On Formal Methods for Design and Verification of Maritime Autonomous Surface Ships

Tobias R. Torben[1], Øyvind Smogeli[1,2], Ingrid B. Utne[1] and Asgeir J. Sørensen[1]

ABSTRACT

Maritime Autonomous Surface Ships (MASS) are approaching a reality, introducing a new level of complexity and criticality to maritime control systems. In this paper we investigate how Formal Methods (FMs) can be used to design and verify maritime control systems for safe and effective MASS. FMs are a family of mathematically based methods for specification and verification. We begin by giving a high-level introduction to FMs. We discuss the current practice for certification of maritime control systems and needs going towards autonomy. We give three specific examples on how FMs can be applied to meet these needs: Formal specification of COLREG, contract-based design and automation of simulation-based testing. Finally, some limitations of FMs are discussed. We conclude that FMs appear as a promising candidate to meet some of the needs going towards autonomy, and encourage further research into FMs for MASS.

KEY WORDS

Maritime Autonomous Surface Ships, Formal Methods, Verification, Specification, Assurance

INTRODUCTION

Maritime Autonomous Surface Ships (MASS) are approaching a reality, with numerous ongoing projects ranging from small research prototypes to full-scale industrial vessels. Although several degrees of autonomy exist, MASS are typically distinguished by being able to operate independently of a human operator in a non-trivial operation, requiring situational awareness and planning abilities. These characteristics have created a need for new design methodologies among MASS developers, as well as a need for new methods and processes for safety assurance among regulators (IMO 2021, NMD 2020) and classification societies (DNV 2018).

Formal Methods (FMs) are a family of mathematically based methods for specification and verification originating from theoretical computer science (Woodcock et al. 2009). FMs offer a high level of assurance and have therefore been used actively in the development and verification of critical systems in other industries, such as aerospace and railway, for several decades. With the advent of autonomous systems, FMs have been considered as a promising candidate to address some of the assurance challenges they introduce. This has resulted in active research on FMs applied to autonomous cars and aerial vehicles over the past decade (Luckcuck et al. 2019).

The maritime industry has not yet seen a significant adoption of FMs. This seems to be changing, however, as a few articles have been published during the last year. Shokri-Manninen et al. (2020) have created a formal automata-based model of single-vessel encounters and synthesized a correct-by-construction navigation strategy. Park and Kim (2020) have synthesized a correct-by-construction controller for automatic docking of marine vessels based on reachability analysis. Foster et al. (2020) present a controller for autonomous marine vessels in form of a hybrid dynamical system and use an automated theorem prover to verify some safety invariants.

This article aims to bring FMs to the attention of the maritime community by first giving a high-level introduction. Next, we review the current practice for design and verification of maritime control systems and discuss some needs going towards autonomy. We then motivate and demonstrate the use of FMs in three specific use cases to meet these needs. Finally, we discuss some of the limitations of FMs.

[1] Centre for Autonomous Marine Operations and Systems (AMOS), Department of Marine Technology, Norwegian University of Science and Technology
[2] Zeabuz AS

AN INTRODUCTION TO FORMAL METHODS

FMs are motivated by the common engineering expectation that mathematical analysis will improve the performance and reliability of a system. Early development of FMs dates to the 1960's, where they were first applied to the design of logic circuits and primitive computer programs.

Figure 1 shows the main steps and activities in a formal development process. We will use this figure to introduce the different FM tools and show where they fit in the development process. We also emphasize that taking on a full-fledged formal development process with all the steps in Figure 1 may be time-consuming and may not be appropriate in all cases. However, tools and methods from individual steps, such as only creating a formal specification, can still give great value to the development process. This is often referred to as *lightweight formal methods.*

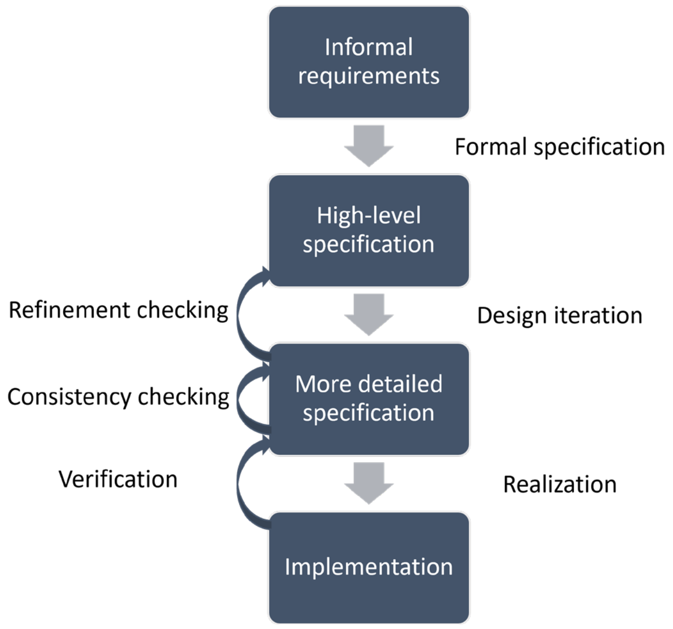


Figure 1: Steps and activities in a formal development process.

