Autonomous Sailing with Sim-to-Real Reinforcement Learning

Master Thesis

Bink, K.J.A.

Delft University of Technology

TU Delft

Thesis for the degree of MSc in Marine Technology in the specialization of Ship Hydrodynamics

Autonomous Sailing with Sim-to-Real Reinforcement Learning

by

Bink, K.J.A.

Performed at

MARIN

This thesis (**MT.23/24.019.M**) is classified as confidential in accordance with the general conditions for projects performed by the TU Delft.
To be defended publicly on Thursday February 28, 2024 at 10:00 AM.

Company supervisors
Responsible supervisor: Dr. Bülent Düz
E-mail: B.Duz@MARIN.nl

Thesis exam committee
Chair/Responsible Professor: Prof. Dr. Gabriel Weymouth
Staff Member: Prof. Dr. Rudy Negenborn
Staff Member: Ir. Jaap Gelling
Company Member: Dr. Bülent Düz

Author Details
Student number: 4566319
Author contact e-mail: kikibinkk@gmail.com

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

When I first came across this project proposed by MARIN, it immediately caught my attention. I had been fascinated by the possibilities of machine learning, specifically RL, for a while. Combining this into a project involving the interesting complexities of sailing and controls was the perfect scenario for me.

The thing that fascinated me about RL was the idea that machines could learn in the same way that humans and animals learn and the potential this unlocks to solve increasingly complicated problems. In the next few decades, RL and other machine learning techniques could become crucial in addressing the engineering, economic, medical, and climate issues facing the 21st century. Before starting my thesis, it almost seemed as if using RL was as easy as telling the machine its goal, and without telling it how to reach that goal it would figure out how to do it. However, I was quickly faced with the (not entirely unexpected) reality that using RL for controls was much more complex. It became clear that an understanding of the dynamical system is crucial to perform validations and evaluations of the methods and resulting controls. Luckily, this is what made this thesis challenging and enjoyable.

I would like to thank my supervisor, Gabe, for his invaluable guidance and teaching me what conducting good research looks like. Our weekly meetings were an immense help to stay inspired and motivated throughout the many challenges encountered. Next, I would like to thank my company supervisor, Bulent, for sharing his knowledge of Reinforcement Learning and always making the time to answer my questions. I also want to express my gratitude to MARIN for providing me the chance to work on this project, and especially for the unique opportunity to conduct tests in the Offshore Basin, which was a great experience in itself. Finally, I would like to thank my family and friends for the support and encouragement during the busy times of performing this research.

Bink, K.J.A.
Delft, February 2024

Abstract

Facing the critical challenge of reducing greenhouse gas (GHG) emissions in the maritime industry, this thesis explores the potential of smart control systems using Reinforcement Learning (RL) for autonomous sailing. Traditional controls for sailing fall short in navigating the complex, dynamic conditions of maritime environments. RL has shown to be effective for continuous control applications in these types of conditions, however, primarily in simulated environments. Therefore, this study aims to show the potential of RL for autonomous sailing control (ASC) by means of a small scale project. A fast-time simulation of an Optimist is used to train the sailing controls required to reach an upwind target. The controls are then transferred to a robotized Optimist in a real-world environment to test the transferability of the simulation trained controls. First, the *reality gap*, or modelling error, between the simulation and real-world environment is quantified to be able to assess the performance of the used techniques to bridge the existing gap. The sim-to-real techniques of Domain Randomization (DR) and the addition of observation noise (ON) are applied during the training process. To test the effectiveness of the trained RL controls, the best performing ones in the simulation are selected and tested in the real-world environment. The performance of the RL controlled Optimist is compared to state-of-the-art controls in robotic sailing. Their performances are measured and compared by means of success rate and a physics-based metric that calculates the efficiency of the sailboat to use the power of the wind to propel itself, called the energy ratio. The results show that the RL controls are highly successful in the sailing simulation, however, the transfer to the real-world remains a major challenge. DR does improve the sim-to-real transfer, resulting in an agent that is able to reach a 100% success rate throughout 12 runs in the real-world environment.

Contents

List of Figures

List of Tables

1

Introduction

The maritime industry faces a major challenge in reducing its Green House Gas (GHG) emissions. In 2018, global shipping emissions reached 1,076 million tonnes of CO2, accounting for about 3% of the total global emissions caused by human activities [1]. In order to align with the emission reduction goals set out in the United Nation's 2015 Paris Agreement, the International Maritime Organization (IMO) launched an emission reduction strategy which outlines the goal to reduce the total annual GHG emissions by at least 50% by 2050 compared to 2008 levels. The strategy also emphasizes the importance of trying to completely eliminate these emissions in the future [2].

One possible solution that has (re)gained attention is the use of wind-assisted ships. These ships harness the power of wind to reduce the propulsive power generated by fossil fuels. Nevertheless, the control of wind-assisted ships is challenging due to their low control authority and the advance planning that is required. This challenge is further compounded by the constantly changing and unpredictable maritime environment it has to operate in, requiring precise maneuvering and the optimal utilization of sails. Controlling these ships requires specialized skills, but due to its novelty, there are not many people with expertise in this area [3].

To overcome these challenges, there is a need to explore innovative control systems that can efficiently and autonomously operate wind-assisted ships. Smart control systems, which leverage advanced technologies such as artificial intelligence (AI), have shown promise in various control applications, including the control of power-propelled ships [4], [5]. These systems are able to integrate real-time data, optimize sailing strategies, and adapt to changing environmental conditions. As a result, they improve efficiency and performance, and offer a potential solution to the complex control problem presented by emerging technologies such as autonomous wind-assisted ships.

The potential of smart control systems for wind-assisted ships is clear, but there is a significant research gap when it comes to applying them in real-world environments. As a result, there has been limited support and funding for exploring newer methodologies on a larger scale. One such methodology of interest is Reinforcement Learning (RL), a subset of machine learning. RL offers promise in addressing complex control problems by allowing systems to learn optimal strategies through trial-and-error in dynamic environments [6]. RL has been successfully applied to various control problems such as flying drones [7], [8], autonomous driving [9], and quadruped robots [10], showing potential in dealing with low control authority and operating in dynamic and stochastic environments.

Since the interest among researchers in using RL for complex control systems is relatively new, many challenges remain that must be addressed to ensure robustness and safety in systems that utilize RL for control. One of these challenges is the need for autonomously and safely gathering large volumes of data necessary for the RL agents to learn robust policies. Simulations are often used to overcome this by enabling fast data collection and iteration. However, due to simplifications and assumptions that have to be made during the modelling process, modelling errors are inherent to simulations. Therefore, simulations are unable to accurately capture the complexity of the real world, often referred to as the *reality gap* [11], [12]. This poses difficulties for applying the RL controls trained in simulations to real-world scenarios. Researchers are working on bridging this gap through more realistic simulations or methods for transferring knowledge from simulation to reality, referred to as *sim-to-real transfer* [11]. Sim-to-real transfer has not been considered in applications that deal with

the low control authority, and the dynamic and stochastic environment present in autonomous sailing applications.

To address this research gap and demonstrate the viability of RL in the real-world for the control of underactuated systems such as wind-assisted ships, it is necessary to initiate small-scale projects to test and showcase its capabilities. For this reason, this thesis proposes a project focused on developing a control system using RL for a small sailboat in a simulation and transferring it to the Offshore Basin at MARIN. The choice of a small sailboat is advantageous since it can be tested and controlled within the confines of a basin that already has sensor systems and cameras set-up, reducing the complexity and risks associated with open water trials. The small sailboat used is an Optimist due to its robustness and stability, which slightly simplifies the complex process of learning to sail. Moreover, there is existing knowledge and experience in autonomous sailing control (ASC) of small sailboats, which can serve as a foundation for developing control strategies and as a benchmark for evaluating the RL controlled Optimist.

To verify that the Optimist is able to sail remotely in the basin, preliminary tests were done with manual control. The necessity of using a simulation was restated during preliminary tests to verify that the Optimist was able to sail remotely in the basin, as it was observed that setting up and completing took about 3 to 4 minutes for each run. Looking at a prior study by Yu et al. [13] on the use of RL for trajectory tracking control of autonomous underwater vehicles (AUV), it required about 3000 episodes of training to learn a robust policy, which would amount to approximately 250 hours of training time in the basin. This estimate is very conservative when considering that in the study by Yu et al. the trajectory was already provided and only path following was considered. The maximum allotted basin time is about 60 hours, which is unlikely to be enough for the RL-controlled Optimist to learn an effective policy. The successful implementation of RL in controlling a sailboat in a simulation and transferring it to a real-world environment would not only showcase the capabilities of this approach for underactuated systems but also lay the groundwork for further research and funding in this area.

Therefore, the aim of this thesis is to demonstrate the application of RL for sailing when trained in a simplified simulation and employed in the real world. By doing so, it seeks to unlock opportunities for the advancement of smart control systems that can facilitate sustainable practices within the maritime industry.

1.1. Research question

The main research question that will be addressed in this thesis is:

> How robust, stable, and effective are RL strategies for sailing upwind when trained in a simulated model and applied in real life?

To answer this question, RL agents were trained to sail to an upwind target in a fast time-domain simulation. The resulting policies were transferred to a robotized Optimist in the Offshore Basin at MARIN as shown in Figure 1.1, where the performance of these agents was evaluated and compared to the state-of-the-art.

Figure 1.1: Remotely sailing Optimist in the Offshore Basin at MARIN

1.2. Structure

First off, the literature review is structured into four parts. In chapter 2, the theory of sailing is discussed to set up autonomous sailing as a control problem. This chapter also provides an overview of previous research methods to identify the state-of-the-art for autonomous sailing control (ASC), which can be later used as a benchmark to evaluate the effectiveness of the RL-controlled sailboat. chapter 3 goes into the fundamentals of RL theory and highlights the recent advancements in the field. Various RL algorithms are discussed to identify which ones are the most suitable to consider for ASC. Next, chapter 4 reviews the literature on sim-to-real transfer, highlighting recent research that is relevant to this application. It sheds light on the methods and approaches used to bridge the gap between simulation and real-world scenarios.

After the extensive literature review, the methods used to address the research question are discussed in chapter 5. The complete system to train and evaluate the RL-controlled Optimist in both the simulation and basin are discussed, as well as the methods used to evaluate the sim-to-real transfer.

The results are presented and discussed in chapter 6. The concluding remarks, in chapter 7, examine the study's context within the broader field, exploring implications and limitations. Finally, chapter 8 proposes potential directions for future research.

2

Sailing as a control problem

Sailing was one of the first methods created by humans for transport over the water. Wind is always readily available which makes it a sustainable way of transportation. However, this dependence on the wind makes that the control is not as straightforward opposed to power-driven vessels. Reaching a target is not as simple as setting a heading in that direction, because going directly into the wind is impossible. Besides wind, the water that the sailboat moves through also plays a major role. The combination of the aero- and hydrodynamic forces is what makes sailing possible. Luckily, sailors do not need to know the exact physics to get from A to B. An experienced sailor considers the changing environmental conditions, such as wind direction, wind speed, waves, and currents. Depending on these conditions, the skipper makes routing decisions and performs certain manoeuvres to sail to the desired destination. As a result, replacing all these tasks to be carried out autonomously is an immensely complex task. [14]

In this chapter, section 2.1 will start off with an overview of the basics of sailing and the most used terms to make sure the non-sailor is able to follow along smoothly. After which it will further expand on the physics and challenges of upwind sailing. The different levels of control associated with sailing upwind will be explained in section 2.2, with that setting up sailing as a control problem. section 2.3 offers an insight into previous work on sailing robotics and the current state-of-the-art sailing robot control for sailing upwind.

2.1. What are the relevant physics of sailing?

Wind is the main propulsor of a sailboat. When discussing wind in sailing, it is important to make the distinction between true and apparent wind. **True wind** refers to the actual direction and speed of the wind in relation to the Earth's surface. **Apparent wind** is the wind experienced by a moving sailboat, which is influenced by the boat's own speed and direction. This is shown in the diagram in Figure 2.1, with true wind denoted as $\vec{V_{tw}}$, apparent wind as $\vec{V_A}$ and boatspeed as $\vec{V_S}$. The apparent wind speed and direction can be calculated as follows:

$$\vec{V_A} = \vec{V_{tw}} - \vec{V_S} \tag{2.1}$$

To effectively adjust the sails and optimize the performance of a sailboat, the concept of apparent wind is important to understand.

Throughout this work, the names of the ship motions in 6 degrees of freedom (DOF) and the (local) coordinate system used are shown in Figure 2.2 below.

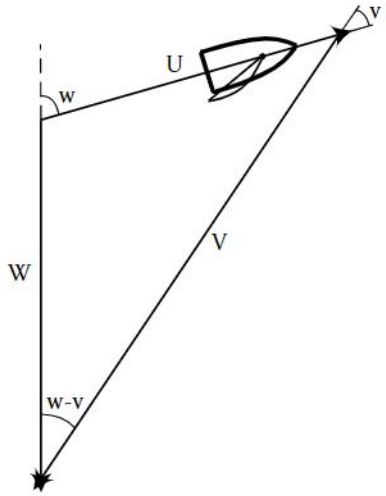


Figure 2.1: Wind triangle showing the true and apparent wind vectors with respect to the boat speed [14]

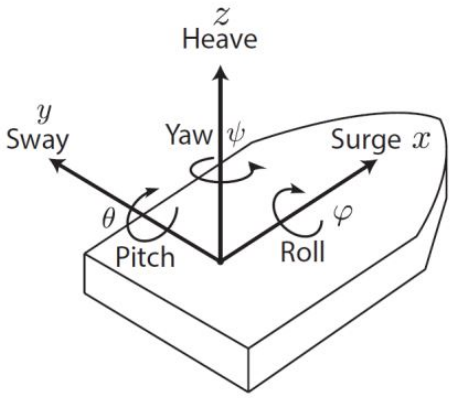


Figure 2.2: Ship motions in 6DOF [15]

2.1.1. Points of sail

The boatspeed and direction of a sailboat are mainly determined by the shape of the sails, the direction of the wind, and the heading angle of the boat. The heading angle of the boat relative to the direction of the wind is called the point of sail. The standard points of sail are shown in Figure 2.3.

