > n_{tracks} \text{ and } x_t^i \in [-150, 150] \right\} \tag{6.13}$$

Where x_t^i denotes element i of the state vector x_t, and $i \in [0, 1]$, meaning that only the position is considered. The distance from failure is then determined by:

$$d = \begin{cases} max((n_{tracks} + 1) - n_{targets}, 0), \text{ if } x_t^i \in [-150, 150] \wedge t = t_{end} \\ \infty, \text{ if } x_t^i \notin [-150, 150] \wedge t = t_{end} \end{cases} \tag{6.14}$$

However, in this single-target simulation it is only necessary to detect whether there exist an track or not in the entire simulation. The decision rule is then:

$$e = \{1 > n_{tracks}\} \tag{6.15}$$

Where n_{tracks} is the number of tracks through the whole simulation. The distance from failure is then determined by:

$$d = max(n_{tracks}, 0) \tag{6.16}$$

In addition is the termination is set to happen when the simulation has reached its end $t = t_{end}$.

Simulator Setup

The simulator setup in this case is similar to the setup of the previous case, but with target departure added when testing the second condition regarding failing to initiate. The reason that this is done is that the tracker will be tested for failing to initiate tracks before they departure. See Figure 6.21 for an illustration of this setup.

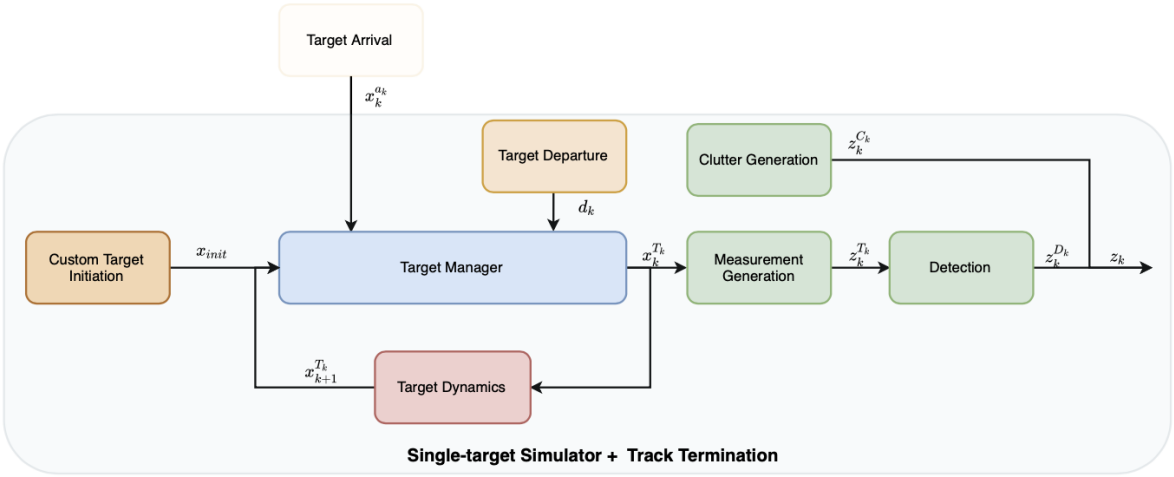


Figure 6.21: Illustration of the single-target simulator with termination of targets used in this case.

Tuning Values

Environment Simulator Parameters The simulator parameters for this case presented in Table 6.18 are set to the same values as in the previous case, with the exception of the departure probability for when testing the second failure condition.

Case	$t_{terminal}$	x_{init}	Ts	σ_a	σ_z	P_D	Λ_c	Λ_a	P_d	Surveillance Area
Slow Initiation	100	$[-75, -75, 0, 0]$	2	0.2	5	0.8	5	0.0	0.0	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$
Failing to Initiate	100	$[-75, -75, 0, 0]$	2	0.2	5	0.8	5	0.0	5e-3	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$

Table 6.18: Simulator parameter values for the track initiation verification case. The parameters are explained in Chapter 5. Slow initiation, failing to initiate and clutter tracks refers to respectively failure event 1, 2 and 3 described in the previous paragraph.

System Under Test Parameters The parameters of the PDAF algorithm was set in the same manner as in the previous case, but with the additional parameters related to the M/N logic for track initiation. M is as explained in Chapter 5 the required number of steps with gated measurements of totally N time steps before the track is initiated. Initiation threshold is the required threshold for euclidean distance a tentative track has to be from an existing track in order for it to be initiated. This is done in order to avoid initiation of the same track several times. The parameter values of the tracker can be observed in Table 6.19.

σ_a	σ_z	P_D	P_G	Clutter Density	Ts	M	N	Initiation Threshold
0.2	5	0.8	0.8	$5/(300^2)$	2	4	5	1.0

Table 6.19: Target tracker parameter values for the track initiation verification case. The parameters are explained in Chapter 5.

Test Method Parameters The AST parameters was set to be equal to the parameters of the previous cases, since the performance of the implementation seemed to work well in this case as well. See Table 6.20 for the parameter values.

N.O. Iterations	UCB Constant c	PW Constant c	PW Constant α	Likelihood Weight	Failure Weight
1000	0.1	2	0.25	1	1e4

Table 6.20: Simulator parameter values for the track initiation verification case. The parameters are explained in Chapter 5.

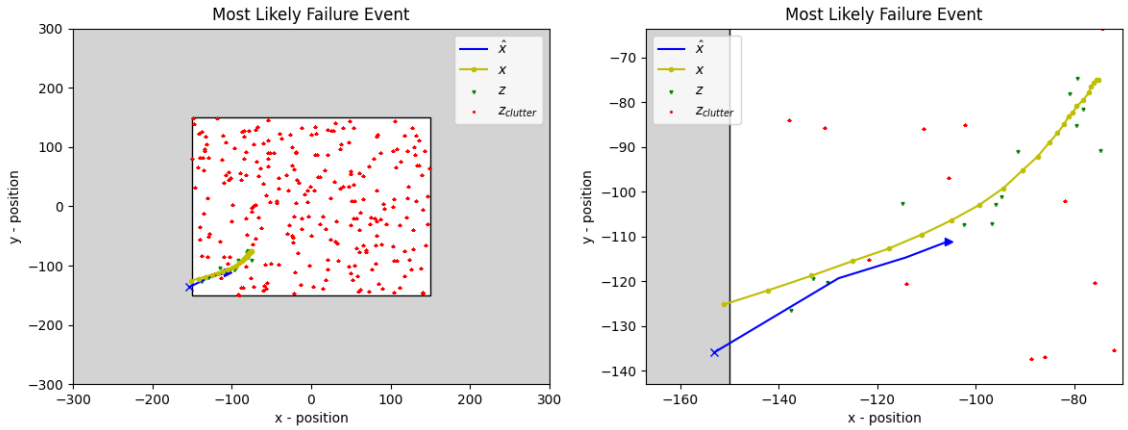
Critic Parameters In this case the Critic parameters was set based on what failures were searched for, slow track initiation and failing to initiate. The parameter values are presented in Table 6.21

Case	Failure Threshold	Distance Metric
Slow Initiation Condition	$n_{steps} = 15$	d = max(15 - N_{steps}, 0)
Initiation Failure Condition	$n_{tracks} = 1$	d = max(n_{tracks}, 0)

Table 6.21: Critic Configuration, N_{steps} denotes the number of consecutive steps from the target is initiated, and n_{steps} denotes limit of steps without initiation before the scenario is to be considered a failure event e.

Results and Discussion

Slow Initiation The first AST run in this case was done in order to find likely errors related to slow initiation of tracks. This means that if the tracker haven't initiated a track in n_{steps} from a target has been initiated, a failure event e has occurred. After 1000 MCTS iterations the track presented in Figure 6.22 was outputted from the algorithm:

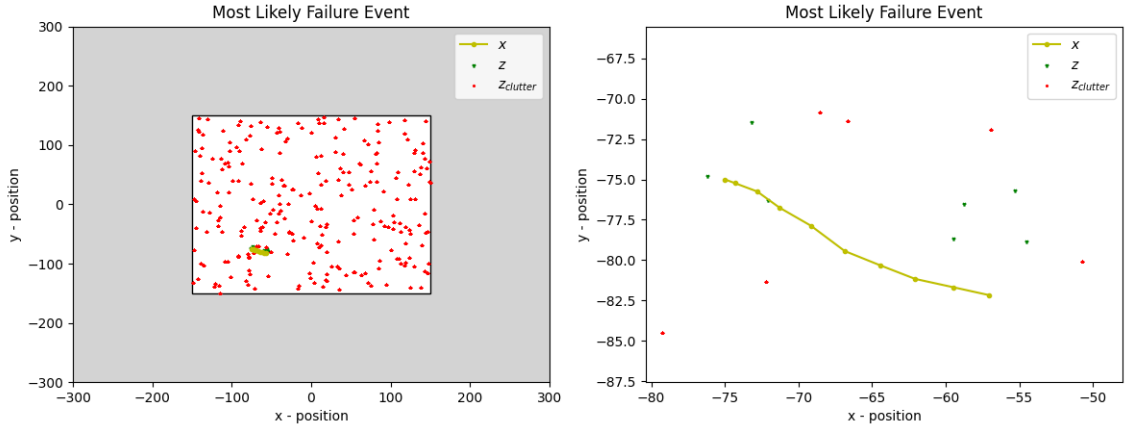


(a) (b)

Figure 6.22: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for track initiation case 1. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step.