Formal specification

The first step in any development process is the requirements capture. This is often based on a text-based *concepts of operations* (CONOPS) document and produces an informal, text-based specification of high-level requirements. In a formal development process, the next step is to take the informal requirements and formulate them in a *formal specification language.*

Formal specification languages come in many forms, depending on what they are designed to describe. For instance, some are more suited for describing behaviour whereas others are better at describing structure. What they all have in common is that they have clearly defined syntax and semantics, that is, there are strict rules on valid statements, and it is clearly and unambiguously defined how the statements are to be interpreted. This not only means that they can be subject to mathematical analysis, but also that they are machine readable and, therefore, can be subject to automated reasoning by a computer.

A widely used class of specification languages are *temporal logics*, which are appropriate when specifying temporal behaviours, such as the ordering of events, timing, and avoidance of deadlocks. A temporal logic specification is written as a formula which combines logical operators, such as AND, OR, NOT and IMPLIES with temporal operators, such as ALWAYS, EVENTUALLY, NEXT and UNTIL. The original and simplest form of temporal logic is Linear Temporal Logic (LTL) (Pnueli 1977), which operates on Boolean signals in discrete time. An example of an LTL formula specifying correct behaviour of a traffic light is given below. In natural language, this formula specifies that always, if there is a red light, then it should eventually turn green after first being yellow.

*ALWAYS (**red** IMPLIES (EVENTUALLY **green** AND (NOT **green** UNTIL **yellow**)))* [1]

Signal Temporal Logic (STL) (Maler and Nickovic 2004) is another type of temporal logic which can be used to specify real-valued signals over continuous time, including timing constraints. It is therefore a popular choice for embedded and cyber-physical systems. STL also include *robustness semantics*, which instead of giving a true/false evaluation of whether a signal satisfies a temporal logic formula, gives a quantitative number on how robustly the formula is satisfied. Hence, STL offers both a syntax to specify behaviours and a metric for evaluating the compliance to the behaviour, which has proven to be a powerful combination.

Another class of specification languages are *set-based,* building on mathematical set-theory. A modern and prominent example of this is Event-B (Abrial 2011), which combined with the free software tool Rodin supports all the steps in Figure 1. Event-B is appropriate for specifying both structure and discrete-event behaviours. The predecessor of Event-B, called B Method, was used to formally specify, verify, and automatically generate 86 000 lines of code for the driverless metro in Paris, resulting in a system for which no bugs have been found (Lecomte et al. 1991).

A final important class of FMs is called *theorem provers.* Here, specifications are written as mathematical statements and there exists tools to generate mathematical proofs of the statements. Theorem provers can be divided into automatic provers, such as Z3 (de Moura and Bjørner 2008), and interactive provers, such as Isabelle (Paulson 1994), where the user interacts with a computer tool to build a proof.

Design iterations

Having created a high-level specification, the next step in a formal development process is to refine the specification in a series of design iterations. Each design iteration adds more detail to the specification and brings it closer to something which can be realized in hardware and software. An important activity for each design iteration is the *refinement checking*, which aims to verify that a refined specification still contains all the properties of the higher-level specification. In other words, if a system meets the refined specification, it should also meet the higher-level specification. Many FM tools include automated refinement checking.

Another important step in the design iterations is *consistency checking*. This involves checking a specification for contradictions. The most obvious form of a contradictory specification is the statement "(**a**) AND (NOT **a**)". In a complex specification however, contradictions can be subtle and hard to detect. A classic example is interface incompatibility. This can lead to expensive redesigns at a later stage in the development process. Many FM tools also support automated consistency checking.

Realization

The next step in a formal development process is to create an implementation in hardware or software based on the formal specification. In its most lightweight form, this can simply involve manual programming from the specification. Although this does not use the full potential of a formal development process, many find it easier to produce correct code when writing from a clear and unambiguous specification where fundamental design flaws have been removed at the design stages.

For some specification languages, there also exist tools which can automatically generate a *correct-by-construction* implementation from the specification. Examples of this include generation of computer code in various programming languages and synthesis of controllers which control a system to meet the specification.

Formal verification

The final step in the formal development process is the *verification.* Here, we define verification as checking that an implementation satisfies the lowest level specification. The most prominent formal verification technique is *model checking.* The input data to a model checking tool is the system to verify and a specification to verify against. The specification is usually in the form of a temporal logic formula. The model checker achieves an exhaustive verification by checking all possible executions of the system against the properties defined in the formal specification. The result is either a verification if no violating behaviour is found, or a falsification with a corresponding counter example if a violating behaviour is found. Some prominent model checkers are SPIN (Holzmann 1997), NuSMV (Cimatti et al. 2002) and UPPAAL (Larsen et al. 1997).