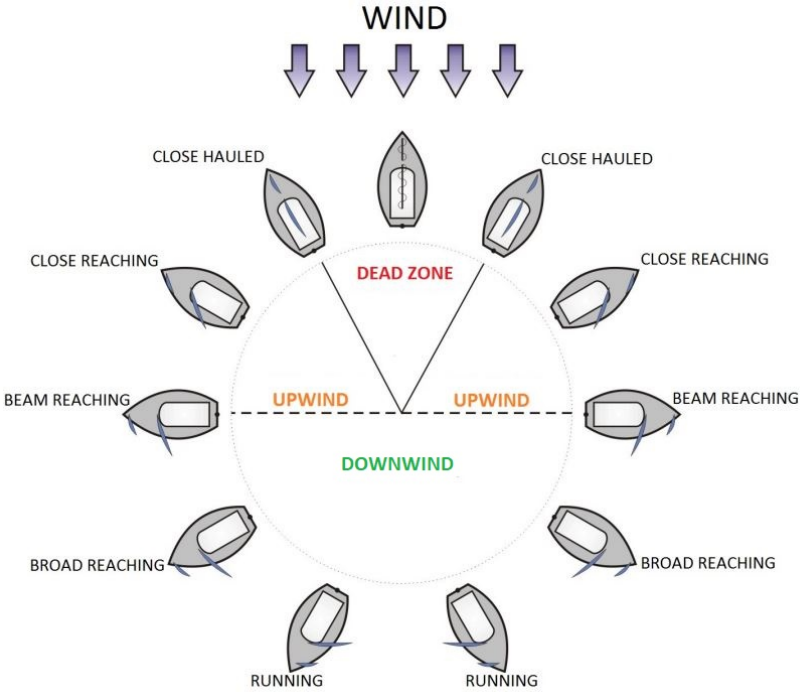


Figure 2.3: (a) Points of sail [16]

2.1.2. Running downwind

In its most simple form, downwind sailing will happen automatically when you put a sailboat in the water as it will be pushed ahead by the wind. However, optimizing this process to reach a downwind target as quickly as possible poses a more challenging task. Achieving this involves utilizing the sail as a parachute to generate a drag force and reducing the resistance of the water on the boat as much as possible. A sailing vessel will accelerate until the forces exerted by the wind and water balance each other. These forces can be calculated by applying Newton's impact theory, yielding the following equations [14]:

$$\begin{aligned} F_D(wind) &= \frac{1}{2} C_D(sail) \cdot A_s \cdot \rho_a \cdot V_A^2 \\ F_D(water) &= \frac{1}{2} C_D(hull) \cdot A_h \cdot \rho_w \cdot V_S^2 \end{aligned} \tag{2.2}$$

$F_D(wind)$ is the drag force due to the wind hitting the sail and $F_D(water)$ is the resistance of the water on the moving hull. V_A and V_S are the apparent wind speed (aws) and the speed of the vessel respectively. C_D is the drag coefficient, it summarizes a multitude of corrections. For the sail this depends on the type and geometry of sail. $C_D(hull)$ is dependent on the geometry of the hull but also includes the rudder and centerboard. A more streamlined boat will have a lower C_D value. When running downwind, at some point it will feel as if the wind completely vanishes since the boat speed approaches the wind speed. This can be explained by Equation 2.1 for apparent wind speed V_A, where the boatspeed is subtracted from the true wind speed V_{tw}. As a consequence, when running downwind, the boat speed can never exceed the wind speed.

Thus, with some simplifications and assumptions for the drag coefficients, it is fairly straightforward to figure out the speed of a vessel running downwind. With any other point of sail, this gets a lot more difficult since lift forces generated by the sail, centerboard and rudder start playing a role. Especially the upwind courses such as close reaching and close hauling rely on lift to create forward speed.

2.1.3. What makes upwind sailing complex compared to downwind sailing?

For upwind sailing, instead of using the drag force on the sail as propulsion, the forward power is now generated by lift forces. The sail generates lift by using the Bernoulli principle and the interaction of

the flow field of the wind around its curved shape, see Figure 2.4. As the wind flows over the sail, it accelerates on the leeward side of the sail, creating lower pressure according to Bernoulli's principle. Simultaneously, the wind slows down on the windward side, resulting in higher pressure. This pressure difference generates lift, similar to how an airplane wing generates lift during flight. Attached flow on both sides of the sail is optimal when going upwind, as depicted in state 2 in Figure 2.4. To achieve this, maintaining an optimal angle of attack for the apparent wind is necessary to prevent flow separation or *"stalling"*. If the angle of attack on the sail gets too small, the sailboat loses its ability to generate lift force, resulting in a decrease in forward speed, known in sailing as being *"in irons"*. This is depicted in Figure 2.3 as the 'dead zone'. In the downwind courses, the sailboat is able to utilize a combination of lift and drag from the wind, while the airflow may partially or completely separate on the leeward side of the sail. Balancing these forces in the right way can potentially allow the boatspeed to exceed the windspeed. The beam reaching course offers the best conditions for achieving this balance, as both drag and lift contribute to propelling the boat forward [17]. Going upwind, the drag force of the wind counteracts forward propulsion but due to careful design of the sail, the drag force going upwind is kept to a minimum. The forces on the sail are generated at the price of heeling and/or leeway (sailing term for sideways drift). The centerboard (or "keel" depending on the type of boat) generates lift in the other direction to counter this leeway and heeling moment by acting as a hydrofoil. The resulting forces on the centerboard and sail are shown in Figure 2.5. The counteracting forces result in a forward motion of the sailboat.

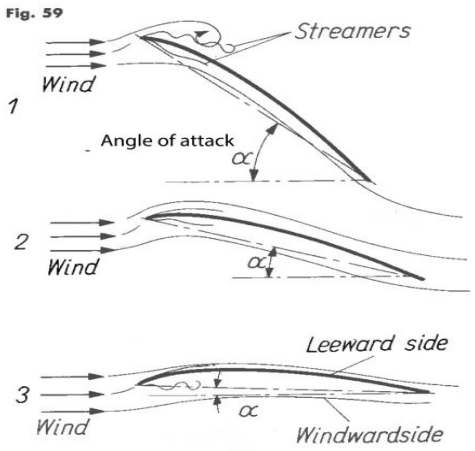


Figure 2.4: Airflow around sail [17]

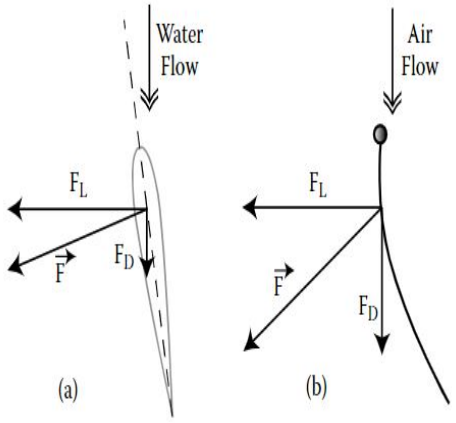


Figure 2.5: Force diagram of (a) centerboard and (b) sail [14]

The aero- and hydrodynamic interactions enable the sailboat to create forward speed. Due to these interactions, predicting the boatspeed when sailing upwind is not straightforward.

2.1.4. How is sailing modeled?

Careful balancing of the forces and moments is necessary to be able to predict speed and model sailing. To predict the speed, models have been developed for this such as velocity prediction programs (VPPs). VPPs exhibit varying degrees of complexity, depending on factors such as the intended use, available data, modeling techniques, and desired accuracy. Simple VPPs may consider only 3 degrees of freedom (DOF), relying on static or quasi-static assumptions and steady-state wind conditions. In contrast, more complex VPPs incorporate all 6DOF, dynamic effects, and environmental factors like time-varying winds, waves, and current. They also account for sail interactions and controls, taking into consideration the complex aerodynamics of sails and optimizing control settings. The complexity of VPPs is often a trade-off between accuracy and computational resources, with simpler models still providing valuable insights for recreational sailing and performance estimation. [18]

One of the most well known VPP tools under sailors is developed and maintained by the Offshore Racing Congress (ORC) [19]. The ORC VPP is a program mainly used to calculate racing yacht handicaps based on a mathematical model of the physical processes embodied in a sailing yacht. It is responsible for the development and maintenance of rating and classification standards that can be

used to rate nearly every kind of boat in a scientific and unbiased manner. An approach to handicapping was first developed in 1978 with the H. Irving Pratt Ocean Racing Handicapping project [20]. This work resulted in the Measurement Handicapping System (MHS) that was used in the United States. The aerodynamic model was subsequently revised by George Hazen [21] and the hydrodynamic model was refined over time as the Delft Systematic Yacht Hull Series was expanded [22]. Many researchers such as Philpott [23], have updated the VPP in the following years, the ORC has made the effort to maintain an up-to-date handicapping system.

The most recent ORC VPP program creates a computer simulation of the boat's performance based on scientific research of boat hulls in hydrodynamic basins, sails in aerodynamic tunnels and measurements taken on real boats as well as computer fluid dynamics (CFD) tools currently available. The VPP is comprised of two parts: the **solution algorithm** and the **boat model**. The solution algorithm must find an equilibrium condition for each point of sailing, where the sail forces and hull forces shown in Figure 2.6 are balanced.

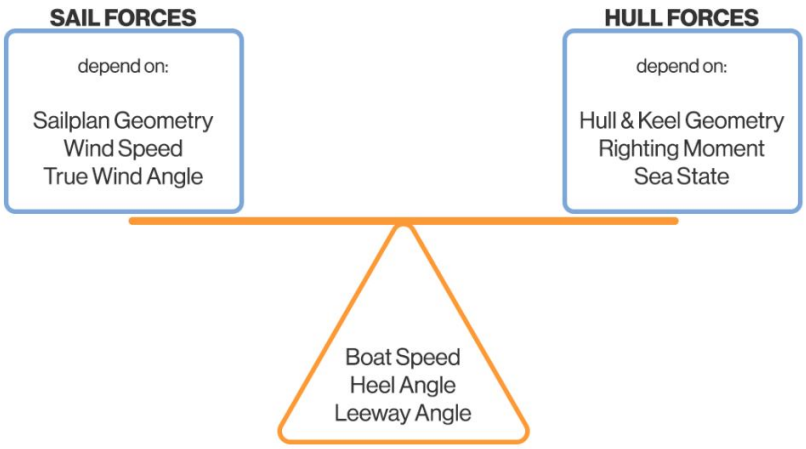


Figure 2.6: Force balance seesaw [19]

A force diagram of the sail and hull forces with the wind triangle is shown in Figure 2.7.

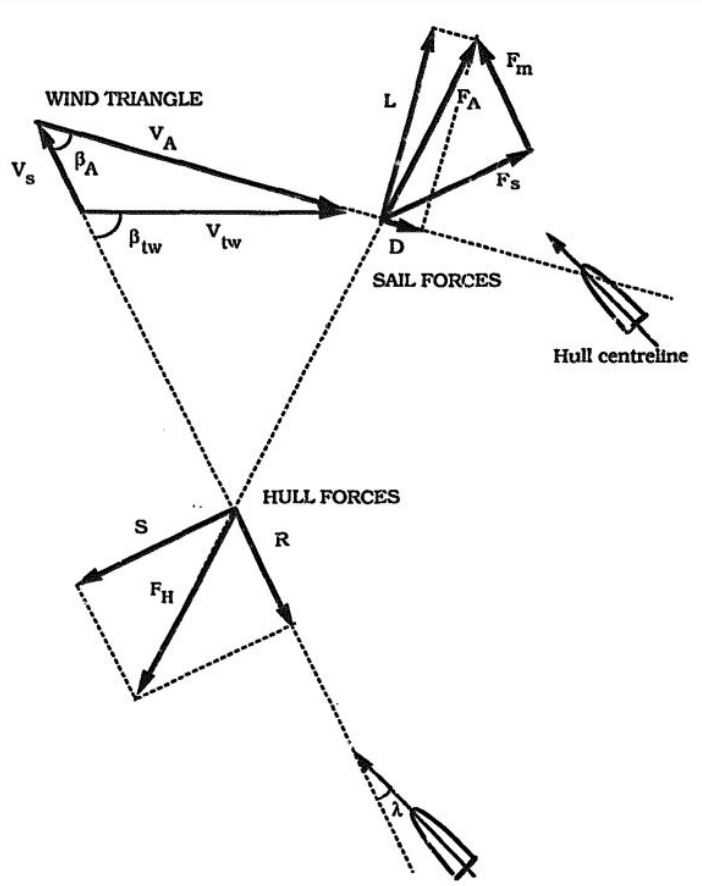


Figure 2.7: Force diagram including the sail and hull forces with respect to the wind triangle. [23]

The wind triangle in Figure 2.7 uses V_A and β_A to denote the apparent wind speed and angle respectively, V_{tw} and β_{tw} for the true wind speed and angle, and V_S for the boat speed. The remaining variables depicted in the diagram will be introduced throughout the following.

The **boat model** of the general VPP determines the steady state conditions by satisfying equilibrium equations. The level of complexity of the VPP can be altered, the following is largely based on the model of Philpott [23]. Two important assumptions are made in this model:

1. **Vertical force balance is always satisfied.** The weight of the vessel will always equal the buoyancy. In reality, the motion of a vessel will cause a change in displacement but the buoyancy force will change very quickly along with this change so this is reasonable to neglect.
2. **Pitching moment balance is always satisfied.** The force on the sails will tend to raise the stern and depress the bow which causes an imbalance in the fore/aft buoyancy, this will in turn resist the pitch. Just like the vertical force balance, the moment caused by the restoring buoyancy force changes quickly so one can assume that the pitching moment is always satisfied.

Sail forces The expression for the drag, D, and lift L forces on the sail (denoted in Figure 2.5(b) as F_D and F_L) are based on Newton's impact theory, similarly to the drag forces of downwind sailing in Equation 2.2.

$$\begin{aligned} L &= \frac{1}{2} C_L A \rho_a V_A^2 \\ D &= \frac{1}{2} C_D A \rho_a V_A^2 \end{aligned} \tag{2.3}$$

With A as the sail area and ρ_a the density of the air. C_L and C_D encompass a multitude of corrections, in Kerwin's model, they are a function of β_{aw}, trim, and the shape of the sail. The trim refers to the

influence of the sailing crew on the shape and angle of the sail, they can change this by sheeting in or out for example. Exactly how the force coefficients depend on these variables is a widely researched topic, where most reliable results are obtained by wind-tunnel experiments [18]. The lift and drag coefficients of an Optimist sail have been obtained this way in the study *'Design of a Foiling Optimist'* by Andersson et al. [24]. Tests in a wind tunnel were performed with an Optimist sail at 1/4 scale, their test set-up and the resulting coefficients are shown in Figure 2.8 and Figure 2.9 respectively.