The tracker struggles to initiate the target through 17 time steps. After the initiation the track manages to follow the target until termination when leaving the surveillance area. Initiation after 17 time steps is a significantly slow initiation. In terms of the objective of the case, the result is satisfying, ASTs ability to find scenarios with slow track initiation has been demonstrated. Hence, the test method's ability to find failure events regarding slow track initiation is verified.

Initiation Failure The second AST run in this case was done in order to find likely errors related to failing to initiate tracks. This means that if the tracker doesn't manage to initiate a track before the track is terminated or the simulation run is over. After 1000 MCTS iterations the track presented in Figure 6.23 was outputted from the algorithm:



(a) (b)

Figure 6.23: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for track initiation case 2. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step.

It can be observed that the target terminates after 10 time steps, giving the tracker a relatively short time frame to initiate the track. With a requirement of gating 4 measurements through 5 consecutive time steps, the tentative tracks are not made tracks in this case. Considering the short trajectory and life time of the target, and the tracker configurations, this seems to be a realistic failure scenario. Finding such a scenario was the goal of the simulation. Hence, the test method is verified in terms of finding scenarios where the tracker is not able to initiate a track.

6.3.5 Verification Case 5: Qualitative Analysis of Adaptive Stress Testing of Track Termination

The objective of this case study is to provide verification that AST is able to find failure events related to track termination. First, are the feature specifications to be verified and their respective conditions for verification presented. Second, how the AST simulator requirements are fulfilled is presented. Then, the simulator setup and the parameter values are presented, before the results are presented and discussed.

Feature Specifications and Verification Requirements

This case is very similar with the previous case of track initiation. The difference is that in this case the objective is to demonstrate that AST is able to find errors related to track termination rather than initiation. The simulation is set up by initializing the PDAF algorithm with the prior distribution of the state for a single target, and by using the M/N logic presented in Chapter 5 to handle the track termination. In addition the target may departure the simulation, making the simulation contain zero targets at some point of the simulation. Then, the feature specifications to be verified is ASTs ability to find failure events related to:

1. Slow termination of tracks.
2. Failing to terminate tracks.
3. Terminating tracks that still exists.

The specifications are verified through the following conditions:

1. When manipulating a single-target simulation the AST implementation should be able to find scenarios the PDAF algorithm with the M/N extension for track termination struggles to terminate the track quickly.
2. When manipulating a single-target simulation the AST implementation should be able to find scenarios the PDAF algorithm with the M/N extension for track termination fails to terminate the track of a departed target.
3. When manipulating a single-target simulation the AST implementation should be able to find scenarios the PDAF algorithm with the M/N extension for track termination, terminates tracks for existing targets.

AST Simulator Requirements

The objective of this section is to discuss how the simulator for this case study should fulfill the AST requirements mentioned in Section 6.2.1. This will include definition of failure events and suitable distance metrics and defining what a terminal state is. The events searched for in this case are as mentioned related to the concept of track termination. The failure events searched for are presented in Table 6.22, with specified failure detection conditions and test metrics. The events and metrics are described in the next paragraphs.

Failure Event	Test Metric	Failure Detection	Distance (d) From Failure	Terminal Event
Slow Initiation	Steps Before Termination	$n_{steps} \geq n_{thresh}$	$\max(n_{thresh} - n_{steps}, 0)$	$t = t_{end}$
Failing to Terminate	N.O. Targets vs. Tracks (t = t_{end})	$n_{tracks} \geq n_{targets}$	$\max((n_{targets} + 1) - (n_{tracks}), 0)$	$t = t_{end}$
Terminating Tracks of Existing Targets	N.O. Targets vs. Tracks (t = t_{end})	$n_{targets} \geq n_{tracks}$ and $x_t^i \in [-150, 150]$	d_{TET}	$t = t_{end}$

Table 6.22: Test metrics and failure thresholds for failure detection, when searching for events related to track termination.

Slow Termination: Slow track termination failure has occurred when the number of steps n_{steps} from the target is terminated to the track is terminated exceeds a specified threshold:

$$e = \{n_{steps} > n_{thresh}\} \tag{6.17}$$

The distance from failure is then determined by:

$$d = max(n_{thresh} - n_{steps}, 0) \tag{6.18}$$

Failing to Terminate: Failure to terminate is defined to be when the number of tracks at the simulation end exceeds the number of targets.

$$e = \{n_{tracks} > n_{targets} \text{ and } t = t_{end}\} \tag{6.19}$$

The distance from failure is then calculated according to

$$d = max((n_{targets} + 1) - (n_{tracks}), 0) \tag{6.20}$$

Terminating Existing Targets: When terminating existing targets which is inside the surveillance area, an failure event has occurred. This is determined by examining the number of targets $n_{targets}$ and tracks n_{tracks} inside the surveillance area $[-150, 150]$ at the simulation end. Hence, the failure event is determined by the following boolean condition:

$$e = \left\{n_{targets} > n_{tracks} \text{ and } x_t^i \in [-150, 150]\right\} \tag{6.21}$$

Where x_t^i denotes element i of the state vector x_t, and $i \in [0, 1]$, meaning that only the position is considered. The distance from failure is then determined by:

$$d_{TET} = \begin{cases} max((n_{tracks} + 1) - n_{targets}, 0), \text{ if } x_t^i \in [-150, 150] \wedge t = t_{end} \\ \infty, \text{ if } x_t^i \notin [-150, 150] \wedge t = t_{end} \end{cases} \tag{6.22}$$

Simulator Setup

The simulator setup in this case is similar to the setup of the previous case. For testing of the first two conditions mentioned above, the simulator is a single-target simulator with termination. For the last case the simulator is a regular single-target simulator.

Tuning Values

Environment Simulator Parameters The simulator parameters for this case presented in Table 6.23 are set to the same values as in the previous case when running a single-target simulator with target departure, except for the last simulation run where the target is never departed.

Case Name	$t_{terminal}$	x_{init}	Ts	σ_a	σ_z	P_D	Λ_c	Λ_a	P_d	Surveillance Area
Slow Termination	100	$[-75, -75, 0, 0]$	2	0.2	5	0.8	5	0.0	5e-3	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$
Failing to Terminate	100	$[-75, -75, 0, 0]$	2	0.2	5	0.8	5	0.0	5e-3	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$
Terminating Tracks of Existing Targets	100	$[-75, -75, 0, 0]$	2	0.2	5	0.8	5	0.0	0	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$

Table 6.23: Simulator parameter values for the track termination verification case. The parameters are explained in Chapter 5.

System Under Test Parameters The parameter values of the tracker was set to be the same as in the previous case. The values can be observed in Table 6.24.

σ_a	σ_z	P_D	P_G	Clutter Density	Ts	M	N	Initiation Threshold
0.2	5	0.8	0.8	$5/(300^2)$	2	4	5	1.0

Table 6.24: Target tracker parameter values for the track termination verification case. The parameters are explained in Chapter 5.

Test Method Parameters The AST parameters was set to be equal to the parameters of the previous cases when searching for events related to termination failure and termination of tracks for existing targets. This was done since the performance of the implementation seemed to work well in these cases as well. However, for the case searching for events with slow termination, the progressive widening constant α was increased in order to make search space wider, making the algorithm explore a larger variety of seed-actions. This means that the MCTS algorithm was tuned to weight exploration more. The parameter values can be observed in 6.25.

Case Name	N.O. Iterations	UCB Constant c	PW Constant c	PW Constant α	Likelihood Weight	Failure Weight
Slow Termination	1000	0.1	2	2/3	1	1e4
Failing to Terminate	1000	0.1	2	1/4	1	1e4
Terminating Tracks of Existing Targets	1000	0.1	2	1/4	1	1e4

Table 6.25: Simulator parameter values for the track termination verification case. The parameters are explained in Chapter 5.

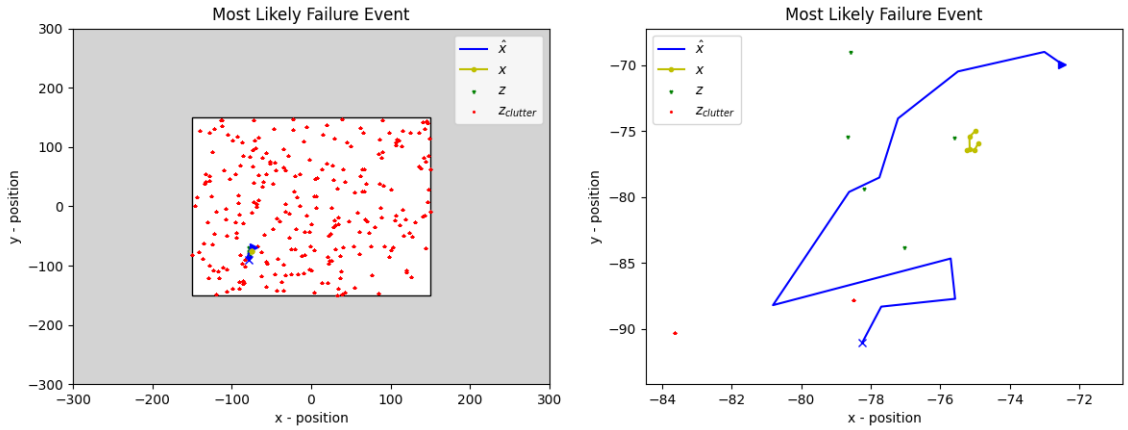
Critic Parameters In this case the Critic parameters was set based on what failures were searched for, which was events related to termination of tracks. Table 6.26 presents the critic parameters of this case.