Automated theorem provers, as introduced earlier, can also be used to obtain a formal verification by for instance proving the correctness of an algorithm. Finally, *reachability analysis* (Asarin et al. 2007) is worth mentioning. This technique identifies the set of reachable states from any state in a system. By defining an unsafe set, this can be used to formally verify safety by showing that the unsafe set is not reachable from any state.

CURRENT PRACTICE IN THE MARITIME INDUSTRY AND NEEDS GOING TOWARDS AUTONOMY

Introducing novel and disruptive autonomous technology will alter the way ships are designed and operated. This has the potential to reduce some risks related to manned operations. At the same time, new risks emerge that need to be identified and mitigated to ensure safety. In this section the current practice for verification of maritime control systems is presented and some challenges and needs going towards autonomy are discussed.

To illustrate the current practice for verification of maritime control systems, we will use the certification process for dynamic positioning (DP) systems as an example. DP systems are complex automatic control systems involving both sensing, actuation, and computer systems (Sørensen 2011). The certification process for DP systems has been developed based on decades of experience. We therefore believe that DP systems represent a relevant example of state-of-the-art practice for verification of complex maritime control systems.

The international guidelines for DP systems are published by the International Maritime Organization (IMO) in IMO 1580 (IMO 2017). These guidelines provide the requirements for achieving a certification in form of a Dynamic Positioning Verification Acceptance Document (DPVAD). The guidelines specify both functional and operational documents in addition to requirements for verification activities. They are mostly concerned with the computer systems, power system, thrusters, and sensor systems. Classification societies publish class notations based on IMO 1580, which add more detailed and possibly additional requirements that strengthen safety and performance.

The overarching safety philosophy for IMO 1580 is heavily inspired by *functional safety.* This means that safety is promoted by identifying a set of *failure modes* and providing safety functions to ensure that no single failure can lead to a loss of position. The safety function is very often implemented by requiring redundancy, such as having two or three separate DP computers, three independent position reference sensors and two segregated power systems. The guidelines require that a *Failure Mode and Effect Analysis* (FMEA) should be carried out, which involves a systematic analysis of the DP system listing all failure modes and demonstrating the safe response. The survey and testing activities required by the guidelines involve a complete, physical survey of the DP system during commissioning and address primarily hardware components. This includes FMEA proving trials, where failure modes are simulated, and proper response by the DP system is verified. Furthermore, periodical testing at least every five years is required in addition to annual surveys. Proper testing and verification of software are not covered by FMEA testing alone. This may be characterized as a weakness with the existing mandatory regulations for software intensive systems such as DP (Johansen et al. 2007, Smogeli and Skogdalen 2011). Attempts at closing this gap includes both product assurance through e.g. Hardware-in-the-Loop testing (DNV 2013) and process assurance through SW development inspired standards such as ISDS (DNV 2017), but neither of these approaches have become mainstream in the assurance of maritime control systems.

Looking towards MASS, the need for new methodology for handling complex, software intensive systems is even more pressing. The challenges of the current, prescriptive classification regime are also acknowledged in the DNV class guideline for autonomous and remotely-operated ships (DNV 2018). A functional safety philosophy alone is not adequate when the level of autonomy increases. A key attribute of autonomous systems is that they interact with dynamic, unstructured, and uncertain environments. A consequence of this is that an autonomous vessel could exhibit unsafe behaviour without any equipment failure, for instance due to an unexpected environmental interaction or insufficient situational awareness. This challenge can be addressed by a complementary safety philosophy called *Safety Of The Intended Functionality* (SOTIF). Looking to the automotive industry, autonomous and advanced automation systems are now certified both for functional safety through ISO26262 (ISO 2011) and SOTIF through ISO21448 (ISO 2019), and it is likely that a similar development must take place for MASS.

Another consequence of interaction with dynamic, unstructured and uncertain environments is that the number of possible scenarios an autonomous system can encounter becomes enormous. Relying on testing alone, which even with massive scaling through parallelized simulators, only can analyse a limited number of scenarios, will therefore not be sufficient to ensure safety. Instead, there is a need for methodology to design systems with mathematically proven safety guarantees which reduce the span of possible scenarios.