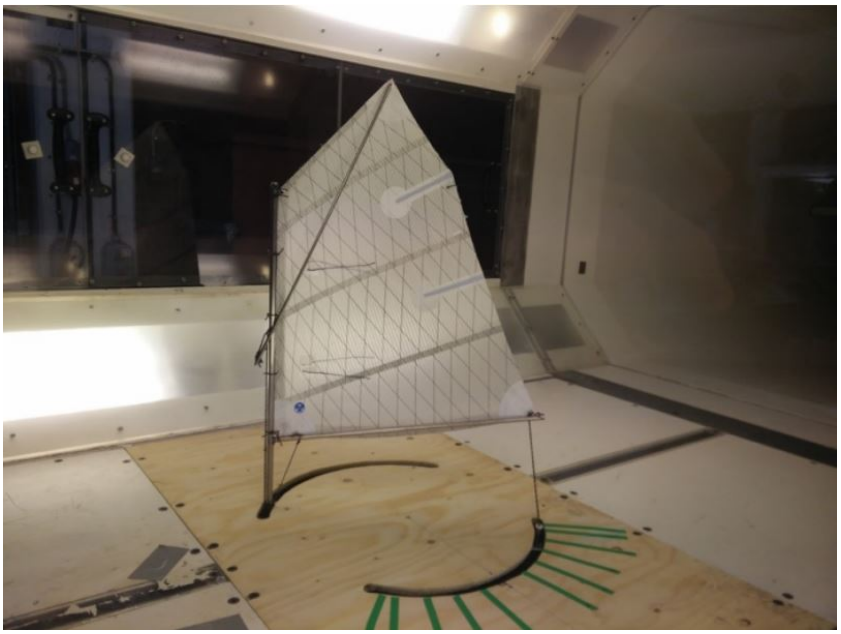


Figure 2.8: The Optimist sail at 1/4 scale in the wind tunnel, from *'Design of a Foiling Optimist'* [24]

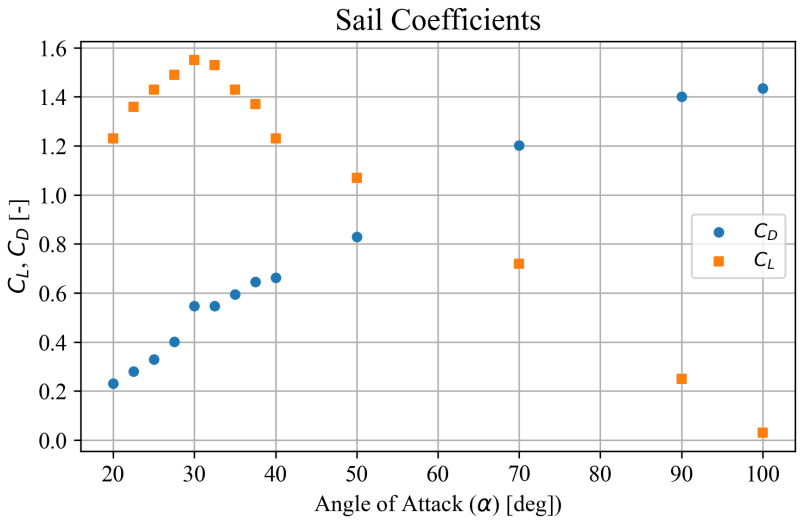


Figure 2.9: Lift and drag coefficients of the Optimist sail, from *'Design of a Foiling Optimist'* [24]

Looking at the force diagram shown in Figure 2.7, the drive force F_m and side force F_s can be expressed as follows:

$$\begin{aligned} F_m &= (F_L cos\beta_{aw} + F_D sin\beta_{aw})cos\phi sin\lambda + (F_L sin\beta_{aw} - F_D cos\beta_{aw})cos\lambda \\ F_s &= (F_L cos\beta_{aw} + F_D sin\beta_{aw})cos\phi cos\lambda - (F_L sin\beta_{aw} - F_D cos\beta_{aw})sin\lambda \end{aligned} \tag{2.4}$$

With:

- $\lambda =$ leeway angle
- $\phi =$ heeling angle

Hull (hydrodynamic) forces The total hydrodynamic force consists of the drag force F_D (or denoted as resistance, R) and lift forces on the hull, rudder and centerboard, induced by the interaction of the moving vessel through water. The drag force consists of several components:

- **Viscous drag:** friction of the water flowing over the surface of the hull and appendages.
- **Residuary drag:** due to the creation of surface waves.
- **Heel drag:** change in wetted surface and immersed hull shape as the ship heels.
- **Induced drag:** created when the hull, keel and rudder produce side force.
- **Raw drag:** due to the ships motion in a seaway.

Current hydrodynamic models use the results of model towing tank tests and CFD data in constant research of these values comparing it with real boat performance. [19]

The lift forces are separated into three parts: the hull lift, the rudder lift, and the centerboard (or keel) lift. The hull lift is largely ignored, since it is negligible at small leeway angles; the maximum leeway angle of a typical sailboat is about 6deg [14]. The side forces produced by the rudder and centerboard (shown as F_L in Figure 2.5(a)) are perpendicular to their velocity, similarly to the lift of the sail. The total sideforce produced by the rudder and keel, considering the heel angle ϕ, can given by:

$$S = \frac{1}{2}\rho_w V_S^2 (C_r A_r + C_k A_k) cos\phi \tag{2.5}$$

With A_r and A_k being the cross sectional areas of the rudder and keel and ρ_w the water density. The lift coefficients, C_r for the rudder and C_k for the keel, are dependent on the respective shapes and on the angles of attack.

Finally, the roll and yaw moment balances of a sailboat in equilibrium need to be satisfied. The heeling moment has contributions from the sideways sailforce and the sideforces generated by the rudder and keel. The lever arms are denoted with z_0, z_r and z_k respectively. z_0 is dependent on the 'centre of effort' of the wind on the sail. With this, an expression for the total heeling moment can be denoted as follows:

$$M_1 = M_s + M_r + M_k = z_o(F_m + F_s) + \frac{1}{2}\rho_w V_S^2 (C_r A_r z_r + C_k A_k z_k) \tag{2.6}$$

The righting moment M_2 is generated by the hydrostatic force on the hull and the weight of the crew (W_c) - if the crew is on the windward side of the sailboat. The lever arm of the hull weight (W_h) depends on the distance of the centre of gravity (CoG) of the hull and the metacenter of the sailboat and is denoted as GZ. d is the distance from the CoG to the location of the crew. The expression for the righting moment can be formulated as:

$$M_2 = M_h + M_c = GZ \cdot W_h + d \cdot W_c \tag{2.7}$$

Next, the yaw moment has 3 hydrodynamic contributions to the yaw moment: the total resistance R, and the rudder and keel side forces S. R acts at the leeway angle λ. It acts on the centre of resistance, the distance from the waterline is given as z_y and the distance from the yaw axis is x_y. The yaw moment at an angle of heel ϕ is then:

$$M_3 = R(z_y cos\lambda sin\phi - x_y sin\lambda cos\phi) \tag{2.8}$$

The moment generated by the rudder and keel sideforces, with lever arms x_r and x_k, acts in the opposite direction:

$$M_4 = \frac{1}{2}\rho_w U^2 (C_r A_r x_r + C_k A_k x_k) cos\phi cos\lambda \tag{2.9}$$

The contribution of the sail force to the yaw moment is then given by:

$$M_5 = (x_s F_s + z_0 r sin\phi F_m) cos\lambda \tag{2.10}$$

With x_s as the distance from the centre of effort to the yaw axis.
Finally, the VPP uses the **solution algorithm** with the following equilibrium equations:

$$R = F_m \tag{2.11}$$

$$F_s = S \tag{2.12}$$

$$M_1 = M_2 \tag{2.13}$$

$$M_3 + M_5 = M_4 \tag{2.14}$$

The algorithm is an iterative procedure at each true wind speed and angle that estimates the sailing conditions. The solution algorithm also seeks to find the highest speed on each point of sailing by adjusting the sail trim parameters for optimum performance.

VPP's are mainly optimized to predict achievable speeds and find the optimal points of sail for certain directions. However, accurate simulation models of sailboats must also focus on modelling the dynamic motion effects on the sailboat. Such models for sailing vessels are derived in several studies [25]–[27]. The study by Masuyama et al. specifically concentrates on optimizing the model during tacking maneuvers, recognizing the difficulty to simulate the highly dynamic behavior occurring during tacks.

To gain insight in the fundamentals behind these models, the 4DOF Eulerian equations of motion from the study by Keuning et al. are presented. The corresponding coordinate system is illustrated in Figure 2.10. In these equations, X, Y, K, and N represent the forces and moments in their respective directions. The derivation of these forces involves a series of assumptions and simplifications, employing coefficients derived from empirical data. For instance, N_{sail} term in Keuning's model is approximated by wind tunnel test results, from which the lift and drag coefficients are obtained.

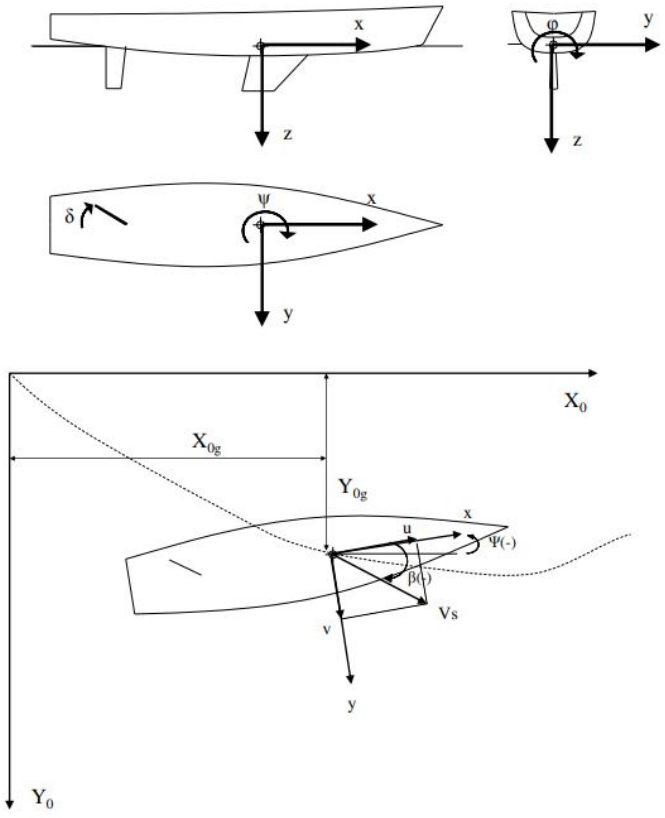


Figure 2.10: Coordinate system used in the mathematical model [25]

The Eulerian equations of motions are then written as follows:

Surge:

$$m(\dot{u} - v\dot{\psi}) = X_U + X_{hull} + X_{rudder} + X_{sail} \tag{2.15}$$

Sway:

$$m(\dot{v} + u\dot{\psi}) = Y_{hull} + Y_{rudder} + Y_{sail} \tag{2.16}$$

Roll:

$$I_{xx}\ddot{\phi} = K_{hull} + K_{rudder} + K_{sail} + K_{stability} \tag{2.17}$$

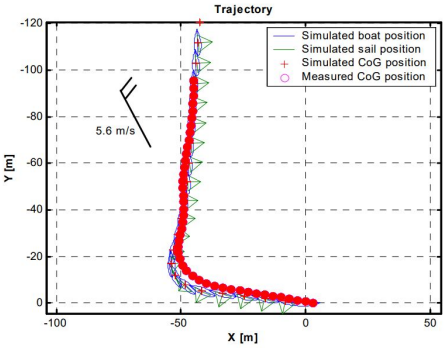
Yaw:

$$J_{zz}\ddot{\psi} = N_{hull} + N_{rudder} + N_{sail} \tag{2.18}$$

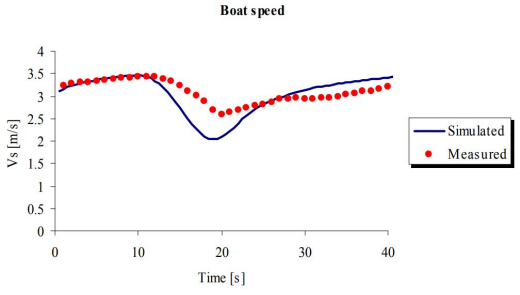
Where:
u = velocity along X axis
v = velocity along Y axis
ϕ = roll (heeling) angle
ψ = yaw angle
I_{xx} = (total) mass moment of inertia in roll
J_{zz} = (total) mass moment of inertia in yaw

For a full representation of all the coefficients, that are either expressed by existing formulations or derivations from data and results that were obtained with the Delft Systematic Yacht Hull, the report by Keuning et al. can be consulted.

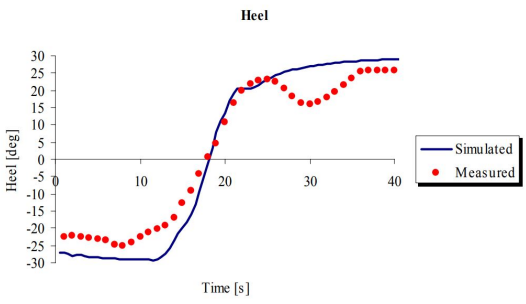
Using such a model to simulate sailing, even though only 4DOF are considerd, yields fairly accurate results as was shown by comparing the simulated data to full scale tests. In Figure 2.11, the trajectory during a tacking maneuver by a ship called 'Checkmate' is shown, as well as the boat speed and heeling angles.



(a) Tacking maneuver



(b) Boat speed during maneuver



(c) Heel angle during maneuver

Figure 2.11: Simulated sailing tacking maneuver compared to full scale tests, from Keuning (2005) [25]

The study of dynamic models for sailboats specifically for control purposes, unlike those for motor-propelled ships, is not extensively explored in research. Some exceptions include the models developed by Buehler et al. [28], Setiawan et al. [29], and Wolniakowski et al. [30]. An important consideration in the development of these models for control purposes is ensuring computational efficiency to facilitate fast feedback. To achieve this, both Setiawan et al. and Wolniakowski et al. simplified their dynamic models to 4DOF, roughly following the structure of the model by Keuning et al. discussed. Buehler et al. manages to successfully incorporate a 6DOF model, including wave effects, but reducing the number of input parameters, such as:

- Change of wind speed and angle over the height of the sail as well as the changing sail shapes are neglected.
- The wave length is assumed to be large compared to the ship length
- Flow separation is estimated with a simple interpolation model.
- Small pitch angle.

With these assumptions, the simulation is able to simulate 120 seconds of sailing in 2 seconds computation time on 2 GHz single CPU computation. The model was not validated on accuracy with empirical data, it was only qualitatively examined for plausibility.