Case	Failure Threshold	Distance Metric
Slow Termination	$n_{steps} = 7$	d = max(7 - N_{steps}, 0)
Failing to Terminate	d=0	d = max(($n_{targets} + 1$) − (n_{tracks}), 0)
Termination Tracks of Existing Targets	d=0	d = $\begin{cases} max((n_{tracks}+1) - n_{targets}, 0), \text{ if } x_t^i \in [-150, 150] \wedge t = t_{end} \\ \infty, \text{ if } x_t^i \notin [-150, 150] \wedge t = t_{end} \end{cases}$

Table 6.26: Critic Configuration, N_{steps} denotes the number of consecutive steps from the target is terminated, and n_{steps} denotes limit of steps without termination before the scenario is to be considered a failure event e.

Results and Discussion

Slow Termination The AST run in this case was done in order to find likely errors related to slow termination of tracks. This means that if the tracker haven't terminated a track in n_{steps} from a target has been terminated, a failure event e has occurred. After 1000 MCTS iterations the track presented in Figure 6.24 was outputted from the algorithm:

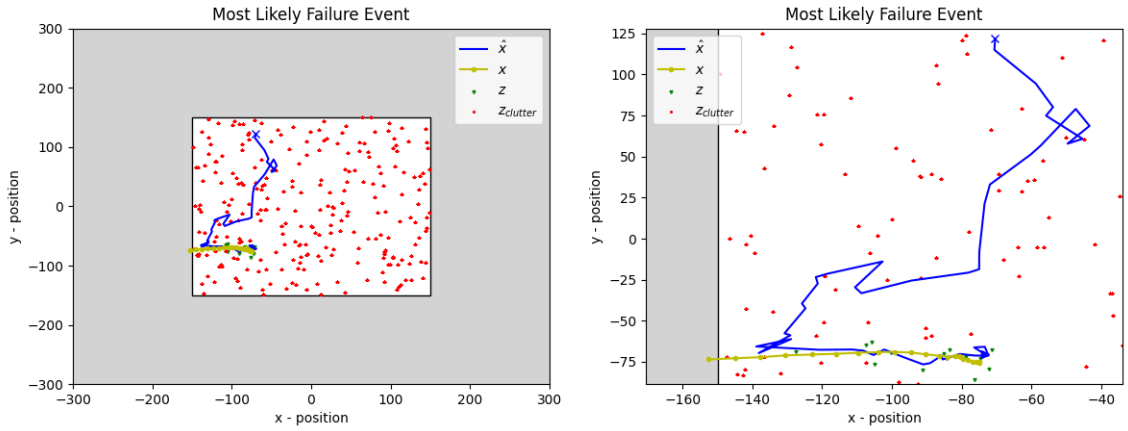


(a) (b)

Figure 6.24: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for track termination case 1. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step

It can be observed that the target exist for just a few time steps before it is departed. The detected measurements originating from the target are distributed with relatively large variance, compared to the target trajectory. However, the plot in Figure (b) of Figure 6.24 is significantly magnified in compared to (a). This means that the track is relatively short in terms of the scale which the simulator and tracker is tuned for. Further, it can be observed that the track outputted from the AST implementation struggles to terminate in 8 time steps after the target has departed the simulation. This is most likely due to the two clutter measurements in close proximity of the track. Hence, the tracker does most likely gate the two clutter measurements a few time steps after the departure of the target, making the M number of gated measurements at the time steps after the departure increase, such that the M is greater than the lower threshold required for the track to continue. Even though the tracker struggles to terminate the track quickly, the track is still short relative to the surveillance are, implying that the failure may not be of high severity. However, the key objective of this case was to verify that the AST implementation is able to find failures where the PDAF algorithm with the M/N extension struggles to terminate the track quickly. This objective was achieved in this simulation. Hence, the test method is verified according to its the first track termination specification.

Failing to Terminate The AST run in this case was done in order to find likely events where the tracker fails to terminate tracks. This means that if the tracker haven't terminated a track in in the entire simulation when the target has departed, a failure event e has occurred. After 1000 MCTS iterations the track presented in Figure 6.25 was outputted from the algorithm:

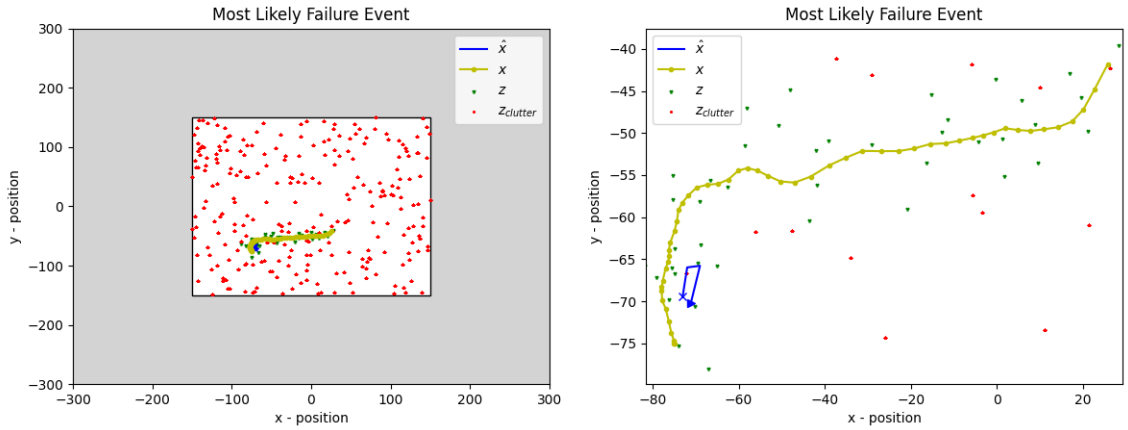


(a) (b)

Figure 6.25: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for track termination case 2. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step

It can be observed that the target departures the simulation by leaving the surveillance area. The track struggles to follow the target when the target get close to departure. This may be due to the target not producing measurements for several time steps, when moving past $x = -110$. In addition several clutter measurements are produced in close proximity of the target, making the track drift of from the target and continue the gating more clutter measurements, resulting in failing to terminate the track when the target departures the simulation. This is also a track loss scenario similar to the results presented in the track loss case. In the track loss case it was mentioned that a tracker algorithm dealing with track existence could avoid such failure by terminating the track and initiating a new track following the target closer. However, in this case it was demonstrated that such an error may occur with such a tracker as well. Further, the objective of this case was to test the second verification condition, which states that the AST implementation should be able to find scenarios where the PDAF algorithm with the M/N extension fails to terminate the track of a departed target. This was the case. Hence, the test method's ability to find events related to termination failure was verified in this case.

Terminating Tracks for Existing Targets The AST run in this case was done in order to find likely errors related to termination of tracks for existing targets. This means that if the tracker terminates a track which never departures the simulation, a failure event e has occurred. After 1000 MCTS iterations the track presented in Figure 6.26 was outputted from the algorithm:



(a) (b)

Figure 6.26: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for track termination case 2. The area shaded gray is the outside of the surveillance area. (b) shows a zoomed in version of the same plot. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step.

It can be observed that the track terminates after a few time steps, while the target continues to exist for the entire simulation. This result may be considered strange, since the track in the plot is surrounded by measurements, which should imply that the validation gate should accept several measurements, making the track continue. However, the plot does not provide a complete picture of the simulation, since the measurements presented are the measurements generated through the entire simulation. Hence, the measurements close to the track was probably generated at earlier time steps, explaining why the track didn't gate them. Looking very closely it can be observed that a clutter measurement is generated close to the track, which may be the reason that the track makes a U-turn. This may have made the track struggle to keep up with the target, which in turn made the track not gate the target measurements. Further, the objective of this case was to test the third verification condition for searching for termination failure. The condition state that the AST implementation should be able to find scenarios where the PDAF algorithm with the M/N extension terminates tracks for existing targets. This was demonstrated with the presented result. Hence, the implementation was verified for this case.

6.3.6 Verification Case 6: Qualitative Analysis of Adaptive Stress Testing of Tracking Multiple Targets

The objective of this case study is to provide verification that AST is able to find failure events in target tracking when simulating and tracking multiple targets. First a set of feature specifications are presented, in addition to a set of conditions for verifying the requirements. Then, the AST simulator requirements and the simulator setup with parameter values are presented. Finally, the results of the case are presented and discussed.