Another gap in the current practice is the lack of focus on software failures. Autonomous vessels will be inherently software intensive, and their software will likely have significantly higher complexity than current systems. The current practice has a high focus on hardware failures, which have a different nature than software failures. The IEC 60050 standard defines a software failure as a manifestation of a dormant software fault, and a software fault is defined as a state of a software item that prevents it from performing as required. Software faults can for instance come in form of specification faults, design faults, programming faults, compiler-inserted faults, or faults introduced during software maintenance (IECTC 2002). A software component may work perfectly for several years before a particular combination of input and internal state causes a software failure. Most of these types of failures are not possible to detect and remove by surveys and onboard FMEA testing only. Simulation-based testing through a Hardware-in-the-loop (HiL) setup has been practiced for DP systems for almost two decades as a voluntary additional service. In HiL testing, the DP software, running on the target hardware, is connected to a simulation model of the vessel (digital twin) and its environment through a HiL interface. This enables more targeted, detailed and extensive testing of the control software compared to onboard testing. It is possible to conduct a complete virtual sea trial before the targeted control system is installed on the ship. The experience from HiL testing of DP software has uncovered a large number of critical software bugs both in the core software and in configuration of particular DP vessels (Smogeli 2015). Large scale simulation-based testing is expected to be a key methodology for verification of MASS (Pedersen et al. 2020). This requires formalization and automation in the scenario generation and selection, simulation evaluation and coverage assessment (Torben et al. 2022). There is also a need for methodology to produce software which is correct-by-construction, which furthermore will result in reducing the test space.

Experience from DP vessels has also shown that system integration is difficult and error prone. Integration often receives little attention in the design stage, where changes are easy and cheap to make, and is instead delayed until commissioning and sea trials. This was the motivation for creating the Open Simulator Platform (OSP) which provides means for sharing simulation-models under a standardized interface while protecting vendor intellectual property (IP) (Smogeli et al. 2020). As the complexity and criticality increase further with MASS, there is clear a need for a structured and formalized integration processes.

Recent advances in Machine Learning (ML) are a key enabler for autonomy. The use of ML methods, which learn from data instead of being programmed from a specification, has enabled engineers to solve problems which are hard to specify completely. Computer vision is a classic example of this. However, ML based software also introduces major challenges for safety verification, as they have a strong black-box characteristic and lack of explainability. In addition, the input may be high-dimensional, such as all the pixels in an RGB image. This renders traditional input-output testing impractical to get sufficient test coverage. Clearly, there is a need for new methods for verification of ML-based software (DNV 2020).

Finally, we will mention the challenge due to lack of experience with autonomous vessels. Going autonomous represents a step change both in terms of the technology and the operations, which means that there is very limited experience, guidelines and best practices to build on. The automotive industry has approached this by releasing millions of cars driving autonomously under human supervision and continuously collecting data and building experience. As the number of MASS will be far less than the number of autonomous cars, combined with the tendency to use customized components and doing one-of-a-kind builds, we cannot expect to gain large-scale experience in this way. This necessitates increased efforts in the design and verification phases.

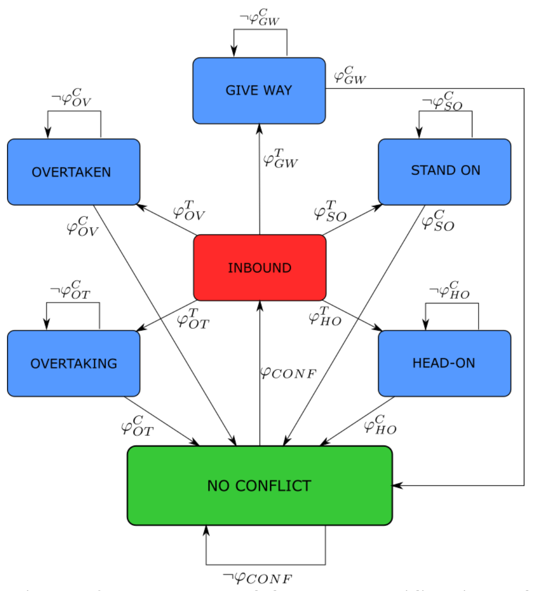
APPLICATIONS FOR FORMAL METHODS TO MASS DESIGN AND VERIFICATION

In this section we will demonstrate how FMs can be used to address some of the needs discussed in the previous section by presenting three specific use cases: Temporal logic specification of COLREG, contract-based design and automated simulation-based testing.

Temporal Logic Specification of COLREG

Safe interaction with other vessels is a challenging and critical aspect of MASS design and verification. Currently, this is regulated by the Convention on the International Regulations for Preventing Collisions at Sea – COLREG (IMO 1972), which specify a set of maritime traffic rules for collision avoidance. COLREG were designed for manned vessels and employ terms such as “due regard”, “appropriate distance” and “ample time”, and are therefore stated in a format that is not fit for computer parsing. To design collision avoidance algorithms and to enable automatic evaluation of COLREG compliance during verification, there is a need for machine readable COLREG. FMs stands out as a good candidate for formal specification of COLREG. An approach for this has been proposed in Torben et al. (2022) using STL, as shown in Figure 2. A similar approach, using the closely related Metric Temporal Logic (MTL) is proposed in Krasowski and Althoff (2021).