2.2. What are the relevant control features of sailing?

It is evident that modeling a sailing vessel using equations is a complex task. Fortunately, sailors do not require an in-depth understanding of the mathematical sailing model to navigate from point A to point B. Hence, when developing a control system for sailing robots, it is beneficial to understand the decision-making process and actions undertaken by sailors.

2.2.1. What basic knowledge of control methods is necessary to describe sailing as a control problem?

To make the connection between human sailors and autonomous control, a familiar distinction from robotics control can be applied: differentiating between path planning and path following. In control theory these are described as follows:

- **Path planning** refers to the process of determining a feasible and optimal path or trajectory from a starting point to a destination. It involves considering various factors such as obstacles, constraints, and the environment to generate a path that avoids collisions and achieves the desired objective. Path planning is also referred to as *'high-level control'*.
- **Path following**, on the other hand, focuses on executing and tracking the path once it has been planned. The path following controller guides the robot's motion to accurately follow the planned trajectory, adjusting its control inputs (such as velocity, or steering angles). This control process often operates in real-time, using feedback from sensors to continuously update and adjust the robot's motion to stay on the desired path. Path following is also referred to as *'low-level control'*.

Another aspect from control theory that is important to consider for sailing is *control authority*. Control authority refers to the degree of influence that control inputs have on the desired behavior or motion of a system [31]. Sailing can be classified as a problem with very low control authority, also referred to as *underactuated*, as the dynamics of the sailboat are influenced by many environmental factors, including wind, water currents, waves, and other environmental conditions. Sailing even highly depends on these factors for generating speed. A huge limitation that comes with this low control authority, is the fact that a sailboat cannot reach an upwind target in a straight line, making path planning a complex task.

2.2.2. Sailing as a control problem

In most sail races, the first part consists of rounding an upwind buoy. The path planning decisions that need to be made by the skipper consist of determining a zigzag trajectory as shown in Figure 2.12. The skipper needs to take multiple factors into account such as wind direction, current, wind speed, and waves. Based on these, the sailor selects an optimal heading angle and strategically chooses

points along the route to perform a *tacking* maneuver: moving the bow of the sailboat through the wind to change from a starboard to port tack or the other way around. The decision of when to tack is crucial since the the sailboat needs to have sufficient speed since it must pass through the wind. The sailboat needs to be able to maintain enough momentum to carry it through the wind without getting stuck in irons. *Gybing*, another essential maneuver in sailing, involves moving the stern of the sailboat through the wind. Unlike tacking, gybing is typically faster because the sail maintains its forward force throughout the maneuver. The sail swiftly switches sides during the gybe, allowing for a quick transition. The downside of gybing is that the swift transition of the sail can create a dangerous situation. Adding to the difficulty of path planning in sailing is the fact that the factors influencing the chosen path are dynamic and stochastic in nature. As a result, the sailor must consistently reassess and update the chosen trajectories to accommodate the changes. This ongoing adjustment is essential to ensure the sailboat remains on an optimal course and maximizes its progress towards the destination. [32]

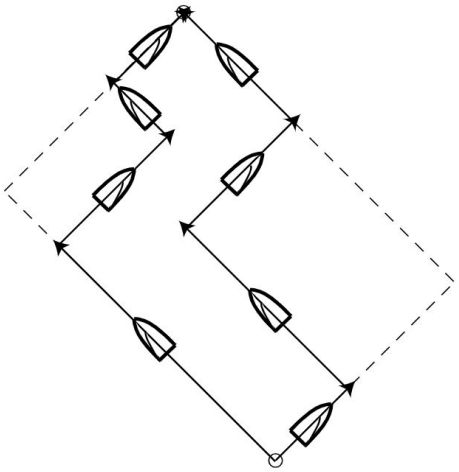


Figure 2.12: Two example upwind trajectories that would take equal time in steady wind [14]

Path following in sailing is done by adjusting the rudder angle, the trim of the sail, and shifting weight. The act of following an upwind zigzag trajectory as described is commonly referred to as *beating* in sailing. Beating involves a series of maneuvers performed by the skipper to steer the sailboat effectively to an upwind point. On the reach, which is the part of the upwind trajectory between tacks, the desired heading angle can be maintained by utilizing the rudder. Additionally, the right trim of sail is necessary to achieve the optimal sail shape, angle of attack, and sail tension, which collectively contribute to propelling the sailboat forward. By managing these factors, the full potential of the sail's lift force is harnessed and the sailboat is propelled with efficiency and speed. When it is time for a tack maneuver, the rudder is used to steer the boat through the wind while allowing the sail to transition from one side (e.g., port) to the other side (e.g., starboard).

2.3. What is the state-of-the-art for control of autonomous sailing control (ASC)?

As a result of the complicated physics and control strategies, replacing these tasks to be carried out autonomously is not a trivial problem. In recent years, the research on ASC has been fairly popular mainly stimulated by the launch of sailing robot competitions such as the Microtransat Challenge and the World Robotic Sailing Competition (WRSC) in 2008. In the Microtransat Challenge, the competitors are tasked with designing a sailing vessel that is able to cross the Atlantic Ocean completely autonomous. The WRSC is a smaller competition, used by many Microtransat challengers as a way to test out their sailing robots. In the WRSC, the 'fleet race' is one of the events. The sailing robots compete against the other to complete a buoy course before their opponents. Competitors have developed many strategies over the years of competition. In Figure 2.13, a diagram of a typical guidance system that is deployed in the competing boats is shown.

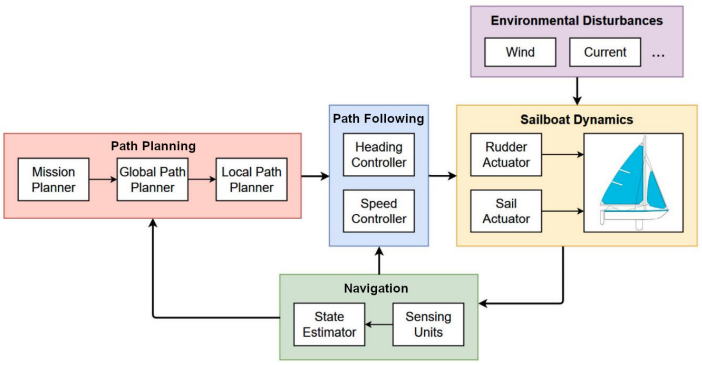


Figure 2.13: Typical Guidance, Navigation, and Control (GNC) system used in autonomous sailboats [33]

In literature, researchers usually focus on specific aspects of this control system, mainly either on the path planning or on the path following modules as discussed before.

The **global path planning** module on a sailing mission is responsible for finding the optimal trajectory (or path) from the starting point to the final point.

The methods found in the literature all address the complexities of maritime navigation, including dynamic environmental factors and the need to balance efficiency, safety, and real-time decision-making in the ever-changing sea conditions. Algorithms like A* [34], Dijkstra [35], and RRT* [36] are based on graph theory and search algorithms, they work by exploring a network of possible paths to find the most efficient route. In *'Minimum Time Sailing Boat Path Algorithm'* [37], Sidoti et al. propose a dynamic programming approach to also systematically explore paths, while saving and reusing subpaths to efficiently determine the optimal route. There's a trade-off between the accuracy and complexity of the methods. Path planning methods like Dijkstra's are simpler but might not always provide the most efficient path in highly dynamic environments. More complex algorithms, such as RRT* or dynamic programming-based methods, offer more precision but at the cost of higher computational demands. Recently, there has been a trend towards machine learning techniques such as Q-Learning [38] and Gaussian process-based Q-learning [39] as they offer adaptability and the ability to learn from experience, without necessarily increasing the computational load by a lot.

Local path planning, as distinguished from global path planning, focuses on adapting the determined path to immediate, dynamic changes in the environment, such as obstacles or shifts in wind direction. This approach is particularly important for short-term sailing maneuvers like tacking and gybing. Local path planning involves generating the optimal desired heading and speed in response to these changes, which is then communicated to a control system. The main two methods found in literature that are used for local path planning are Line-of-Sight (LOS) [39]–[41] and Potential Field (PF) [40], [42]–[44]. LOS is a technique where the path is adjusted to maintain a direct 'line of sight' to the next waypoint or target, navigating around obstacles while keeping this line as straight as possible. Potential Field (PF), on the other hand, treats obstacles and targets as sources of repulsive and attractive forces, respectively. The vessel then navigates through these fields, being 'pushed away' from obstacles and 'pulled towards' targets, creating a path that dynamically adjusts to environmental changes. Plumet et al. [40] explored a combination of the two methods to optimize the local path planning by staying focused on the target while avoiding no-go zones and obstacles using local potentials.

Finally, **path following** in autonomous sailing consists of the heading and speed control. While the boat's dynamics are complex, the rudder and sail angles are typically controlled independently for simplicity. Utilizing the rudder angle for the heading control and the sail angle for speed control as shown in Figure 2.14. The research on path following is the most widespread.

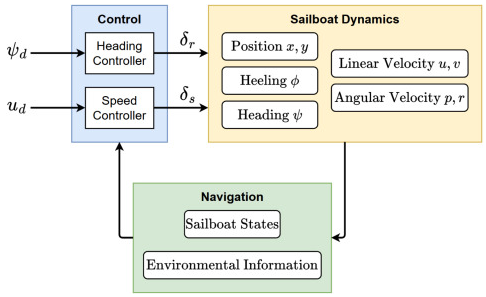


Figure 2.14: Overview of general path following system, with rudder angle δ_r and sail angle δ_s [33]

In terms of the **speed control**, a distinction can be made between offline, online, and AI-based control methods as shown in Figure 2.15.

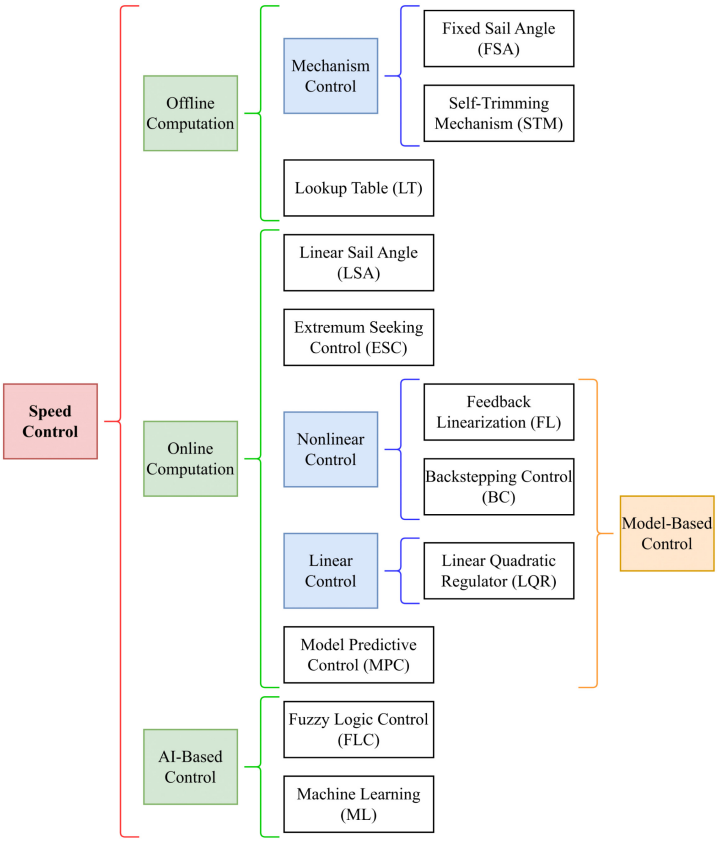


Figure 2.15: Overview of speed control methods found in sailing robots [33]

In practice, offline control methods continue to be favored for their straightforward implementation and proven efficacy. Lookup Tables (LT) are a simple yet effective strategy used in autonomous sailboat control for adjusting the sail angle according to the current apparent wind. These tables are created from either calculated data, experimental results, or polar diagrams. Despite their utility, LTs have the drawback of having to obtain this data beforehand and not being able to optimize the sail setting exactly based on the current conditions. To address this, interpolation methods are often applied in real-time to manage situations where the current state is not covered in the table. Several projects have used a LT for their sail control include the FASt project [45], the Avalon project [34], and the Southampton Sailing Team (SOTON) [46]. With the latter one winning the WRSC two times in a row in 2016 and 2017.

In terms of the **heading control**, various approaches have been developed and utilized as well. For instance, [47] presents a classical PI heading controller based on a second-order linear model, without considering sail control. To adapt to environmental changes, extensions to classic PID controllers have

been explored, such as gain scheduling based on the current situation [16]. More advanced solutions employ nonlinear techniques, with fuzzy controllers [48], [49]. Fuzzy control is popular due to its ability to incorporate qualitative knowledge from experienced sailors without the need for a formal mathematical representation. These model-free solutions demonstrate robust tracking of desired routes, however, fuzzy control for sailing requires numerous rules so the setup process can be quite demanding. Furthermore, Briere [50] concluded that fuzzy controllers lack the full range of desired adaptive capabilities necessary for sailing.

The need for adaptive capabilities is further emphasized by the achievements of SOTON, which were mentioned before. Their performance could be attributed to the implementation of an adaptive probabilistic tack maneuver decision method on their sailing robot *Black Python* [46]. This approach enables the Black Python to adjust its tacking strategies based on previous successes. There are 4 possible tacking strategies, so the sailing robot first needs to explore and test how much time it takes to complete each tack. Based on the obtained data, it determines the most effective tacking strategy and assigns it a higher probability. The success of SOTON in the competition serves as a testament to the significance of adaptive capabilities in achieving optimal performance in sailing robotics.

While the methods outlined have proven its robustness and effectiveness, it is recognized that for an optimal control system the different aspects in the control system have to be working closely together. For example, only considering the sail angle for speed control works sufficiently, however, the rudder usage and the heel angle of the boat can contribute to speed control as well. A study by He et al. has considered controlling the heeling angle of the boat by including a weight to mimic the a sailor on the boat [51]. The same goes for using the sail angle and heeling angle for heading control, instead of just the rudder angle. For this, hybrid methods have been explored. For example, Sun et al. [52] studied the effectiveness of a hybrid control scheme, in which pseudo-spectral (PS) optimal control method is used in heading control, and a model-free framework guided by Extreme Seeking Control (ESC) is used for the sail control. The focus was to optimize the energy consumption of an autonomous sailboat, resulting in notable energy savings of 7% on average.

The achievements of SOTON's Black Python and advancements in hybrid control schemes in autonomous sailboats underscore the potential of Reinforcement Learning (RL) in this area. RL's ability to adapt and optimize strategies in response to the dynamic and unpredictable nature of sailing conditions is a key factor in its effectiveness. This adaptability, coupled with RL's capacity for optimizing performance, makes it particularly suited for enhancing autonomous sailing, showcasing its broader applicability in dynamic and challenging environments. The capabilities and previous conducted research in RL for control will be further discussed in the following chapter,chapter 3.