Feature Specifications and Verification Requirements

This case is an extension of all of the previous cases, where the tracker has to deal with the same environment with the addition of the simulator generating multiple targets. The previous cases covers different failure events when tracking a single target. This case will examine AST's ability to find some of the same type of errors, and maybe new ones, when applying a more general test metric. Then, the feature specifications to be verified is AST's ability to find failure events related to some of the following phenomenons:

1. Track Loss
2. Track Initiation Failure:
   (a) Slow initiation of tracks
   (b) Failing to initiate tracks
3. Termination Failure:
   (a) Slow termination of tracks
   (b) Failing to terminate tracks
   (c) Terminating tracks that still exists

The specifications are verified by meeting the following conditions, which are mentioned in previous cases as well:

1. When manipulating a multi-target simulation the AST implementation should be able to find scenarios where the PDAF track(s) drifts away from the target.
2. When manipulating a multi-target simulation the AST implementation should be able to find scenarios where the PDAF tracking algorithm with the M/N extension for track existence struggles to quickly initiate a track for an existing target.
3. When manipulating a multi-target simulation the AST implementation should be able to find scenarios where the PDAF tracking algorithm with the M/N extension for track existence fails to initiate a track for an existing target.
4. When manipulating a multi-target simulation the AST implementation should be able to find scenarios where the PDAF tracking algorithm with the M/N extension for track existence struggles to quickly terminate a track for a departed target.
5. When manipulating a multi-target simulation the AST implementation should be able to find scenarios where the PDAF tracking algorithm with the M/N extension for track existence terminates tracks for existing targets.

AST Simulator Requirements

The objective of this section is to discuss how the simulator for this case study should fulfill the AST requirements mentioned in section 6.2.1. This will include definition of failure events and suitable distance metrics and defining what a terminal state is. Table 6.27 presents the test metric, failure definition, distance form failure and termination condition for this case.

Test Metric	Failure Detection	Distance (d) From Failure	Terminal Event
GOSPA	GOSPA $\geq GOSPA_{thresh}$	$\max(\text{GOSPA}_{thresh} - GOSPA, 0)$	$t = t_{end}$

Table 6.27: Test metrics and failure thresholds for failure detection, when searching for failure events in a multi target scenario.

The events searched for in this case are as mentioned related to track loss, and track initiation and termination. These are events searched for in previous cases. However, the test metrics used in the previous cases are specified for tracking of single targets, and it may be proven difficult to apply them to a multi-target scenario. Hence, a test metric generalized for tracking of multiple targets are sought for. For this case the Generalized Optimal Sub-pattern Assignment (GOSPA) metric presented in [44]. GOSPA compares the number of tracks and targets, and classifies false tracks (tracks not belonging to targets), undetected targets and matching tracks and targets. GOSPA is calculated through the following procedure:

- Finding the optimal assignment between sets:
  - Pairs of tracks and targets are left unassigned if the distance between the pairs exceeds a specified limit $d(x, y) > c$. Unassigned elements are referred to false/missed targets.
  - Otherwise the pairs are assigned.
- The cost of a assigned pair x and y is denoted by $d(x, y)$.
- Unassigned elements are given the cost $\frac{c}{2}$.

Then, the GOSPA value is expressed by:

$$GOSPA = \left[\min_{\gamma \in \Gamma} \left(\sum_{(i,j) \in \gamma} d(x_i, y_i)^p + \frac{c^p}{2} \left(|X| - |\gamma| + |Y| - |\gamma| \right) \right) \right]^{1/p} \tag{6.23}$$

where X is the set of targets, Y is the set of estimates and Γ is the set of possible assignments. The parameter p is used to penalize outliers, meaning that larger value of p, the more the outliers are penalized. For this case study the tuning parameters is set to $p = 2$ and $c = 8$. Simplified the GOSPA express the following:

$$GOSPA = \text{Localization Error} + \frac{c}{2}(n_{mt} - n_{ft}) \tag{6.24}$$

where n_{mt} refers to number of missed targets and n_{ft} refers to number of false targets.

Simulator Setup

For this case two different simulator setups are used. The firs setup is similar to the single-target setup presented in Figure 6.18, but instead of only initializing a single-target, two targets are initiated. This setup is used to create a simple multi-target scenario with less complexity than a scenario with target arrival and departure. The second simulator setup is the complete multi-target simulator described in Chapter 5, which has target arrival and departure in addition to the the components of the first setup.

Parameter Values

Environment Simulator Parameters The simulator parameters for this case presented in Table 6.28 are set to the same values as in some of the previous cases. For the first simulator setup the parameter values are set to be the same as for the single-target tracking case, but with the addition of an extra target, such that the initial target states are set to be $x_{0,0} = [-75, -75, 0, 0]$ for the first target, and $x_{0,1} = [75, 75, 0, 0]$ for the second target. For the second simulator setup the following extensions are made:

- The arrival density is set to be $\Lambda_a = 5e - 2$.
- The departure probability was set to be the same as for the track termination case where the departure component was used, $P_d = 5e - 3$.

´

Simulator Setup	$t_{terminal}$	x_{init}	Ts	σ_a	σ_z	P_D	Λ_c	Λ_a	P_d	Surveillance Area
1	100	$[-75, -75, 0, 0]$, $[75, 75, 0, 0]$	2	0.2	5	0.8	5	0	0	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$
2	100	None,	2	0.2	5	0.8	5	5e-2	5e-3	$x_{[min,max]} = y_{[min,max]} = [-150, 150]$

Table 6.28: Simulator parameter values for the mulit-target verification case. The parameters are explained in Chapter 5.

System Under Test Parameters The parameters of the PDAF algorithm was according to the tuning strategy set to correspond with the parameter values of the environment simulator, similar to the previous cases. In order to isolate some of the failure modes from the previous cases, the PDAF tracker is tested with and without the M/N initiation and termination extension in some of the simulation runs with the first simulator setup. This is done in the following manner:

- PDAF tracking without the M/N extension, meaning the tracker is initiated with the initial states of the targets and is not able to initiate/terminate tracks by itself.
- PDAF tracking with the M/N extension for track initiation, but not for track termination. This means that the tracker has to initiate the tracks by itself and is not able to terminate tracks by itself.
- PDAF tracking with the M/N extension for track termination, but not for track initiation. This means that the tracker is initiated in the same way as for the first run and may terminate the tracks by itself. However, it is not able to initiate tracks by itself.
- PDAF tracking with the M/N extension for track initiation and termination. This means that the tracker handles both initiation and termination by itself.

When running simulations with the second setup, the tracker must handle initiation and termination by itself. The parameter values when doing this is the same as for the last simulation run with the first setup. The parameter values of the tracker can be observed in Table 6.29. The first four simulation runs are the runs for the first simulator setup, and the fifth is the simulation run for the second simulator setup.

Simulation Run	σ_a	σ_z	P_D	P_G	Clutter Density	Ts	M	N	Initiation Threshold	Initiation	Termination
1	0.2	5	0.8	0.8	$5/(300^2)$	2	N.A.	N.A.	N.A.	False	False
2	0.2	5	0.8	0.8	$5/(300^2)$	2	4	5	1.0	True	False
3	0.2	5	0.8	0.8	$5/(300^2)$	2	4	5	1.0	False	True
4	0.2	5	0.8	0.8	$5/(300^2)$	2	4	5	1.0	True	True
5	0.2	5	0.8	0.8	$5/(300^2)$	2	4	5	1.0	True	True

Table 6.29: Target tracker parameter values for the multi-target verification case. The parameters are explained in Chapter 5.

Test Method Parameters The AST parameters was set to be equal to the parameters of the previous cases, except the slow termination case. This was done since the performance of the implementation seemed to work well in this case as well. The parameter values of the test method are presented in 6.30

N.O. Iterations	UCB Constant c	PW Constant c	PW Constant α	Likelihood Weight	Failure Weight
1000	0.1	2	0.25	1	1e4

Table 6.30: Simulator parameter values for the single-target verification case. The parameters are explained in Chapter 5.

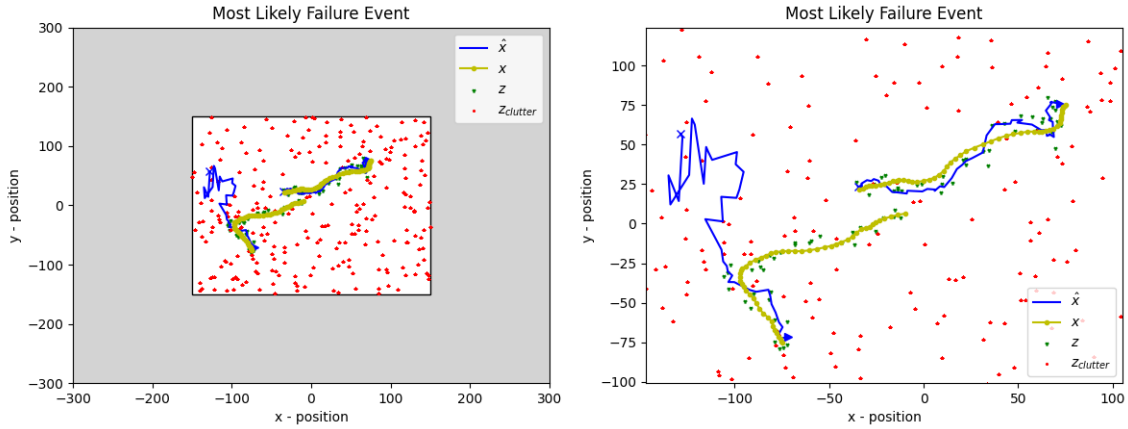
Critic Parameters In this case the Critic parameters is set based on the failures that is searched for, which is determined by comparing the GOSPA value with a threshold. The threshold was set differently for the different simulation runs, since it was a significant difference in the magnitude of the GOSPA between the cases. See Section 6.2.1 for further details about the parameters. The parameter values for the critic are presented in Table 6.31.