It may also be appropriate to develop a new collision avoidance protocol partially replacing COLREG, which is designed to be machine readable from the beginning. Using a formal language to specify the protocol also enables formal verification of safety properties. A similar development can be observed in the aviation industry, where the traditional TCAS protocol has been superseded by the ACAS X protocol, which has been formally specified and verified (Jeannin et al. 2017).



$$
\begin{aligned}
&\varphi_{colreg} = \Box(HO \rightarrow \varphi_{HO} \wedge GW \rightarrow \varphi_{GW} \wedge OT \rightarrow \varphi_{OT})\\
&\varphi_{HO} = t_{CPA} \leq t_{CPA,turn} \rightarrow \beta_r \in [-170^\circ, -10^\circ]\\
&\varphi_{GW} = t_{CPA} \leq t_{CPA,turn} \rightarrow d_{CPA} \geq d_{CPA,min}\\
&\varphi_{OT} = t_{CPA} \leq t_{CPA,turn} \rightarrow d_{CPA} \geq d_{CPA,min}\\
&\varphi_{OT}^{C} = \varphi_{OV}^{C} = \varphi_{GW}^{C} = \varphi_{SO}^{C} = \varphi_{HO}^{C} = t_{CPA} \leq 0\\
&\varphi_{NC}^{C} = d_{CPA} \leq d_{CPA,min} \wedge t_{CPA} \leq t_{CPA,min}
\end{aligned}
$$

Figure 2: Example of formal specification of COLREG using STL. The left figure shows a finite-state machine for selecting the COLREG situation (HO = Head-on, GW = Give way, SO = Stand-on, OT = Overtaking, OV = Overtaken). Each situation has a trigger condition based on the sectors for the heading and bearing of an incoming vessel. To the right, a set of STL formulas are given for the COLREG situations which require explicit action by own ship. The reader is referred to Torben et al. (2022) for more details.

Contract-based design

Modularity in design is a key success factor for managing complexity. This is motivated by the "divide and conquer" problem solving strategy, where a complex problem is broken up into a set of simpler subproblems, and the solutions of the subproblems are combined to solve the complex problem. Moreover, having a good framework for system integration of modular entities is a great benefit when interfacing with third-party systems and when integrating Commercial Off-The-Shelf (COTS) components. A challenge is to ensure that unsafe interactions across modules are sufficiently accounted for and mitigated.

Contract-based design is a formal approach for building modular systems (Sangiovanni-Vincetelli et al. 2012). Each modular unit is called a *component*, having a clearly defined interface in terms of inputs and outputs. Each component has a formal *contract* in assume-guarantee form. Verifying a single component involves producing evidence that the component satisfies a set of guarantees on its outputs, given that a set of assumptions on its inputs are satisfied. Contracts can be formulated in a formal specification language. When integrating a set of components, this reduces to checking the compatibility of the contracts. This activity is often termed *compositional reasoning.* The contract-based approach also supports design iterations, where a high-level abstract design is refined into more concrete and detailed designs, as illustrated in Figure 1.

Figure 3 shows an example of an autonomy system high-level component structure. The autonomy system component (grey) is intended to be connected to a DP system (not illustrated) which feeds the autonomy system with navigation data and controls the MASS to follow the motion reference given by the autonomy system. In addition, the autonomy system has environmental interactions in form of traffic, obstacles and weather. Looking inside the grey box, we can see that the abstract autonomy system component description is refined by three components, each with their own contract. To verify that this is a correct implementation, refinement checking is necessary. An informal description of refinement checking in assume-guarantee contracts is "assume no more, guarantee no less". This means that the combination of assumptions of the three subcomponents are less restrictive than the assumptions of the abstract autonomy system component, and that the combination of guarantees of the subcomponents are strong enough to enforce the guarantees of the abstract component. When integrating the three components, the compositional reasoning involves proving that the contracts of connected components are compatible. An example of a contract for the motion planning component, written in natural language could be:

Assuming that the tracks of other vessels are at accurate to 10m and that the navigation data are accurate to 1m, the motion reference is guaranteed to always have a distance of at least 50m to other vessels.

The compositional reasoning would then require verifying that the contract of the situational awareness component actually guarantees an accuracy of less than 10m on its tracks, and that the contract for the DP system guarantees navigation data with accuracy less than 1m. The refinement checking would involve verifying that the distance to other vessels guaranteed by the motion planning component is less than the distance guaranteed by the abstract autonomy system component. This shows how refinement with refinement checking can project high-level requirements onto lower-level components in a coherent way. If contracts are written in a formal specification language, both the refinement checking, and the compositional reasoning can be performed by an automated theorem prover.