3

RL for autononomous sailing control

From the previous section, section 2.3, it can be seen that both the high and low-level control benefit from having adaptive capabilities to effectively navigate the dynamic environment. RL offers a potentially unique and essential contribution. RL methods utilize powerful algorithms to search for optimal controllers of systems characterized by unknown or highly uncertain, nonlinear, and possibly stochastic dynamics.

This chapter will introduce the reader to RL and highlight its key advantages and challenges in the context of complex control systems. This knowledge will help to understand the choices made in setting up the RL framework and methods used to develop the RL control policy. The basic concepts and algorithms in RL are mainly based on the works of Sutton and Barto [6], who are known to be early pioneers in the field of RL.

3.1. What is the learning process in RL?

RL is a framework for solving optimal control problems that are typically formalized as a Markov decision process (MDP) [53]. An MDP is used as a mathematical structure to express decision making. In MDPs, feedback from the system is received at each time step in the form of a *state* signal, and an *action* is taken in response. As a result, the decision rule is a state feedback control law, called a *policy*. The action that is taken will change the state of the system, after which the transition will be evaluated by a reward function R (also negative cost ρ in control). In RL, the notation for a state and action at time-step t are s_t and a_t respectively, while in robotics or control these are denoted as x_t and u_t. In this work, the RL notation will be used.

For a problem to be defined as an MDP, it has to satisfy the *Markov Property*. This says that each state is dependent only on its preceding state, the selected action taken and the reward received after the action was executed. So it assumes that the state, denoted with *s*, holds all relevant information. Transitions are generally stochastic, which means that when an action a_t is taken in a state s, the state will change randomly to s_{t+1} depending on the environment, or dynamics, of the system. As a result of the transition, a scalar reward is received according to the reward function to evaluate the immediate effect of action a_t. This process is shown below in Figure 3.1. This cycle will continue until the *terminal state* has been reached, denoted as s_T. The terminal state could either be reached when an agent has completed its task or after a finite number of steps have been completed.

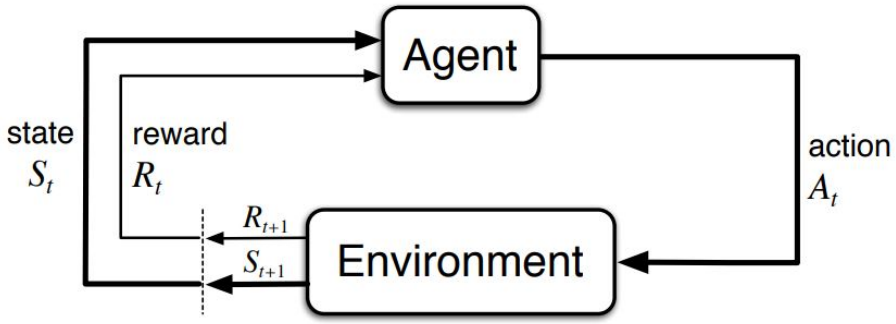


Figure 3.1: The agent-environment interaction in a Markov decision process [6]

The objective of RL is to maximize the (expected) cumulative reward by training an agent (the controller). The agent can explore the environment in the beginning stages to learn the probabilities associated with each action that will lead to the highest reward in the set of states. These probabilities are mapped in the agent's policy $\pi_t(a|s)$. The probability of next state s_{t+1} and reward r given any state s_t and action a_t is expressed as:

$$p(s', r|s, a) \doteq \mathsf{Pr}\{s_{t+1} = s', r_t = r|s_t = s, a_t = a\} \tag{3.1}$$

It follows that the expected reward for the action a performed in state s is denoted as:

$$r(s, a) \doteq \mathbb{E}[r_t|s_t = s, a_t = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in S} p(s', r|s, a) \tag{3.2}$$

With $\mathcal{R}$ being the set of all possible rewards and S the set of all states. The probability of reaching s' given s and a is given by:

$$p(s'|s, a) \doteq \mathsf{Pr}\{s_{t+1} = s'|s_t = s, a_t = a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a) \tag{3.3}$$

Finally, the expected reward of reaching s' when performing a in s is defined as:

$$r(s, a, s') \doteq \mathbb{E}[r_t|s_t = s, a_t = a, s_{t+1} = s'] = \frac{\sum_{r\in\mathcal{R}} r \cdot p(s', r|s, a)}{p(s'|s, a)} \tag{3.4}$$

The objective of RL is reached when the rewards in each state are maximized; thus maximizing the sum of rewards. This sum of rewards is called the *return* G_t. To ensure that the agent is focused more on instant rewards than future ones, it is common to use a *discounted return*. This reduces each reward by a discount factor γ $(0 \leq \gamma \leq 1)$. G_t is then expressed as:

$$G_t \doteq r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^T r_T = \sum_{k=0}^{T} \gamma^k r_{t+k} \tag{3.5}$$

The *value* of a decision policy π is the expected cumulative reward under stochastic transitions from a state s when following π. The value of a policy $v_\pi(s)$ is estimated by the *state value function* v_π, which is defined as:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s\right] \tag{3.6}$$

RL often uses a *state-action value function* $q_\pi(s, a)$ (or *Q-function*) instead of using value functions directly. This is the value of the agent performing an action a in s and is defined as:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t|s_t = s, a_t = a\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s, a_t = a\right] = \mathbb{E}_\pi \left[r_t + \gamma v_\pi(s')\right] \tag{3.7}$$

The reason that Q-functions are useful is that when the optimal Q-function q_* is available, the optimal policy can be computed by selecting the action with the largest optimal Q-value at each state [54]. On the other hand, computing π_* from V_* is difficult since it involves a model. Initially, an RL problem is considered model-free, meaning it doesn't rely on an explicit model of the environment. However, in recent years, there has been an increasing interest in model-based RL methods. These approaches utilize data to learn models of the environment and utilize them partially in the learning process, rather than relying solely on a pre-existing model. This allows them to incorporate the benefits of both model-based and model-free approaches. Model-based methods excel in situations where the dynamics of the environment are well understood and can be accurately modeled. However, model-based RL may struggle when the environment is complex and its dynamics are difficult to capture accurately [55]. Whether to use model-free or model-based methods for control problems serves as a foundation for an interesting discussion, which will be briefly addressed in subsection 3.1.4.

3.1.1. Bellman equations

The *Bellman equations* play a fundamental role in the field of dynamic programming (DP) and RL. It provides a recursive relationship between the value function of a state and the value functions of the successive states [56]. The equation expresses the value function of a state as the expected immediate reward plus the discounted value of the next state which gives an estimation of the value function without explicitly modeling the dynamics of the environment. This allows for iterative methods to estimate their values and by repeatedly applying the Bellman equation, the value function can be updated with new information obtained through interactions with the environment. When reapplying this update rule infinitely many times, v_π will converge. This is called *iterative policy evaluation*. The Bellman equation for the state-value function can be written as follows:

$$v_\pi(s) = \mathbb{E}_\pi \left[r + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) v_\pi(s') \right] \tag{3.8}$$

Similarly, the Bellman equation for the state-action value function can be expressed as:

$$q_\pi(s,a) = \mathbb{E}_\pi \left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \sum_{a' \in A} \pi(a'|s') q_\pi(s',a') \right] \tag{3.9}$$

The *Bellman optimality equations* follow from the Bellman equations to define the optimal value functions. The optimal state-value function $v_*(s)$ is defined as:

$$v_*(s) = \max_{a \in A} \{ r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) v(s') \} \tag{3.10}$$

This equation states that the value of a state under the optimal policy is equal to the maximum expected return that can be achieved by taking the the best action in that state and subsequently following the optimal policy. Similarly, the optimal Q-function $q_*(s,a)$ is expressed as:

$$q_*(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a' \in A} q(s',a') \tag{3.11}$$

The optimal value functions provide the foundation for determining the optimal policy. The optimal policy π_* is a policy that, for each state, selects the action with the highest value according to the optimal state-action value function:

$$\pi_*(a|s) \in \arg\max_a q_*(s,a) \tag{3.12}$$

Solving the Bellman optimality equations allows finding the optimal policy and value functions which will enable the agent to make the best decisions in the given environment. Various algorithms leverage these equations to learn the optimal policy and value functions through iterative updates, this is called *value iteration* or *Q-iteration*. Q-iteration is almost the same classical DP, but it is applied forward in iterations instead of backward in time [54].

Policy iteration alternates between *policy evaluation* and *policy improvement* to find an optimal policy by maximizing the expected cumulative rewards. With policy evaluation, the current policy can

be evaluated but it does not give whether the policy is optimal for the state s. This is what policy improvement does. It compares the value of the current policy π and the q-function of the a potential better policy π' until the found policy is already optimal.

3.1.2. Temporal Difference RL

While DP is effective for policy optimization, it relies on knowing the environment's dynamics. For RL, temporal difference (TD) learning overcomes this limitation. These algorithms use the difference between the estimated value of a state and the value of the next state to update the current state's value estimate. This difference is referred to as the temporal difference error. By reducing this error step-by-step over time, TD algorithms converge to more accurate value estimates.

One of the most well-known tabular TD algorithms is *TD(0)*, which updates the value estimate of a state based on the immediate reward and the estimated value of the next state. The update is performed using the equation:

$$V(s_t) \leftarrow V(s_t) + \alpha \left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right) \tag{3.13}$$

Where $V(s_t)$ and $V(s_{t+1})$ are, respectively, the value estimates of the current and next state, α is the learning rate to determine the step-size of the update and r_{t+1} is the reward received from going from s_t to s_{t+1}

Another example of a highly popular TD learning algorithm is *Q-learning* [57]. It learns the optimal Q-function q_* by iteratively updating Q-values based on temporal differences. It is a tabular method, which means that it works with a table of Q-values where each entry represents the expected cumulative reward for taking a specific action in a given state. It updates the Q-values with an iterative rule based on the observed rewards and state transitions with the *Q-learning update equation* as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right) \tag{3.14}$$

Both Q-learning and TD(0) are only suitable for discrete problems with small state and action spaces. They serve as the foundation for many modern RL algorithms that have been developed to handle larger state and action spaces, including continuous ones. In the following sections, some of these advanced algorithms will be explored to extend beyond the limitations of discrete problems and offer solutions for more complex scenarios such as autonomous sailing control.

3.1.3. What is the learning process in DRL opposed to RL?

As mentioned, a significant drawback of the early algorithms is their limited applicability for problems characterized by large or continuous state and action spaces, as they are tabular methods. This causes them to suffer from the *curse of dimensionality*, which was first observed in DP [56]. This refers to the phenomenon where the complexity and sparsity of data increase exponentially as the number of dimensions and features grows. In TD methods, this is a consequence of the tabular representation of the Q-functions and policies to map states to optimal actions. The curse of dimensionality leads to big challenges such as computational and sample efficiency, as these problems require exponentially larger amounts of data to achieve reliable and meaningful results. Function approximation techniques become essential when dealing with larger or more complex problems like most control problems, as they deal with continuous state and/or actions spaces.

With function approximation, the agent is able to generalize from observed data to estimate values or policies for states and/or actions that have not been visited yet. Depending on the type of algorithm, the policy and/or the q-function is approximated so the problem is reduced to learning a parameterized vector or function.

Some examples of widely used function approximation techniques are policy gradient and actor-critic methods. The main idea behind policy gradient learning is the use of *probabilistic policies* and learn them without needing to estimate the value functions. This can be accomplished with the REINFORCE algorithm [58]. REINFORCE calculates the gradient of the expected return with respect to the parameters of the policy and performs gradient descent to update the policy. The policy then gradually improves its performance by iteratively adjusting its parameters in the direction of higher expected returns. Policy gradient learning can suffer from high variance of the gradient estimates which causes very slow learning. **Actor-critic** methods aim to overcome this issue by combining value functions with

an explicit representation of the policy. The policy (or the ”actor”) learns by using feedback from the value functions (the ”critic”).

However, the combination of function approximation techniques with neural networks has revolutionized the field of RL and continuous control, opening doors to solving complex real-world problems. Neural networks are able to approximate complex, nonlinear functions and with that represent complex patterns and dependencies in high-dimensional state and/or action spaces. When neural networks are implemented into RL problems, it is referred to as deep reinforcement learning (DRL). The framework for DRL is shown in Figure 3.2.

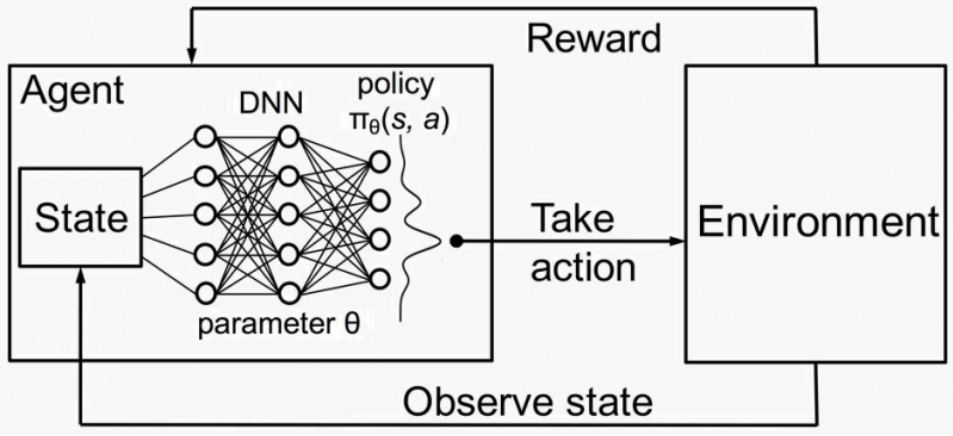


Figure 3.2: The agent-environment interaction in a DRL framework. θ represents the parameters that define the learned policy. [59]

As a result of the successes, DRL gained popularity, and the terms ”Deep Reinforcement Learning” and ”Reinforcement Learning” have started to be used interchangeably. RL now often implies the usage of deep neural networks, given their effectiveness and widespread adoption in the field. However, not all RL algorithms necessarily utilize deep neural networks, as traditional function approximators are still relevant and applicable in certain scenarios.