Simulation Run	Failure Threshold	Distance Metric
1	GOSPA = 65	d = max(65 - GOSPA, 0)
2	GOSPA = 65	d = max(65 - GOSPA, 0)
3	GOSPA = 50	d = max(50 - GOSPA, 0)
4	GOSPA = 55	d = max(55 - GOSPA, 0)
5	GOSPA = 90	d = max(90 - GOSPA, 0)

Table 6.31: Critic Configuration for the multi-target case.

Results and Discussion

PDAF Without Track initiation and Termination The first simulation run in this case was done with the first of the presented simulator setups and the tracker used was the PDAF tracker without the M/N extension. After 1000 MCTS iterations the following results was outputted from the algorithm:



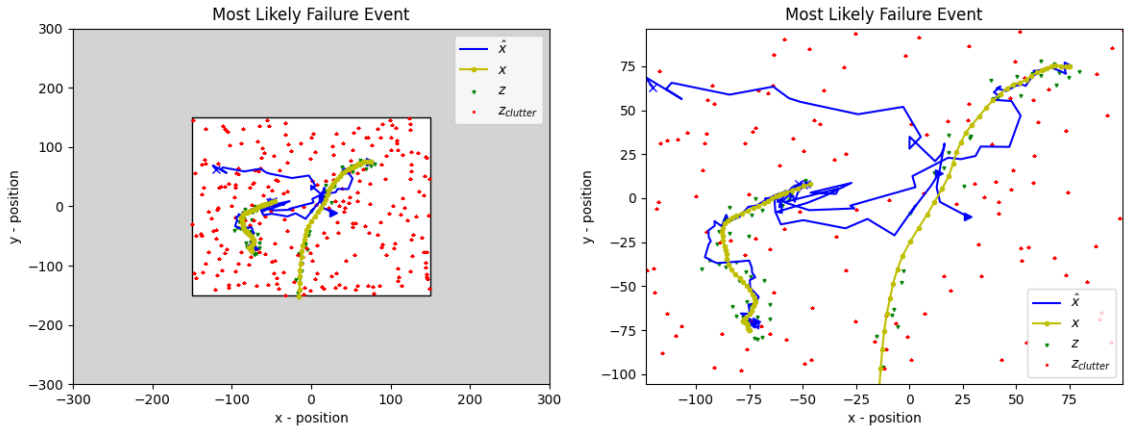
(a) (b)

Figure 6.27: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for Multi-target case 1. (b) shows a zoomed in version of the same plot. The area shaded gray is the outside of the surveillance area. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step.

It can be observed that two targets moves inside the surveillance area, and at some point are relatively close to each other. The tracker seems to follow one of the targets relatively close. For the other target the track drifts when the target makes a sudden turn. It can also be observed that in the area of the turn, several clutter measurements are generated through the simulation. This could be the reason of the drift. The drift is significantly high, which may categorize this scenario as a track loss scenario. This is probably an event which would have been avoided when using the M/N track existence extension, where the track may have been terminated during the drift and initiated again after the turn of the target. However, the objective of this case was to find similar failure events as in the previous cases, but with a more general test metric. Considering verification condition 1, which state that the AST implementation should find scenarios where the PDAF track drifts away from the target, the implementation using GOSPA as test metric is verified.

PDAF With Initiation and Without Termination The second simulation run in this case was done with the first of the presented simulator setups and the tracker used was the PDAF

tracker using the M/N extension for only initiation. The tracker was limited to not being able to terminate tracks. After 1000 MCTS iterations the following results was outputted from the algorithm:

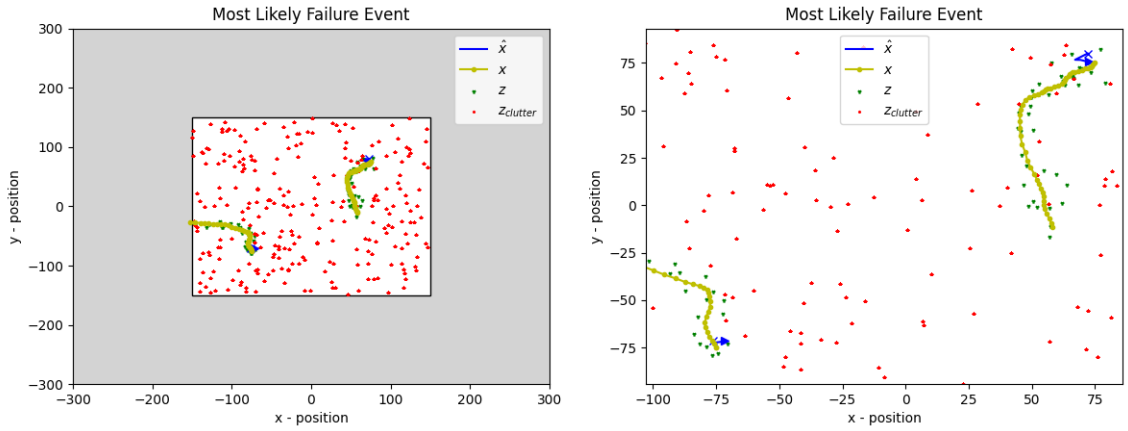


(a) (b)

Figure 6.28: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for Multi-target case 2. (b) shows a zoomed in version of the same plot. The area shaded gray is the outside of the surveillance area. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step.

It can be observed that two targets moves insides the surveillance area during the simulation. The tracker seems to following one of the targets quite good. However, for the second target the tracker struggles after a few time steps, and the estimate starts to drift towards the other target. The track is not terminated since the tracker is not allowed to do so. Then, a new track is initiated, which starts to drift towards the other track as well. Finally, a second track is initiated, which also drifts of from the target. Hence, one of the targets is not tracked for most of the simulation. It can be observed that the drift of the tracks occurs in an area where the target does not produce measurements and clutter measurements occurs, which may be the explanation for why the tracker struggles. Further, the objective of this simulation run was to find some of the same errors searched for in the previous case studies. In this case the failure is a track loss event which meets the first verification condition, stating that the AST implementation should be able to find scenarios where the PDAF track(s) drifts away from the target. Hence, the implementation is verified for this mulit-target scenario.

PDAF Without Initiation and With Termination The third simulation run in this case was done with the first of the presented simulator setups and the tracker used was the PDAF tracker using the M/N extension for only termination. The tracker was limited to not being able to initiate tracks. After 1000 MCTS iterations the following results was outputted from the algorithm:

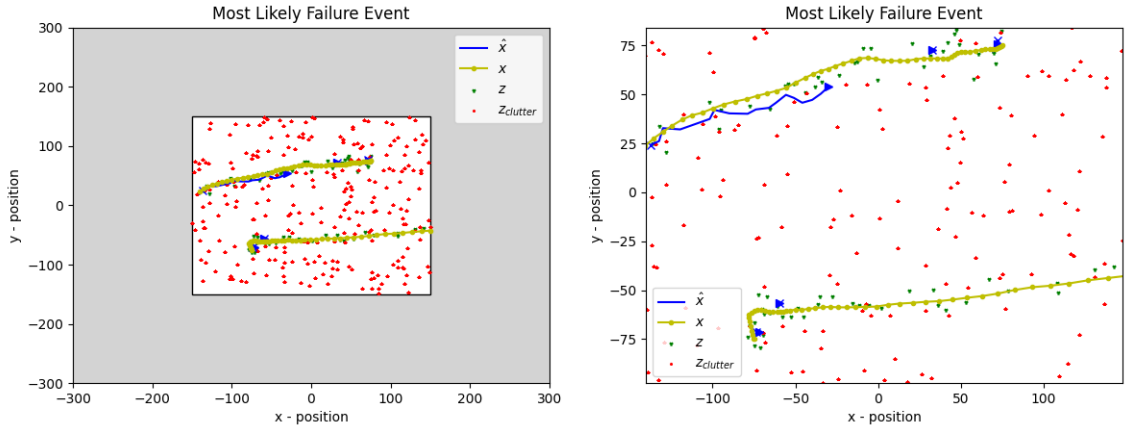


(a) (b)

Figure 6.29: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for Multi-target case 3. (b) shows a zoomed in version of the same plot. The area shaded gray is the outside of the surveillance area. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step.

It can be observed that two targets moves inside the surveillance area. For both of the targets, the tracker terminates the manually initiated track in the beginning of the simulation. Hence, the targets are not tracked after the first time steps. Considering the fifth condition for verification, which states that the AST implementation should be able to find scenarios where the tracker terminates tracks for existing target, the implementation is verified in terms of it's ability to find termination failure events related to termination of existing targets.