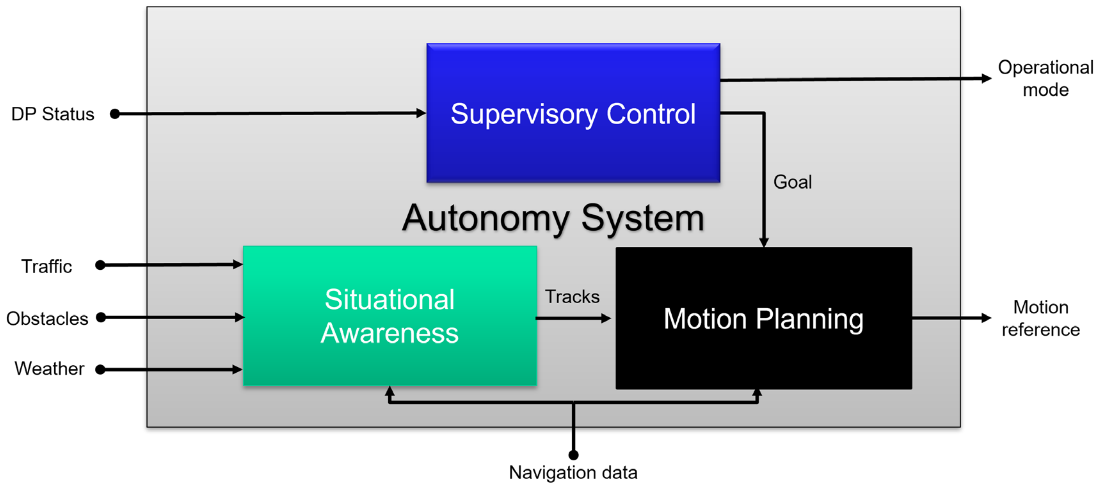


Figure 3: An example of a component structure for a high-level description of an autonomy system.

Automated simulation-based testing

In the final example we will show an example of how formal specification can be used to automate simulation-based testing. Evaluating the results of a simulation is often a non-trivial exercise which typically has been performed manually. If the testing scales to thousands of simulations, it is evident that manual evaluation is not practical, and for millions of simulations it is not practically possible. Torben et al. (2022) show how specifying requirements to test against in STL enables automatic evaluation of simulations using the STL robustness metric. The STL robustness metric gives a quantitative number on how robustly a simulation satisfies a requirement. If the robustness is greater than zero, the simulation satisfies the requirement, and if the robustness is less than zero, the simulation violates the requirement. The magnitude of the robustness is a measure of how much a signal can change without violating the requirement. Efficient algorithms exist for STL Monitors, which evaluate a signal against an STL formula (Fainekos et al. 2009).

The automatic testing methodology of Torben et al. (2022), shown in Figure 4, combines an STL monitor with a Gaussian Process (GP) model, which is used for smart scenario selection and coverage assessment. The user inputs an STL requirement and a test case. The test case is parametrized by a set of parameters which define the test space. The objective is to produce evidence that the system satisfies the STL formula to a desired probability level over the entire test space. The automatic testing algorithm selects a specific test case from the test space and runs this in the simulator. The STL monitor evaluates the simulation results against the STL requirement producing an STL robustness score. The GP model considers the STL robustness as an unknown function of the test case parameters and estimates the expected value and uncertainty of this function over the entire test space. The STL robustness score from a simulation is added to the GP model as an observation of this unknown function. This is shown to the right in Figure 4, where the GP estimate of the STL robustness is plotted in orange against the scenario parameters "*maneuver angle*" and "*obstacle speed*". The black dots show observations. The GP model is used to select the next test case to simulate in a smart way, where it favours cases which have low robustness or high uncertainty. This adaptive sampling is prominent in Figure 4, where the black dots are denser in areas with low robustness. After each simulation, two termination criteria are checked: If a simulation which falsifies the requirement is found, the algorithm terminates in a falsified state. This is the case in Figure 4, where falsifying scenario is identified at speed = 10m/s and maneuver angle = 27 degrees. If no falsifying scenario is found, the termination is determined by the desired level of confidence. If, for instance, 99% confidence that the system satisfied the requirement is desired, then the algorithm terminates in a verified state if the lower 99% confidence interval of the GP model is greater than zero over the entire test space. This shows that the simulation-based testing is completely automatic after the test case and requirement are defined.