3.1.4. Model-free vs. model-based RL

In RL, a distinction is made between model-free and model-based RL methods. Until recently, model-based algorithms were generally outperformed by model-free methods in terms of performance. However, in recent years, significant advancements have been made in model-based RL, leading to several notable successes where model-based approaches have surpassed model-free algorithms in terms of sample efficiency, as well as robustness and stability [60]. While model-based reinforcement learning (MBRL) methods offer obvious advantages, they also face challenges. The performance of MBRL heavily depends on the quality and accuracy of the learned model. If the model is biased or inaccurate, it can lead to suboptimal policies. Also, the learned model may overfit to the training data which can result in poor generalization and poor performance in unseen situations. These challenges makes the complexity of implementing MBRL very high, which has put the main focus of RL research on model-free methods since this is more straightforward to new problems. This can explain why promising results in MBRL have only been achieved recently. Opinions on the most effective method for the future of RL are highly diverse. From a control perspective, it is believed that a combination of these approaches will emerge as the most effective solutions for future challenges [61]. In the case of ASC, the added complexity in MBRL is a reason to stick with model-free RL. There has not been any research on ASC with RL methods in the real-world. To showcase the capabilities of an RL approach to problems like this, model-free RL is a sensible starting point.

3.2. What existing RL methods are most suitable to consider for ASC?

To determine the most suitable existing RL methods for ASC, it is essential to consider the unique characteristics of sailing as a control problem, as covered in chapter 2. In brief, sailing involves operating

a sailboat in a stochastic and dynamic environment with limited control authority. Additionally, real-life control problems depend on hardware capabilities, such as computation power, actuator outputs, and sensor information, which introduce slight state and action delays (*latency*) and noisy signals. All these aspects must be taken into account when setting up the method and selecting the appropriate RL algorithm. Fortunately, there has been overwhelming research on applying RL for control in recent years, offering general findings that can guide the process. It will be beneficial to examine these general findings and delve deeper into studies relevant to problems similar to sailing.

Researchers in RL for control tend to take different approaches. It is preferred to perform the training straight in the real environment as this gives the most robust and stable results [55], [62]. However, training in the real world is not always feasible. For example, experiments that require human supervision and are not easily repeatable, training in real life would take way too much time, and thus money. Training in a real environment brings along safety concerns as well. Therefore, an important topic in RL for control methods has been to train the agent in a simulated environment and transferring it to the real world, referred to as *sim-to-real* [11]. This is a crucial aspect of RL research as it is pivotal for addressing real-world problems effectively, it will be elaborated on in chapter 4.

Another challenge in applying RL to control problems - whether the training is done in the real environment right from the beginning or in a simulation first - is to make a careful decision on what RL methods to use and to implement them in the right way. There are over a 100 different RL algorithms due to RL being such an active area of research. Researchers constantly strive to develop more efficient, stable, and sample-efficient RL algorithms. This leads to the evolution and refinement of existing algorithms, as well as new algorithms that address specific limitations or improve upon previous approaches. Variants on RL algorithms may incorporate previously discussed techniques such as experience replay or eligibility traces among many others. These modifications aim to enhance performance, stability, convergence, and/or sample efficiency in specific contexts. The existence of this vast amount of RL algorithms makes that it is imperative to select and tailor RL algorithms to match the requirements of the application that it is going to be used for.

For ASC, there are different ways to approach the problem to determine the specifications. Important aspects to consider are the state and action representation, the choice of RL algorithm and the reward design which will be discussed in the following sections.

3.2.1. State and action representation

For ASC, there are different ways to approach the problem to determine the specifications. One crucial aspect is the state representation that is chosen for the control problem and the available actions for the agent. The state representation refers to how the system perceives and encodes the relevant information about the environment and the vessel itself [63]. By representing the state of the sailboat with a finite number of variables, an approximation of reality is made since in reality this would be infinite. Therefore, it is important to capture enough information for the agent to perform effective path planning and control without making the representation too large and slowing down training.

To determine a fitting state representation, one can leverage past research and its findings. However, the research exploring the application of RL in ASC is limited and solely focuses on the path planning aspect which lowers the number of necessary states. Da Silva Junior [38] proposed Q-learning to generate paths based on the wind direction, while also dealing with potential obstacles. To use the Q-learning algorithm, the state and action spaces need to be discrete, so the the environment is mapped in blocks as an NxN matrix. Suda [64] did use a continuous state space consisting of the three state variables of the sailboat, x, y, and θ, and the two state variables of the stochastic process of the wind, ω and h, creating a five-dimensional continuous state space. The action space was discretized by restricting the actions to only choose the tack to be on (starboard or port side) and assuming a default point of sail for that tack. The objective of the research was aim to optimize the VMG and compare the performance against a state-of-the-art classic control method, model predictive control (MPC). The results obtained are promising, as the DRL was able to beat the MPC almost 90% of the time.

In contrast, in this case the RL agent should consider both the path planning and following aspects of the Optimist, which means that the action space should consist of the low-level controllers and the state should include information about the state of those controllers. The low-level controllers are the standard ones, so the rudder and the sheet of the boat. Another action that is interesting to include is a weight that can mimic the sailor. This type of control was tested by He et al. and showed that only shifting a weight is enough to control the heading of a sailboat in light conditions [51]. This is interesting

Table 3.1: Proposed state and action space for ASC.

Action space ($\mathcal{A}$)	**State space ($\mathcal{S}$)**	**Unit**
Move rudder	Rudder angle (δ_r)	rad
Let sheet out/in	Sheet length (l)	m
Move weight	Weight position (x_w)	m
	Boom angle (δ_s)	rad
	Distance to target (x_t, y_t)	m
	Velocity (u, v)	m/s
	Yaw angle (ψ)	rad
	Yaw rate (r)	rad/s
	Roll angle (ϕ)	rad
	Wind heading (α_{tw})	rad
	Wind speed (v_{tw})	m/s

to include in the action space since it represents sailing more accurately. The state space will then have to include the position and velocities of the rudder, sail and the weight. Finally, the state space should also include information about the target of the agent, this can be included with the position information as 'Distance to target' in x and y direction. An overview of the proposed state and action space is shown in Table 3.1 below.

In this way, the state and action spaces are similar to the inputs and outputs used for state-of-the-art sailing robots. However a big difference between classic control methods and using RL for control is the way the states are utilized. As discussed in section 2.3, in many cases the speed control is based on the sail angle that is received and this is adjusted to control the speed. Same thing goes for the heading control and the rudder angle. The RL agent utilizes a neural network to learn this information from scratch, which means that it has to figure out during training what outputs can be used to change the desired states. The advantage of this is that it there is potential for the agent to figure out how to use a combination of controls to change states in a more effective manner. This represents real sailing more accurately as steering for example is not done solely by moving the rudder, but can also involve moving the weight and sheeting the sail in/out [14].

These actions and state spaces both consist of continuous variables as opposed to previous studies on RL for sailing upwind. Since the state and action spaces differ from previous studies done for RL control in sailing upwind such as Suda and Da Silva Junior, it would be unwise to blindly adopt the RL algorithms used in those studies. Instead, it is essential to consider the unique characteristics of the current problem and carefully select RL algorithms for the task.

3.2.2. Choice, implementation and evaluation of an RL algorithm

The main aspects of ASC that influence the choice of RL algorithm, besides the continuous state and action spaces, are the stochastic and dynamic environment the model operates in, the low control authority and the latency and noise present due to the use of sensors in the real life problem. The need for sample efficiency is high when training in a real environment or a high fidelity simulation is desired, due to the time-consuming nature of repeating episodes. Additionally, for the development of a stable and robust control system, generalization capabilities are essential to effectively handle the uncertainties present in the environment. Lastly, due to the vast amount of RL algorithms that are available all over the internet, it is important to make sure that the RL algorithm to be applied to a complex problem is robust and reliable, and has been extensively tested on a multitude of problems.

The *Stable Baselines* library [65] is a collection of popular RL algorithms implemented in Python. It provides a set of robust and reliable implementations of state-of-the-art RL algorithms. These algorithms are widely used in the research community and industry for training and evaluating RL agents in various environments so it serves as a good basis to choose from. RL Zoo [66], created by the same developers, provides sets of tuned hyperparameters for numerous continuous control problems, facilitating quick and efficient agent deployment and benchmarking. In addition to these tools, RL Glue [6] is another useful framework in this context. It acts as a standardized interface between RL algorithms and environments, ensuring compatibility and ease of integration. RL Glue facilitates seamless ex-

Table 3.2: RL agents used in different continuous control problems

Author & year of publication	Title	RL agent	Reason given	Tested in real-world setting?
Tan et al. (2018) [10]	*Sim-to-Real: Learning Agile Locomotion For Quadruped Robots*	PPO	Stable on-policy method and can be easily parallelized	Yes
Hwangbo et al. (2019) [67]	*Learning agile and dynamic motor skills for legged robots*	TRPO	Has already shown to learn locomotion policies in simulation in [68]	Yes
Havenstrøm et al. (2021) [69]	*Deep Reinforcement Learning Controller for 3D Path Following and Collision Avoidance by Autonomous Underwater Vehicles*	PPO	-	No
Reda et al. (2022) [70]	*Learning to Brachiate via Simplified Model Imitation*	PPO	For its effective utilization of hardware resources in parallelized settings which results in reduced wall-clock learning time	No
Wang et al. (2022) [71]	*Sim-to-Real: Mapless Navigation for USVs Using Deep Reinforcement Learning*	PPO	-	Yes

perimentation with different algorithms and environments without the need for extensive modifications. This standardization is crucial for fair and consistent benchmarking of RL algorithms, making it a good setup for both experimental and practical applications in RL.

The RL glue framework consists of the following components:

- The **environment** receives the observation from the environment and extracts the relevant information to turn into a state to output to the agent, while also providing the reward for that state.
- The **agent** implements the RL algorithm, making decisions based on the information it receives and learning from the outcomes of those decisions.
- the **experiment** controls the length of the training and the starting and stopping of episodes when terminal conditions (specified in the reward function) are reached.

Even though these interfaces facilitate the setup of the RL problem, it is crucial that a careful approach is taken when setting up the RL framework with RL glue and implementing an algorithm from Stable Baselines to a new problem such as this one. This includes decisions like selecting an appropriate RL algorithm for example. Looking at the choice of algorithm made by other researchers that utilize RL for continuous control and underactuated systems might give a good idea. Especially ones that were trained and tested in real life as well. A summary is shown in Table 3.2.

As can be seen, PPO is the most used agent. TRPO is similar to PPO as TRPO was the predecessor for the PPO algorithm. PPO was designed to be simpler to implement and more computationally efficient than TRPO. Both algorithms were designed mainly to address the problem of stability in policy optimization. Traditional policy gradient methods such as REINFORCE can suffer from high variance and poor convergence, especially in complex environments. TRPO and PPO use techniques that limit the deviation between current and the updated policies, preventing large policy updates that might cause instability. So the choice of PPO seems to be a sensible one, also considering that the resulting policies in the shown studies seem to perform well. For example, Tan et al. states: ”While our learned gaits are as fast as the ones carefully tuned by experts, they consume significantly less power (35% and 23% reduction for galloping and trotting respectively).” One of the results from the quantitative evaluation performed by Hwangbo et al. showed that the learned policy was the most accurate in following velocity commands, shown in Figure 3.3.

However, the choice for using PPO is not never very well substantiated and there is no comparison to other algorithms in these papers. The only reasons (if any) that are mentioned are shown in the Table 3.2. While PPO seems to be promising, it is worth looking into why few researchers seem to consider other RL algorithms that are also designed for continuous control problems such as SAC for example. One reason for this could be that TRPO was introduced in 2105 and PPO in 2017, being early algorithms able to deal with continuous action and state spaces, thus being the first choice. Other reasons have been looked into by some researchers who have attempted to compare several agents in continuous control problems (in simulation). The findings are summarized in Table 3.3 below.

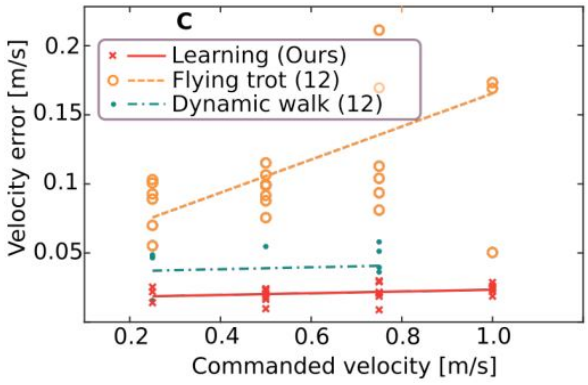


Figure 3.3: Comparison of the learned controller against the best existing controller, in terms of given forward velocity commands of 0.25, 0.5, 0.75, and 1.0 m/s.[67]

Table 3.3: Previous research on benchmarking RL algorithms for continuous control problems. *MuJoCo is a fast physics engine that has several environments that includes models like bipeds, quadrupeds and robot arms [72]

	Task	RL agents considered	Best performing agent
Duan et al. (2016) [73]	several MuJoCo* tasks	TRPO, DDPG, REINFORCE, REPS, TNPG, CEM, CMA-ES, and RWR	TNPG/TRPO
Henderson et al. (2019) [74]	several MuJoCo* tasks	TRPO, DDPG, PPO, and ACKTR	-
Larsen et al. (2021) [75]	USV path following & collision avoidance	PPO, DDPG, SAC, and TD3	PPO

While these studies are useful and highlight some differences in performances of agents on different problems, the main conclusion from Henderson et al. is most notable: there is no clear winner among all benchmark environments. The reason for this is that the performance of an agent is highly sensitive of changes to its hyperparameters, reward scale, batch size, and network structure. This is shown by testing different RL agents (PPO, TRPO, and DDPG) in benchmarked environments and only making simple changes to the policy or value network activations. The results show that they significantly affect performance, see Figure 3.4.