PDAF With initiation and termination The third simulation run in this case was done with the first of the presented simulator setups and the tracker used was the PDAF tracker using the M/N extension for both initiation and termination. After 1000 MCTS iterations the following results was outputted from the algorithm:

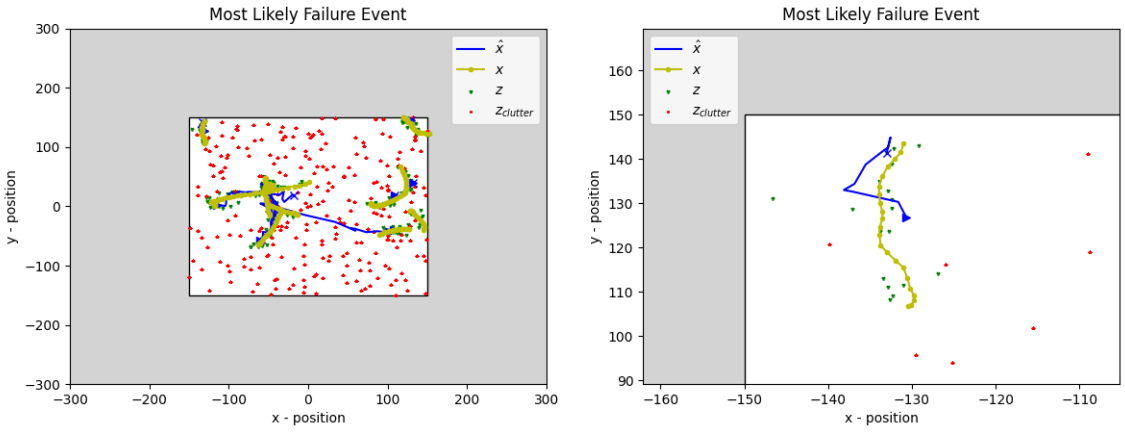


(a) (b)

Figure 6.30: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for Multi-target case 4. (b) shows a zoomed in version of the same plot. The area shaded gray is the outside of the surveillance area. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step.

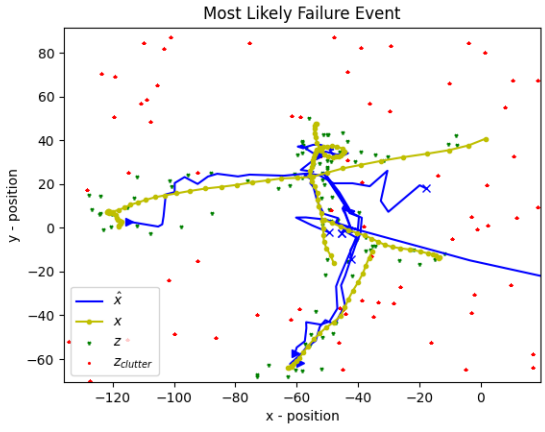
It can be observed that two targets moves inside the surveillance area. For one of the targets does the tracker initiate two tracks, that it terminates in just a few time steps. After the two tracks are terminated, the tracker does not initiate any new tracks. Hence, the target is for the most of the time not tracked. This is similar to the previous simulation run. For the second target the same type of failure occurs, but with the main difference that the tracker manages to track the target for a longer time. Hence, the AST implementation is verified for it's ability to find scenarios where the tracker terminates the track for a existing target. However, this failure would not have been significant if it weren't for the fact that the tracker struggles to initiate a new track for both of the target after terminating the initial track(s). The tracker manages to initiate a track for the second target, but terminates the track immediately. Hence, the scenario may be considered a failure scenario where the tracker fails to initiate tracks for existing targets. Considering these results the AST implementation is verified according to the verification requirement regarding finding failure events where the tracker fails to initiate tracks.

PDAF M/N in a Complete Multi-target Simulation The fifth simulation run in this case was done with the complete multi-target simulator and the tracker used was the PDAF tracker using the M/N extension for both initiation and termination. After 1000 MCTS iterations the following results was outputted from the algorithm:

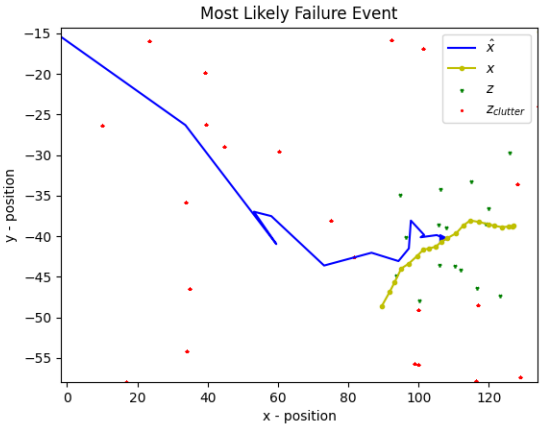


(a)

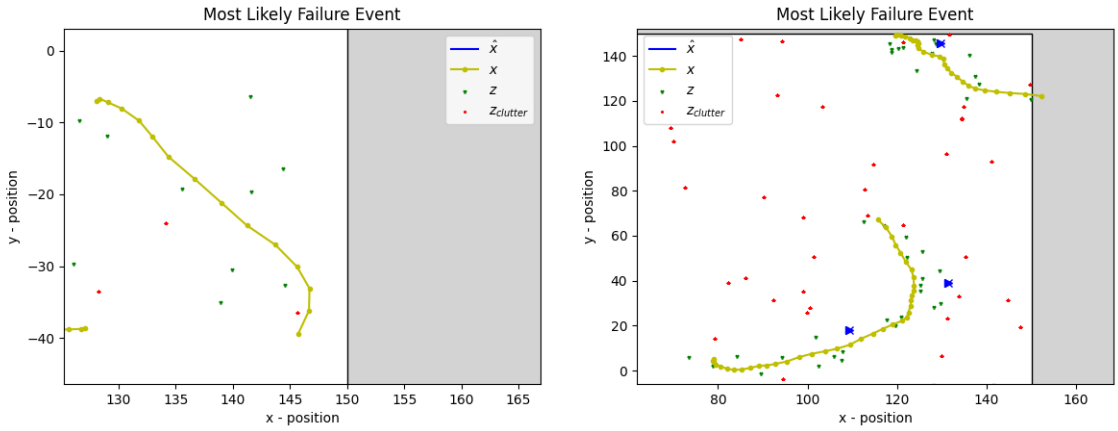
(b)



(c)



(d)



(e) (f)

Figure 6.31: (a) shows the track $\hat{x}_{0:t_{end}}$, target trajectory $x_{0:t_{end}}$ and the measurements $z_{0:t_{end}}$ relative to the surveillance area for Multi-target case 5. (b-f) shows zoomed in versions of the same plot. The area shaded gray is the outside of the surveillance area. Triangle denotes the initial state and the cross denotes the final state of the track. Clutter measurements are included for each simulation step.

It can be observed that through the simulation run are nine targets initiated and moves inside the surveillance area. The results are examined one sub-figure at the time:

- **Sub-figure (b):** It can be observed that the tracker manages to track the target through half of the target trajectory, before terminating the target. Hence, this is the case of termination of existing targets.
- **Sub-figure (c):** It can be observed that this plot is a quite complex multi-target scenario, but with close examination it can be observed that the following events occurs:
  - Track loss event due to target interaction.
  - Initiation of multiple tracks for a single target.
  - Failing to initiate track for existing target.
- **Sub-figure (d):** It can be observed the track is initiated in the middle of the target trajectory, and the track initiated starts to drift. Hence, the following failure events occurs:
  - Track loss event due to mis-detections and clutter measurements.
  - Slow initiation of track for existing target.
- **Sub-figure (e):** It can be observed that the tracker does not initiate any tracks following the target. Hence, a failure scenario with initiation failure has occurred.
- **Sub-figure (f):** It can be observed that for two targets, the tracker initiates and terminates tracks immediately, which in the previous simulation run was considered failing to initiate.

Considering all the failure events uncovered in this single simulation, the AST implementation is verified for finding some likely failure events of the PDAF target tracker with the M/N track existence extension when running a multi-target simulation. In addition, the did the method seem to find a failure event not searched for in earlier cases, which is initiation of multiple tracks for the same target. This would not have been considered a failure if one of the tracks was terminated quickly. However, instead both of the tracks continued to exist following the target through the simulation.

6.3.7 Validation Case: Quantitative Analysis of Adaptive Stress Testing of Situational Awareness

This objective of this case is to validate the AST implementation. This is done by comparing the success rate of AST with the success rate of a random search. The method is considered validated if it meets it's verification requirements for all of the cases with a significantly higher success rate than when performing a random search. In addition is the proposed open loop local control results compared to the results of using global control. This is done in order to determine if the use of the new control option improves the performance of the search. The results are presented in Table 6.32.