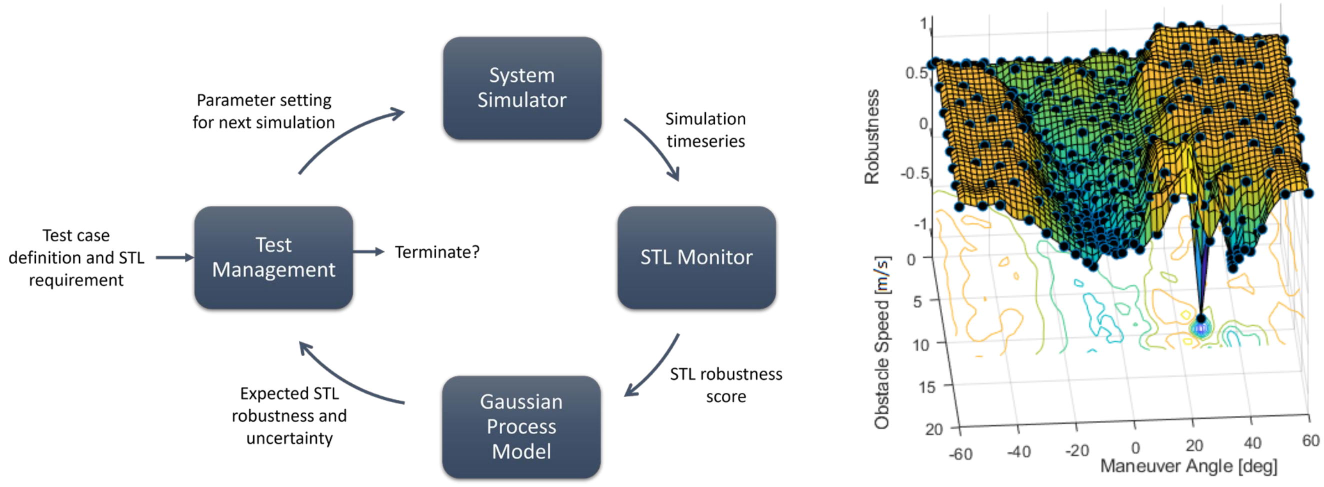


Figure 4: Left: Overview of the automatic test method of Torben et al. (2022). Right: Estimated STL robustness surface after a falsifying scenario is found.

LIMITATIONS OF FORMAL METHODS

The previous sections have shown some of the possibilities FMs have to offer. However, FMs not fit for all types of problems. In this section we will briefly discuss some limitations of FMs.

Scalability is a well-known limitation of FMs. Most FMs operate on discrete, finite-state models. As the number of variables in a model increases, the number of possible states increases exponentially. This is known as the *state space explosion* problem. Since a model checker exhaustively checks all possible executions, this limits the size of models which can be model checked. This has, however, been improved with recent innovations in model checking algorithms and the large increase in computational power.

FMs have traditionally also had limited support for systems with continuous dynamics, as they are often based on finite-state models. An approach to use FMs on continuous systems is to create a discrete approximation. However, this often ends up in huge state spaces which limits the size and accuracy of the discrete approximation. Several automated theorem provers have inherent support for continuous systems, by being able to use known theories on real-valued numbers in their reasoning.

Related to the two previous limitations is the *reality gap* problem. Formal verification is performed on a mathematical model of reality. In order for the verification of the mathematical model to have value as verification evidence, it needs to be an accurate representation of the real system. For some applications, such as logic circuits or software generated from a formal specification, the mathematical model is highly accurate. However, due to the scalability issues and limited support for continuous dynamics, some systems need to be simplified before they are subject to formal verification. Validating the fidelity of the simplified model is crucial in order to trust the results of a formal verification.

Finally, there is a long-standing challenge of limited uptake of FMs among professionals. We believe this is mainly caused by the perception that FMs are cumbersome, difficult and time-consuming. The majority of the background theory which FMs are built on originate from theoretical computer science and discrete mathematics, which is unfamiliar to many. This can make FMs seem intimidating to begin with. Also, taking on a formal development process often requires investing considerably more time in the design phase. However, this time is often returned in the verification and operational phases.

CONCLUSIONS

MASS are approaching reality, introducing a new level of complexity and criticality to maritime control systems. Given that the maritime industry already struggles with managing the complexity of current control systems, it is evident that new methods for development and verification are necessary to enable a safe and efficient introduction of autonomy. We have therefore proposed FMs as a candidate to aid the design and verification of safe MASS. We have discussed the needs the maritime industry faces and given three specific use cases demonstrating how FMs can be applied to meet these needs. Our conclusion is that FMs has the potential to meet many of the needs, despite some limitations. We therefore encourage further research into FMs for MASS.

ACKNOWLEDGEMENTS

The work is partly sponsored by the Research Council of Norway through the Centre of Excellence funding scheme, project number 223254, AMOS, and project ORCAS with project number 280655.

REFERENCES

Abrial, Jean Raymond. Modeling in Event-b: System and Software Engineering. Cambridge University Press, 2011.

Asarin, Eugene, et al. “Recent Progress in Continuous and Hybrid Reachability Analysis.” Proceedings of the 2006 IEEE Conference on Computer Aided Control Systems Design, CACSD, IEEE, 2007, pp. 1582–87.

Cimatti, Alessandro, et al. “NuSMV 2: An Opensource Tool for Symbolic Model Checking.” Lecture Notes in Computer Science, vol. 2404, 2002, pp. 359–64.

de Moura, Leonardo, and Nikolaj Bjørner. “Z3: An Efficient SMT Solver.” International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, 2008, pp. 337–40.

DNV. Rules for Classification of Ships Part 6 Chapter 22 - Enhanced System Verification. 2013.

DNV. Recommended Practice: Integrated Software Dependent Systems (ISDS). 2017.

DNV. Class Guideline: Autonomous and Remotely Operated Vehicles. 2018.

DNV. Recommended Practice DNVGL-RP-0510: Framework for Assurance of Data - Driven Algorithms and Models. 2020.