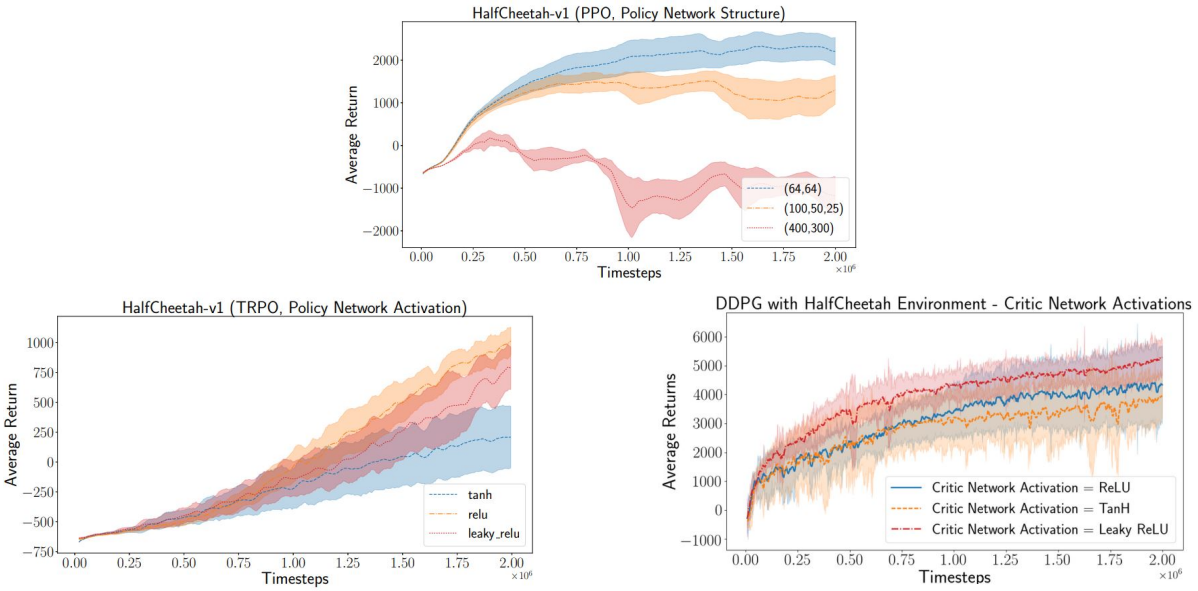


Figure 3.4: Significance of Policy Network Structure and Activation Functions PPO (top), TRPO (bottom left) and DDPG (bottom right). [74]

Consequently, comparing different algorithms is not straightforward. The recommendation given by Henderson et al. is that first of all, attention should be given to reducing this sensitivity and making RL algorithms less susceptible to changes. While RL algorithms are still sensitive, reproducability should be of main concern. This means reporting all hyperparameters, implementation details, experimental setup, and evaluation methods for novel work. Regarding evaluation methods, it was shown by Henderson et al. that stochastic environments makes for a high variance in evaluation results, emphasizing the need for more samples to obtain meaningful conclusions. Particularly in complex environments such as the HalfCheetah in MuJoCo [72]. This was tested by performing 10 experimental trials, for the same hyperparameter configuration, only varying the random seed among 10 trials. The results were split in two sets of 5 which were averaged together. The results are shown in Figure 3.5. It can be seen that the variance between runs is enough to create statistically different results.

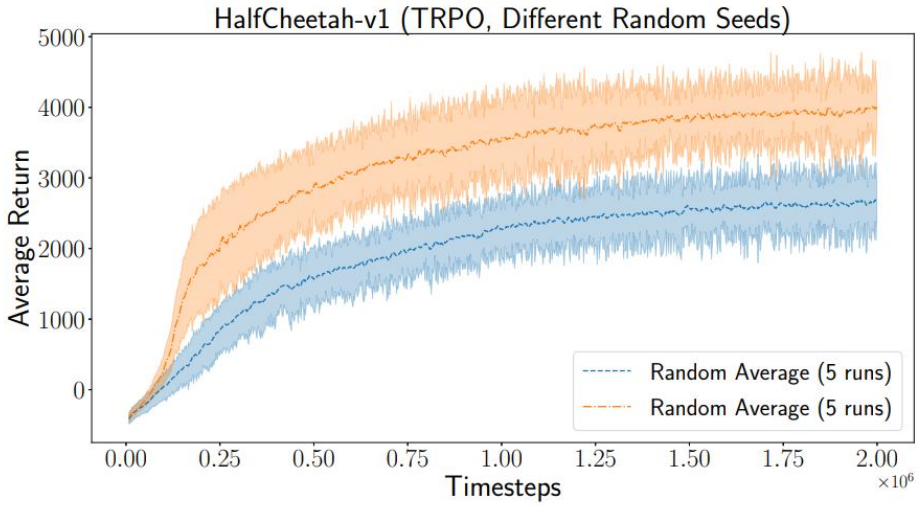


Figure 3.5: TRPO on HalfCheetah-v1 using the same hyperparameter configurations averaged over two sets of 5 different random seeds each. The average 2-sample t-test across entire training distribution resulted in t = −9.0916, p = 0.0016. [74]

According to Henderson et al. it is not possible to recommend a specific number of trials, as it is highly dependent on the problem that is being solved. Instead, a bootstrap power analysis can provide insight into the number of trial runs necessary for statistical significance.

3.2.3. Reward design

An additional finding mentioned in several studies [67], [74], [75] that needs to be considered concerns reward design in the context of this research. The most commonly employed method is manual reward shaping [76], but a major disadvantage is that human bias is being introduced. Sparse rewards that only present the agent with a reward at each episode's termination induce the least amount of bias. However, this sparsity can lead to slow or even non-converging training, particularly in complex environments with demanding exploration requirements.

In contrast, dense rewards provide the agent a reward at each time step in the environment, improving the learning rate. To implement a dense reward function, the designer must have sufficient domain knowledge and/or needs perform extensive testing to shape relevant information in the environment into rewards. [75]

In practice, striking the right balance between dense and sparse rewards depends on the specific problem and the capabilities of the agent. Techniques to overcome the disadvantages of reward sparsity have been developed such as imitation learning, where the agent can learn from an expert policy. Another technique employed by researchers is curriculum learning, where the tasks of the agent are increased in difficulty to increase the probability of task completion. These techniques can also be used to improve the generalization capabilities of the agent which is helpful for sim-to-real transfer. They will be discussed in more detail in chapter 4.

In the case of ASC, inspiration can be gained from Suda et al., as the goal of reaching an upwind target is the same. The reward function that was used is defined follows:

> *For every control step during which the boat has not reached its goal state yet, it is given a reward of −1 units, with the meaning of time cost. The goal is reached when the boat crosses the line from the windward mark directly in windward direction. If the boat crosses the finish line, the episode terminates. If the boat cannot reach the goal within K time steps, for some sufficiently high value of K, the episode terminates and the cumulative reward is set to −K. Thus, essentially, we are treating this problem as episodic RL, and the negative cumulative reward of an episode is equal to the time it would take the boat to reach the goal line. [64]*

This reward function is quite sparse, depending on exploration to reach the goal since the agent is not encouraged to make progress towards the goal in the form of rewards. That is feasible in the case of Suda et al. since the action and observation space are different as described earlier. In our case, the action space is continuous, making it much harder for the agent to reach the target by chance. Therefore, to encourage the agent to make progress towards the target, intermediate rewards will most likely be necessary. Intermediate rewards should not be too large though, as it has been shown that this encourages agents to keep the episode alive to keep accumulating this intermediate reward [12].

Furthermore, safety is of high importance since the control policy should be transferable to the real world. So to keep the boat from crashing into the sides of the basin and to prevent capsizing, termination states should be determined and sufficient penalties should be given when these are reached. Thus, termination should happen when the boat gets too close to the sides of the environment, when a certain roll angle is reached, and in this case, gybing should be penalized as well. While gybing is a common sailing maneuver, it can be dangerous if not executed properly. During a gybe, the boom swiftly changes sides since the wind is coming from behind so the sail is always catching wind. The sudden and large change of boom angle can put excessive stress on the sails and other equipment while also forming a risk of capsizing. [14]

4

Sim-to-real transfer for control applications

One of the primary challenges of RL for control is the difficulty to safely and autonomously gather large volumes of data. To overcome this challenge, simulations are the preferred platform for data collection. Simulations are able to run faster than real-time and multiple runs can be initiated simultaneously, allowing for efficient data collection without the need for constant human supervision. This makes training in the simulation environment faster, more efficient and thus less expensive than on physical robots [55]. Despite convincing results produced by RL research, especially in simulated environments, applying these findings to real-world robot applications has proven to be a huge challenge. This difficulty mainly stems from the modelling errors inherent in simulations, commonly referred to in RL research as the *reality gap* [11], [12].

In subsection 2.1.4, the types of models used to simulate sailing are described. Each of these simulations make certain assumptions and simplifications, which could contribute to the modelling errors. Furthermore, in control systems, the determination of states and the execution of actions is another challenge that is made significantly more difficult in real life compared to simulation. In the real world, the accuracy of determining states is dependent on the quality of the sensors that are used. Furthermore, these sensors will introduce noise to the state representation. As for action execution, it is carried out by actuators which in combination with the sensor suite and layers of software introduce latency into the process due to response times.

To address the challenges of sim-to-real transfer, a part of RL researchers has focused on bridging this *reality gap*. The research on RL for autonomous sailing has not considered this, as the results were exclusively produced using the simulation model [38], [64].

4.1. What methods have been used in similar applications to bridge the reality gap?

In the field of sim-to-real RL for continuous control in complex and dynamic environments, several attempts have been made to bridge the reality gap. One approach is to improve the fidelity of the simulation environment to minimize mismatches between real and simulated environments. This can be done in an analytical or data-driven way, with the latter also known as system identification [7], [72]. However, improving the simulation can be cumbersome, and usually means that the computational efficiency is lowered which is not desirable in RL. Another approach is to accept the imperfections in the simulation and instead aim to make the learned policy robust to variations in system properties. Over recent years, several techniques have been developed and tested to achieve this. Broadly, the methods can be categorized into domain randomization (DR), several types of transfer learning such as imitation learning (IL), curriculum learning (CL) or domain adaptation (DA). Another one is Meta RL (or lifelong learning), which is mentioned as 'the final goal' of several RL researchers. These are algorithms that can 'learn to learn', which could theoretically improve the sim-to-real transfer as well. [11]

While lifelong learning algorithms are an exciting concept, this research will focus on slightly more established techniques, which have already been tested in real-world scenarios. Furthermore, in the problem with the Optimist at hand, optimizing the simulation will be cumbersome, also considering that the RL model has to be set up and trained from scratch. Therefore, the focus will be on methods for transferring knowledge learned in simulation directly towards their deployment in real-world environments. For this to be successful, a control policy needs to be robust and stable to deal with the changes when transferred to the real world environment. To get a better idea of the current techniques and the research that has been performed, a selection of studies on continuous and underactuated control systems is summarized in Table 4.1.

Table 4.1: Previous research on sim-to-real transfer for continuous control.

Author	Title	Method(s)	Reality gap quantified?
Tan et al. (2018) [10]	Sim-to-Real: Learning Agile Locomotion For Quadruped Robots	DR	No
Muratore et al. (2019) [77]	Assessing Transferability from Simulation to Reality for Reinforcement Learning	DR	No
Loquercio et al. (2020) [78]	Deep Drone Racing: From Simulation to Reality With Domain Randomization	DR	No
Wada et al. (2022) [79]	Sim-to-Real Transfer for Fixed-Wing Uncrewed Aerial Vehicle: Pitch Control by High-Fidelity Modelling and Domain Randomization	DR	No
Wang et al. (2022) [71]	Sim-to-Real: Mapless Navigation for USVs Using Deep Reinforcement Learning	DR, CL	No
Kaufmann et al. (2023) [8]	Champion-level drone racing using deep reinforcement learning	DR	No

Domain Randomization (DR) stands out as the most employed method in these studies. DR is a broad term, the way in which it is implemented differs depending on the specific control problem. Throughout the studies mentioned, there are two main types of implementation: dynamics randomization and randomizing the initial conditions of the environment during the training.

Dynamics randomization is employed in studies by Tan et al., Muratore et al., and Wada et al., involves altering the physical properties within the simulation. Tan et al, focused on varying physical parameters for quadruped robots such as friction coefficients, while also adding perturbation forces during the training. Wada et al. adapted this technique for aerial vehicles by specifically targeting the randomization of aerodynamic coefficients.

Muratore et al. applied DR on two tasks: a 2 DoF Ball-Balancer and the linear inverted pendulum, called Cart-Pole. Both the dynamics and the initial conditions were varied in the system. The dynamics randomization was done by varying the physical parameters of both systems during the trainings such as varying the masses and friction coefficients. The initial conditions were varied by changing the starting positions of the Ball-Balancer and the inverted pendulum

Studies like Loquercio et al., Kaufmann et al., and Wang et al. only focused on randomizing initial conditions in the training environment. For Loquercio et al., the drones use visual perception to determine its path, therefore factors such as lighting, viewpoint, and background elements are randomized in the training, as well as the starting positions and the locations of the targets. The latter was also employed by Kaufmann et al. and Wang et al. Kaufmann et al. further enhances this approach by perturbing the states using non-parametric empirical noise models estimated from data collected on the physical system. Meanwhile, Wang et al. combines the initial condition randomization with curriculum

learning, gradually increasing the complexity of the environment.

When looking at some of the results from the studies of Loquerico et al., Kaufmann et al. and Tan et al. in Figure 4.1, Figure 4.2, and Figure 4.3 below. Each of the studies showed that the RL agents were successful in bridging the reality gap, and even beating results of professional pilots in the case of Kauffman for example. Loquerico et al. shows that their drones beat the success rate of a professional pilot, although with slower laptimes. Tan et al. shows that their applied methods improve the sim-to-real transfer over approaches where such methods were not applied.

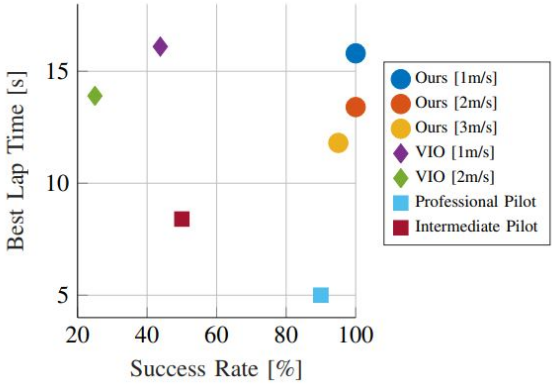


Figure 4.1: Lap times and success rates of the drones completing a track by Loquercio et al. [78]

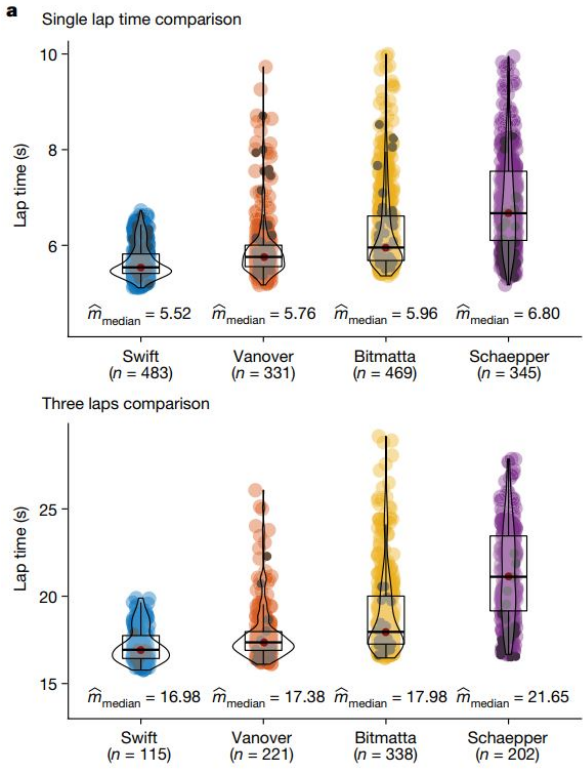


Figure 4.2: Lap times of the drones tested by Kauffman et al., Swift is the RL-controlled drone and the others are operated by professional drone pilots. [8].