	Success Rate After 1000 Iterations							
	MCS	MCTS						
Case	Global	Global	Target dynamics	Measurement Generation	Detection	Clutter Generation	Target Arrival	Target Departure
State Estimation (RMSE > 9)	0.0%	49.1%	0.0%	39.8%	N.A.	N.A.	N.A.	N.A.
State Estimation (NEES Fraction Above CI > 0.4)	0.0%	72.2%	0.0%	31.3%	N.A.	N.A.	N.A.	N.A.
State Estimation (NEES Fraction Below CI > 0.4)	0.0%	35.3%	0.0%	28.9%	N.A.	N.A.	N.A.	N.A.
Single-target Tracking (Validation Gate Loss Condition)	1.2%	96.1%	91.5%	84.1%	0.8%	0.0%	N.A.	N.A.
Single-target Tracking (Euclidean Distance Loss Condition)	1.5%	97.3%	92.5%	88.6%	15.4%	0.0%	N.A.	N.A.
Track Initiation (Slow Initiation)	4.1%	97.0%	0.0%	98.2%	78.7%	0.1%	N.A.	N.A.
Track Initiation (Initiation Failure)	3.1%	93.2%	0.0%	0.0%	0.0%	0.0%	N.A.	0.0%
Track Termination (Slow Termination)	0.0%	72.6%	0.0%	0.0%	0.0%	0.0%	N.A.	94.7%
Track Termination (Failing to Terminate)	0.0%	74.4%	0.3%	44%	36.3%	18.1%	N.A.	0.8%
Track Termination (Terminating Existing Tracks)	0.0%	18.1%	0.4%	0.0%	0.0%	0.0%	N.A.	N.A.
Multi-target Simulation (Without Track Existence)	0.0%	42.1%	0.0%	0.0%	0.0%	0.0%	N.A.	N.A.
Multi-target Simulation (With Track Initiation)	0.0%	60.3%	7.8%	0.0%	0.0%	0.0%	N.A.	N.A.
Multi-target Simulation (With Track Termination)	22.6%	95.8%	91.5%	100%	99.5%	98.4%	N.A.	N.A.
Multi-target Simulation (With Track Initiation/Termination)	0.0%	12.5%	28%	47.8%	22.7%	44.6%	N.A.	N.A.
Complete Multi-target Simulation (With Track Initiation/Termination)	0.0%	15.7%	0.0%	0.0%	0.0%	0.0%	0.1%	0.0%

Table 6.32: Success rate for every case running both MCS and MCTS using open loop global control. The highest success rate for each case is marked with the color red. In addition the success rates when running MCTS using open loop local control with every stochastic process in the simulator are presented for each case. N.A. denotes not applicable, which is the case when a simulator component is not used.

It can be observed that the AST search using MCTS outperforms the MCS search with a significant margin in all of the case studies. With the failure thresholds set in these cases, the MCS performance is low. This may imply that the MCS search may be classified as incapable of finding failure events of such magnitude. The exception is in the third multi-target simulation case, where the MCS search manages to find failures with an success rate of 22%. However, in that case it may be argued that the failure threshold was set too low. This argument may be made since the AST search managed to find errors of such magnitude at almost all the MCTS iterations using global control and for all iterations when using local control of the measurement generation process. The MCTS search results have demonstrated that the MCTS search is very efficient in finding failure events where the magnitude of the error is large. Hence, the AST implementation is considered validated for the applications demonstrated in the verification case studies. According to the scientific approach described earlier in this chapter, the implementation may at this stage be passed forward to development phase 2. Then, the objective is as mentioned earlier to validate the implementation in an industrial high fidelity test environment. The implementation would then go through a qualitative analysis similar to the presented verification cases, before a quantitative analysis similar to this validation case is performed. In addition, another option is that the implementation at this stage is further extended with additional specifications, which would require further iterations through the development cycle presented in Figure 6.2.

The secondary objective of this case was to compare the results of using the open loop global control option with the open loop local control option. With this objective in mind it can be observed that the global option did have the highest success rate in most of the case studies. However, in three of the cases did the local control option outperform the global control option. This were the three following cases:

- The multi-target simulation case where the targets were initialized prior to the simulations, and the PDAF tracker with the M/N extension handling track initiation and termination.
- The multi-target simulation case where the targets were initialized prior to the simulations, and the PDAF tracker with the M/N extension handling only track termination.

- The track termination case where the objective was to find likely failures related to slow termination of tracks for departing targets.

For the two first cases, it the highest success rate occurred when controlling the process of sampling of measurements. For the last case the highest success rate was achieved by controlling the target departure process. Hence, the local control option did demonstrate to be better than the global control option in some cases. However, the scope for testing this option in this thesis, is not large enough to be conclusive about the effectiveness of the method. At this stage the method shows promise in some applications. Hence, further research is recommended in order to make a valid conclusion about the usefulness of the method.

6.4 Discussion

In these case studies, an AST system has been designed and implemented through a set of iterations through the development cycle presented in Figure 6.2. The development cycle is a part of the scientific approach for evaluation of AST as a test method, and provides a proposal for how the method should be tested and developed before it can be deployed for industrial testing of the situational awareness of an autonomous passenger ferry. In addition to the AST system being implemented, the test environment consisting of system under test and environment simulator has been designed and developed in parallel with the test method. The objective of the test environment development process has been to make the AST system subject of an increasingly more complex test environment. For each iteration the AST implementation has been tested and evaluated, resulting in the verification of the method's abilities to find common failure events in target tracking algorithms that may be used in situational awareness. Hence, the research has uncovered that the method functions as specified when applied to testing of situational awareness. In addition, the method has been evaluated by comparing it with the baseline value of a random search, which resulted in AST outperforming the random search with a significant margin. Hence, the method has demonstrated a high potential for finding likely failure events in situational awareness.

Even though the results presented appears to be promising, a key problem arises when applying the method in the way it has been in some of the case studies. The problem arises due to the fact that the test metrics used are specific for the events searched for in each individual case. This simplifies the search by making it highly directed. Hence, from these cases it can be concluded that the test method finds exactly what it searches for, but having the same problem with high exhaustiveness as for formal verification methods. This is due to the fact that an individual test metric has to be designed for each possible failure that exists, which may be infeasible. When dealing with an autonomy system it is important that the test method is able to search for failure events which the developer does not about in beforehand. Considering the problem of moving the residual risk of "the long tail of the probability distribution" further to the right, which was described in Chapter 2, it is important to have a method that can find errors that have not been thought of *a priori*. It is likely to believe that the residual risk is due to unpredictable scenarios. Hence, being able to find such scenarios would be highly beneficial in order to provide statistical evidence that the system is safe. In addition does the specific test metrics used in those cases not work for testing the situational awareness in complex multi-target scenarios. This is due to the fact that the test metrics requires knowledge about which exact target each track belongs to. Then, the targets and tracks would have to be pairwise assigned through an data association algorithm. Having these considerations in mind, more general test metrics and definitions of failure events should be sought for. This was the main motivation for applying the more generalized test metric GOSPA in the multi-target cases. The results demonstrated that using such a test metric may result in finding the same type of failure events as when using metrics specified for certain types of events. In addition, the method did find a failure event not defined or searched for in the previous cases. Hence, the method did show promising results for finding failure events which have not been explicitly defined *a priori*. This may be one of the most promising results presented in this thesis. However, the scope of the research in this master thesis is not large enough to make conclusions regarding the method's ability to find new undefined failure events in target tracking algorithms used in situational awareness. Hence, further research exploring the use of different generalized

test metrics should be performed in order to explore this topic further.

To deal with the high complexity of the failure definitions in target tracking, another approach may be to apply the method on a higher system level. Then, AST could be used to find likely failure events for e.g. the entire collision avoidance pipeline, including the object detection, situational awareness and motion planning components. AST would then rather create traffic scenarios that are inputted to the object detection component, which processes the scenarios and generates measurements for the situational awareness. Then, the target tracking algorithm of the situational awareness outputs a track to the motion planning algorithm, which decides what route the vessel should follow. The failure definition would in this case be significantly simpler, by defining collision with other objects as failure and using the distance from other objects as distance metric. This test regime would also be beneficial due to testing the interaction between multiple system components. This is a benefit due to safety being as earlier mentioned an emergent system property, meaning that failures may arise due to interaction between different system components. The object detection, situational awareness and motion planning components may then be observed closely during these simulations, making it possible to study how different failures occurs due to different inputs and outputs from the different components. Then, it could also be possible to detect previously unknown errors in the situational awareness without directly interacting with it.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

The main objective of this master thesis project has been to evaluate the use of AST in testing, verification and validation of Zeabuz's autonomous urban passenger ferries. The goal has been to provide a contribution to the following research question:

> *How is it possible to accumulate enough experience regarding safety and validation for autonomous passenger ferries such that it can be argued that the autonomy system is sufficiently safe?*

The research question was further examined through the following sub-questions:

1. What is the current state-of-the-art methods used in testing, verification and validation of autonomous systems, and how does AST fit into this context?
2. Can AST be used in order to perform state-of-the-art safety validation of the situational awareness in autonomous urban passenger ferries?

The contribution to first sub-question was done through reviewing literature related to AST and other methods of testing, verification and validation of autonomous systems. Generally, it has been concluded in the literature that traditional formal methods aren't sufficient in safety validation of autonomous systems. Hence, informal methods such as simulation-based testing is also necessary in order to provide safety proofs. However, previously applied simulation-based methods typically rely on manually constructing cases where the system under test will fail. This makes it a tremendous task to test an autonomous system. There exist methods for sampling of complete simulation cases in order to make the system struggle. However, these methods aren't as flexible as AST, as they require complete access to the simulator. On the other hand AST maximizes the likelihood of the failure, which no other method previously has done. The AST method has shown promising results in the aerospace and automotive industry. Hence, the method has in theory and through previous applications shown to be promising in terms of finding likely failure events in complex control systems, such as collision avoidance systems. In addition, AST has demonstrated to be an efficient method for finding unpredictable scenarios resulting in unforeseen failure events. This has been the key motivation for studying the method in for marine autonomous systems. Due to the large number of possible scenarios that an autonomous system may encounter, it is not possible to mitigate this risk completely. Hence, application of the AST method alone is not enough to provide validation that an autonomous system is safe. Hence, a combination of formal and informal testing, verification and validation methods are required to certify that the system is safe.