DNV. RULES FOR CLASSIFICATION Ships Part 6 Additional Class Notations Chapter 3 Navigation, Maneuvering and Position Keeping. 2021.

Fainekos, Georgios E., and George J. Pappas. “Robustness of Temporal Logic Specifications for Continuous-Time Signals.” Theoretical Computer Science, vol. 410, no. 42, Elsevier B.V., 2009, pp. 4262–91.

Foster, Simon, et al. “Towards Deductive Verification of Control Algorithms for Autonomous Marine Vehicles.” Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems

Holzmann, G. J. “The Model Checker SPIN.” IEEE Transactions on Software Engineering, vol. 23, no. 5, 1997, pp. 279–95

IECTC. IEC 60050: International Electrotechnical Vocabulary. 2002.

IMO. COLREGs - International Regulations for Preventing Collisions at Sea. 1972.

IMO. Guidelines for Vessels and Units with Dynamic Positioning (DP) Systems. MSC.1/Circ.1580, 2017.

IMO. Outcome of the Regulatory Scoping Exercise For The Use Of Maritime Autonomous Surface Ships (MASS). 2021.

ISO. “Road Vehicles — Functional Safety.” ISO 26262, 2011.

ISO. “Road Vehicles - Safety of the Intended Functionality.” ISO 21448, 2019.

Jeannin, Jean-baptiste, et al. “Formal Verification of ACAS X, an Industrial Airborne Collision Avoidance System.” International Conference on Embedded Software, 2015, pp. 127–36.

Johansen, Tor A., et al. “Experiences from Hardware-in-the-Loop (HIL) Testing of Dynamic Positioning and Power Management Systems.” OSV Singapore, 2007, pp. 41–50.

Krasowski, Hanna, and Matthias Althoff. “Temporal Logic Formalization of Marine Traffic Rules.” IEEE Intelligent Vehicles Symposium (IV), 2021, pp. 186–92.

Larsen, Kim G., et al. “UPPAAL in a Nutshell.” International Journal on Software Tools for Technology Transfer, vol. 1, 1997, pp. 134–52.

Lecomte, Thierry, et al. “Formal Methods in Safety Critical Systems.” Safety and Reliability, vol. 11, no. 4, 1991, pp. 6–17.

Maler, Oded, and Dejan Nickovic. “Monitoring Temporal Properties of Continuous Signals.” Lecture Notes in Computer Science, vol. 3253, 2004, pp. 152-166.

NMD. Føringer i Forbindelse Med Bygging Eller Installering Av Automatisert Funksjonalitet, Med Hensikt å Kunne Utføre Ubemannet Eller Delvis Ubemannet Drift. 2020.

Park, Jinwook, and Jinwhan Kim. “Autonomous Docking of an Unmanned Surface Vehicle Based on Reachability Analysis.” International Conference on Control, Automation and Systems, 2020, pp. 962–66.

Paulson, Lawrence C. Isabelle: A Generic Theorem Prover. Vol. 828, Springer Science \& Business Media, 1994.

Pedersen, Tom Arne, et al. “Towards Simulation-Based Verification of Autonomous Navigation Systems.” Safety Science, vol. 129, no. December, Elsevier, 2020, p. 104799.

Pnueli, Amir. “The Temporal Logic of Programs.” Annual IEEE Symposium on Foundations of Computer Science, IEEE, 1977, pp. 46–57.

Sangiovanni-Vincentelli, Alberto, et al. “Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems.” European Journal of Control, vol. 18, no. 3, Elsevier, 2012, pp. 217–38.

Shokri-Manninen, Fatima, et al. “Formal Verification of COLREG-Based Navigation of Maritime Autonomous Systems.” Lecture Notes in Computer Science, Software Engineering and Formal Methods, 2020

Smogeli, Øyvind. “Managing DP System Software - A Life-Cycle Perspective.” IFAC-PapersOnLine, vol. 48, no. 16, 2015, pp. 324–34.

Smogeli, Øyvind, et al. “Open Simulation Platform – An Open-Source Project for Maritime System Co-Simulation.” 19th Conference on Computer and IT Applications in the Maritime Industries, 2020, pp. 239–53.

Smogeli, Øyvind, and Jon Espen Skogdalen. “Third Party HIL Testing of Safety Critical Control System Software on Ships and Rigs.” Offshore Technology Conference, no. 2011, One Petro, 2011, pp. 839–45.

Sørensen, Asgeir J. “A Survey of Dynamic Positioning Control Systems.” Annual Reviews in Control, vol. 35, no. 1, 2011, pp. 123–36.

Torben, Tobias R., et al. “Automatic Simulation-Based Testing of Autonomous Ships Using Gaussian Processes and Temporal Logic.” Journal of Risk and Reliability, To be published, 2022.

Yan, Rongjie, et al. “Formal Collision Avoidance Analysis for Rigorous Building of Autonomous Marine Vehicles.” Embedded Systems Technology, edited by Yuanguo Bi et al., Springer Singapore, 2018, pp.