The contribution to the second sub-question has been done through a set of case studies. In the case studies it has been discussed how the method may be deployed for testing of the situational

awareness of an autonomous passenger ferry. This includes how the test method itself may be verified and validated through a iterative development cycle, where the implementation of the method was subject to a set of qualitative and quantitative analyses. Through the master project an AST system has been implemented through a set of iterations of the described iterative development cycle. Furthermore, the AST system has been connected to a situational awareness test environment developed in parallel with the AST system. The developed test environment consisted of a PDAF target tracking algorithm with the M/N extension for track existence, which acted as system under test. The test environment did also include a multi-target simulator, which acted as environment simulator. In order to control the test environment a set of appropriate control methods were proposed. One of the proposed methods was tested for the first time in this thesis, yielding inconclusive results. Furthermore, the AST system has been subject of the developed test environment and evaluated in a set of case studies. For each case the complexity of the simulation did increase. The results did demonstrate that the AST system was able to find common failure events in target tracking, which provided verification of the implementation. AST did find a case with a failure event not predicted by the author. This result demonstrated the method's potential for finding unforeseen failure events. In addition, the quantitative results did demonstrate that the AST implementation with MCTS was able to find failure events in a significantly more efficient manner than when running s random search. However, to say for sure that the AST method can perform as well as or better than the current state-of-the-art methods for safety validation of autonomous marine systems, further research is required.

7.2 Further Work

Considering the discussion in the case studies chapter, the following items are suggested for further work:

- Perform comprehensive simulations using the proposed open loop local control method, in order to verify and validate the control method.
- Perform research exploring generalized test metrics for defining failure events in target tracking algorithms used in situational awareness, and perform comprehensive simulations with the explored test metrics in order to determine the method's ability to find unforeseen failure events in situational awareness.
- Perform further validation of AST as a test method through qualitative and quantitative analyses of the implementation in a high fidelity test environment. This could be e.g. the Zeabuz milliAmpere 2 (mA2) digital twin.
- Develop algorithmic extensions to AST in order to improve AST's performance. This could be using a search algorithm extended for selecting multiple actions at each time step, improving the performance of the open loop local control method.
- Test the method in application on a higher system level, e.g. the complete collision avoidance pipeline in the Zeabuz system.
- Test the use of Differential Adaptive Stress Testing for evaluation of the performance of a system under development relative to the performance of an established system.

Bibliography

[1] T. R. Torben, J. A. Glomsrud, T. A. Pedersen, I. B. Utne, and A. J. Sørensen, "Automatic simulation-based testing of autonomous ships using gaussian processes and temporal logic," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 0, no. 0, p. 1748006X211069277, 0.

[2] R. Lee, O. J. Mengshoel, A. Saksena, R. W. Gardner, D. Genin, J. Silbermann, M. Owen, and M. J. Kochenderfer, "Adaptive stress testing: Finding likely failure events with reinforcement learning," *Journal of Artificial Intelligence Research*, 2020.

[3] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, "Adaptive stress testing of airborne collision avoidance systems," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pp. 6C2–1–6C2–13, 2015.

[4] J. B. Sørensen, "Adaptive stress testing of situational awareness for an autonomous urban passenger ferry.," 2021.

[5] "Autonomy vocabulary defintion." `https://www.vocabulary.com/dictionary/autonomy`. Accessed: 12-11-2021.

[6] A. J. Sørensen, *Marine Cybernetics, Towards Autonomous Marine Operations and Systems*. Department of Marine Technology, Norwegian University of Science and Technology, 2018.

[7] L. R. A. A. P. Grewal, Mohinder S.; Weill, "Global positioning systems, inertial navigation, and integration," *Hoboken, New Jersey, USA: Wiley-Interscience, John Wiley Sons, Inc*, 2007.

[8] "Ieee standard adoption of iso/iec 15026-3 – systems and software engineering – systems and software assurance – part 3: System integrity levels," *IEEE Std 15026-3-2013*, pp. 1–51, 2013.

[9] "Ieee standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, 1990.

[10] J. Kapinski, J. V. DeshmuKh, X. Jin, hisahiro ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE CONTROL SYSTEMS MAGAZINE*, 2016.

[11] C. Kern and M. R. Greenstreet, "Formal verification in hardware design: A survey," in *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, p. 123–193, TODAES, 1999.

[12] A. Pnueli, "The temporal logic of programs," *Foundations of Computer Science*, p. 46–57, 1977.

[13] R. W. Gardner, D. Genin, R. McDowell, C. Rouff, A. Saksena, and A. Schmidt, "Probabilistic model checking of the next-generation airborne collision avoidance system," *Digital Avionics Systems Conference (DASC)*, 2016.

[14] J.-P. Katoen, "The probabilistic model checking landscape," *ACM/IEEE Symposium on Logic in Computer Science*, p. 31–45, 2016.

[15] J. H. Gallier, "Logic for computer science: Foundations of automatic theorem proving," *Courier Dover Publications*, 2015.

[16] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, "A formally verified hybrid system for the next-generation airborne collision avoidance system," *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2015.

[17] H. Krasowski and M. Althoff, "Temporal logic formalization of marine traffic rules," in *2021 IEEE Intelligent Vehicles Symposium (IV)*, pp. 186–192, 2021.

[18] T. Torben, O. Smogeli, I. Utne, and A. Sørensen, "On formal methods for design and verification of maritime autonomous surface ships," 04 2022.

[19] O. Smogeli and T. Augustson, "Third party hil testing of safety critical control system software on ships and rigs," vol. 1, 07 2012.

[20] T. Johansen, A. Sørensen, O. Nordahl, O. Mo, and T. Fossen, "Experiences from hardware-in-the-loop (hil) testing of dynamic positioning and power management systems," 09 2007.

[21] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, p. 254–257, 2011.

[22] D. T. D. A. K. J. J. X. . D. J. V. Dreossi, T., "Efficient guiding strategies for testing of temporal properties of hybrid systems," *NASA Formal Methods Symposium*, p. 127–142, 2015.

[23] R. Lee, O. Mengshoel, A. Saksena, R. Gardner, D. Genin, J. Brush, and M. J. Kochenderfer, "Differential adaptive stress testing of airborne collision avoidance systems," in *2018 AIAA Modeling and Simulation Technologies Conference*, p. 1923, 2018.

[24] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, "Adaptive stress testing for autonomous vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1–7, 2018.

[25] M. Koren and M. J. Kochenderfer, "Efficient autonomy validation in simulation with adaptive stress testing," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 4178–4183, 2019.

[26] K. D. Julian, R. Lee, and M. J. Kochenderfer, "Validation of image-based neural network controllers through adaptive stress testing," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–7, 2020.

[27] R. Lee, J. Puig-Navarro, A. K. Agogino, D. Giannakoupoulou, O. J. Mengshoel, M. J. Kochenderfer, and B. D. Allen, "Adaptive stress testing of trajectory planning systems," in *AIAA Scitech 2019 Forum*, p. 1454, 2019.

[28] R. J. Moss, R. Lee, N. Visser, J. Hochwarth, J. G. Lopez, and M. J. Kochenderfer, "Adaptive stress testing of trajectory predictions in flight management systems," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pp. 1–10, IEEE, 2020.

[29] K. El-Awady, "Adaptive stress testing for adversarial learning in a financial environment," *arXiv preprint arXiv:2107.03577*, 2021.

[30] S. J. Russell and P. Norvig, *Artificial Intelligence : a Modern Approach, third edition*. Pearson Education Limited, 2016.

[31] B. Wang, "Monte carlo tree search: An introduction."

[32] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," vol. 2006, pp. 282–293, 09 2006.

[33] E. Brekke, *Fundamentals of Sensor Fusion: Target tracking, navigation and SLAM*. 2021.

[34] "Adaptive stress testing python toolbox documentation." `https://ast-toolbox.readthedocs.io/en/latest/tutorial.html#introduction`. Accessed: 14-11-2021.

[35] T. Dingsøyr, T. Dybå, and N. Moe, "Agile software development: An introduction and overview," *Agile Software Development, by Dingsøyr, Torgeir; Dybå, Tore; Moe, Nils Brede, ISBN 978-3-642-12574-4. Springer-Verlag Berlin Heidelberg, 2010, p. 1*, vol. -1, p. 1, 04 2010.

[36] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[37] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[38] R. J. Moss, "POMDPStressTesting.jl: Adaptive stress testing for black-box systems," *Journal of Open Source Software*, vol. 6, no. 60, p. 2749, 2021.

[39] R. Lipkis and A. Agogino, "Adastress.jl."

[40] M. Coren, A. Corso, R. Moss, and X. Ma, "Adaptivestresstestingtoolbox.py."

[41] P. Sinclair and A. Prengere, "Mcts implementation."

[42] H. Baier and P. I. Cowling, "Evolutionary mcts for multi-action adversarial games," in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, 2018.

[43] H. Baier and P. I. Cowling, "Evolutionary mcts with flexible search horizon," in *AIIDE*, 2018.

[44] A. S. Rahmathullah, Á. F. García-Fernández, and L. Svensson, "Generalized optimal sub-pattern assignment metric," *CoRR*, vol. abs/1601.05585, 2016.

Johan Bakken Sørensen

Adaptive Stress Testing of Situational Awareness for an Autonomous Passenger Ferry

NTNU
Norwegian University of
Science and Technology