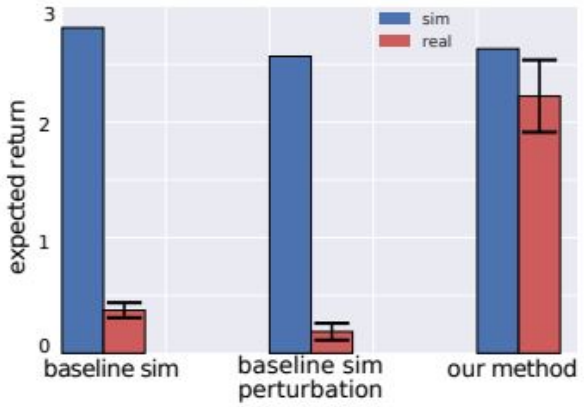


Figure 4.3: Sim-to-real transfer results of differently trained controls for quadruped robots by Tan et al. [10].

Looking at the sim-to-real transfer for the ASC, using DR seems crucial to succeed, as seen in various studies. Also, adding noise to the state, similar to Kaufmann's method, is an interesting approach to attempt to improve ASC's sim-to-real transfer even further. Kaufmann's approach, however, relied on 'system identification'—utilizing actual drone data to learn noise models for signal perturbation. Given the limited real-world data available in this context as of now, employing established control theory techniques for noise addition can be used instead. One commonly used method to model sensor noise is Gaussian noise (or normal noise), which is a type of statistical noise following a normal distribution, which is also known as a Gaussian distribution.[80]

While the results of current sim-to-real transfer are promising, there is one major limitation present in all of the researches in Table 4.1. A critical yet often overlooked aspect is the measurement of this 'reality gap', or modelling errors. This oversight is concerning considering the broader implications of sim-to-real methods. Knowing the size of the reality gap would tells us how much of a gap the sim-to-real methods need to bridge. Thus, it would help placing individual studies within the larger context of sim-to-real transfer research. It can provide a benchmark for comparing different approaches and techniques, allowing for a clearer assessment of their relative effectiveness. This is crucial for the progress of the field as a whole, as it encourages the development of more targeted and effective sim-to-real strategies. For that reason, in this work, an effort will be made to quantify the reality gap.

4.2. How does the performance of an autonomously controlled sailboat change from sim-to-real?

In the context of the ASC problem, the reality gap will depend on the accuracy of the model used to simulate sailing control. To effectively evaluate the robustness of RL control in ASC and its sim-to-real transfer capabilities, it is imperative to have a high-speed simulation environment for training RL agents. Therefore, the used model must be able to simulate control of the Optimist with the states and actions as described in Table 3.1 and its corresponding behavior in a way that is fast to be able to iterate quickly and repeat this for many episodes. In the literature, as touched on in subsection 2.1.4 as well, the focus of models used to evaluate robotic sail control systems lays in the dynamic motion effects, and not as much in optimizing the exact prediction of forward speed, like in advanced VPPs [28]–[30]. Since this project is done in collaboration with MARIN, utilizing their in-house fast time-domain seakeeping and maneuvering software known as XMF (or xSimulation) [81] is a sensible choice.

The XMF framework consists of libraries, where each library adds a different capability to the solver. In each library, several classes or nodes are found. Each node provides a specific functionality and is defined through input properties. Possible outcomes from these nodes are listed as output properties. The set up of an XMF model consists of the body, the environment, and the numerical integrator, which executes the time-domain simulation.

The Optimist model within the XMF framework is a complex structure composed of three primary bodies: the hull, boom, and ballast. These components are linked through various constraints to maintain structural integrity and functionality. The hull, as the largest and heaviest component since it houses all the actuators, serves as the core of the assembly. Attached to the hull via a hinge joint is

the boom, allowing for pivotal movement. The ballast is designed to move laterally, facilitated by a prismatic joint that connects it to the hull. Each of these bodies is defined within its own right-handed Local Coordinate System (LCS), detailed in the Figure 4.4 below.

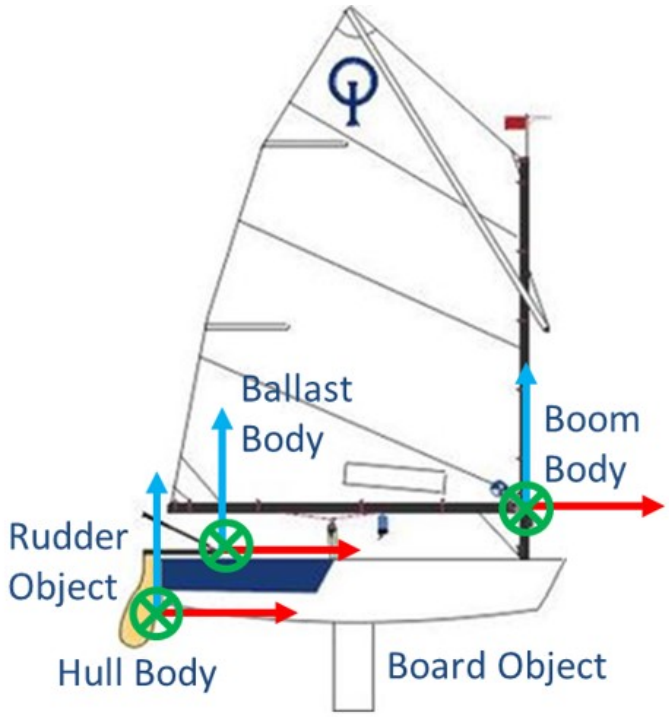


Figure 4.4: Bodies used in the Optimist assembly in XMF and their corresponding coordinate systems.

The origin of the boom is at the intersection between the boom profile and the centerline of the mast. The origin of the ballast is chosen at the center of the aluminum profile, which is the support of the ballast actuator on the hull edges. Unlike the primary bodies, the board and rudder are modeled as 'objects', this means that they have a frame of reference, but no inertia like a body. The mass and moments of inertia of the hull, boom, and ballast are given in Table 4.2 below.

	Mass [kg]	x [m]	y [m]	z [m]	k_{xx}	k_{yy}	k_{zz}
Hull	108.959	1.041	0.000	0.238	0.414	0.744	0.729
Ballast	42.591	0.000	0.000	0.000	0.113	0.153	0.176
Boom	1.651	-0.970	0.000	0.340	1.183	1.438	1.014
Hull-Ballast-Boom	*153.201*	*0.912*	*0.000*	*0.321*	*0.381*	*0.650*	*0.637*

Table 4.2: Body mass/Inertia properties bodies w.r.t. LCS Hull. The values in the last row are obtained by converting the seperate values to a single lumped body.

The other main dimensions and characteristics of the Optimist in the XMF model are given in Table 4.3

	[m]
Length (L_{pp})	2.041
Beam (B)	1.112
Aft draft (T_a)	0.141
Fwd draft (T_f)	0.075

Table 4.3: Main particulars of the Optimist assembly

Implementing the hydrostatic and hydrodynamic forces of the Optimist model were done in a fairly

straightforward way, as the XMF framework already has many existing classes/nodes for this. The hydrostatic forces in XMF were modelled based on the geometry of the hull. The hydrodynamics were implemented in XMF by setting a damping and a maneuvering coefficient. The damping node decomposes the water speeds without current and waves for a number of polygons on the ship's hull at the location of the centroid in two directions (perpendicular & parallel). These speeds are then used to determine the forces and moments. Furthermore, the maneuvering was implemented by setting a coefficient for the drift induced yaw moment. The lift and drag coefficients of the Rudder2020 [82] model were used for the implementation of the rudder, they are shown in Figure 4.5.

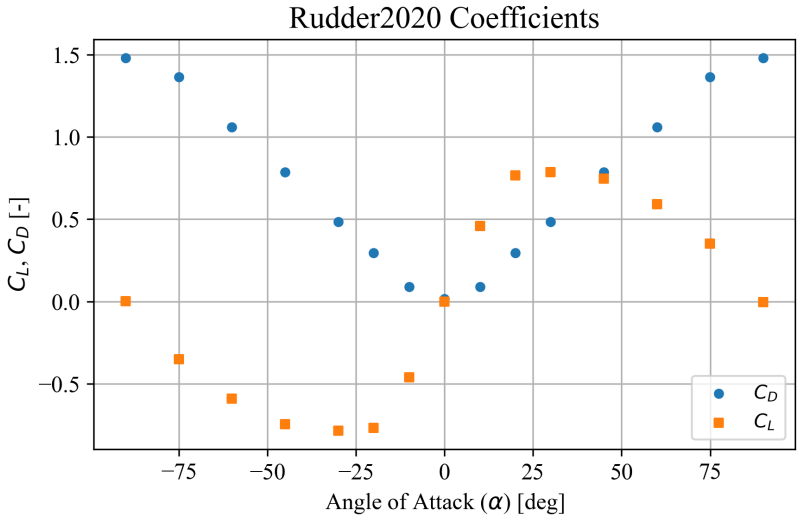


Figure 4.5: Lift and drag coefficients of the Rudder2020 model [82]

XMF has mainly been used to test motor propelled ships, so modelling the aerodynamic forces and the effect on the ships motions has not been done in XMF before. The implementation is described in section 5.3.

When all the forces on the Optimist body are known, the numerical integrator, (2nd order Runge-Kutta) executes the time-domain simulation by calculating the impact of the forces on the vessel's position and orientation over time. It breaks down the simulation into small time steps (set to $\Delta t = 0.01$), applying the forces to predict the vessel's next position and orientation. Over successive iterations, this creates a time-domain simulation of the Optimist, capturing its motions in 6DOF.

To simulate control of the Optimist in the XMF model, it receives action inputs for the rudder, ballast and sheet. The input is scaled to the specified ranges and the actuator rates in XMF are set to simulate the realistic actuator rates from the basin. The ranges and rates that were used in the XMF environment are:

Table 4.4: Ranges and Actuator Rates

	Range	Actuator Rates
Rudder	[-1.222, 1.222] rad	0.200 rad/s
Ballast	[-0.45, 0.45] m	0.200 m/s
Sheet	[0, 1]	0.250 s^{-1}

The XMF model receives input and records the desired output every timestep.

To measure the reality gap and to evaluate the sim-to-real transfer capabilities of the RL agents, a real-world setting is essential for testing. The Offshore Basin at MARIN is a controlled real-world environment that has the capabilities to set the wind level, waves, and/or current. A digital twin of this environment already exists within the XMF framework so the Optimist model can be simulated using this environment. Furthermore, the communication between the simulation and basin environments is already in place, making it an ideal testing environment for this small-scale project. The complete

set-up of the basin with the measurement systems and the real-life Optimist model that were used for testing will be elaborated on in section 5.2.

5

Training, evaluation and testing methods

The methodology section outlines the techniques employed to address the research questions. It begins with a general overview of the basin, XMF, and RL environment setup in section 5.1. A more detailed description of each element within the setup is given in the following sections, starting with the basin set-up in section 5.2 and the XMF model in section 5.3. After that, the RL environment is discussed in section 5.4, elaborating on the training methodologies for various RL agents and the sim-to-real methods employed.

Furthermore, the set-up of the state-of-the-art control method that was used to compare the performance of the RL agents is described in section 5.5. The method used to measure the reality gap is described in section 5.6

section 5.7 discusses the methods used to evaluate the agents in the simulation and section 5.8 outlines the metrics that were used to evaluate their performance and robustness, and to select the agents to test in the basin. Finally, section 5.9 discusses the methods used to evaluate the agents in the basin.

5.1. System overview

An overview of the full set-up is shown in Figure 5.1.

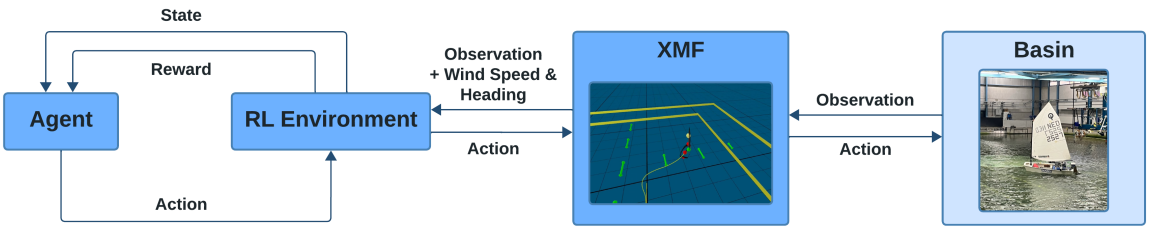


Figure 5.1: General overview of the set-up

The set-up allows for a smooth connection between the sailing simulation (XMF model, elaborated on in section 5.3) and the basin environments, using the XMF model as a way to communicate with the RL environment. The XMF environment is always connected to the RL environment, the connection to the basin can be added when desired. When the basin is in-the-loop, the measurements from the basin are sent to the XMF model, which calculates the wind speed and heading based on the position and heading angle of the Optimist. This is done because there are no sensors to measure the wind heading and wind speed in real-time by the Optimist. Instead, the spatial wind data that was obtained with steady wind field measurements in the basin is used. These measurements were carried out at

wind levels of 40% and 60% at a height of z=1.5m on a grid of 56 points in the xy-plane. The resulting wind field data is used to send the current wind speed and heading to the Optimist based on its position and heading angle in the basin. The RL environment then translates the input from XMF to the input or 'state' that is sent to the agent. The RL environment also calculates the rewards associated with that state using the reward function. The states and reward function used to train the agents are elaborated on in section 5.4. The agent outputs an action command to the RL environment consisting of the rudder action between [-1, 1], the sheet length [0, 1] and ballast position [0, 1]. The XMF environment scales the output and calculates the corresponding actuator outputs based on the specified actuator rates. If the basin is in-the-loop, these outputs are sent from XMF to the basin. In that case, the rates are already taken into consideration by the actions that XMF outputs, but the rate limits of the actuators in the basin are still set to ensure safety.

A more detailed description of the XMF model and the systems used in the basin is given in the following sections, starting with the basin set-up.

5.2. Basin set-up

The robotized Optimist that is employed in the basin is shown in Figure 5.2.

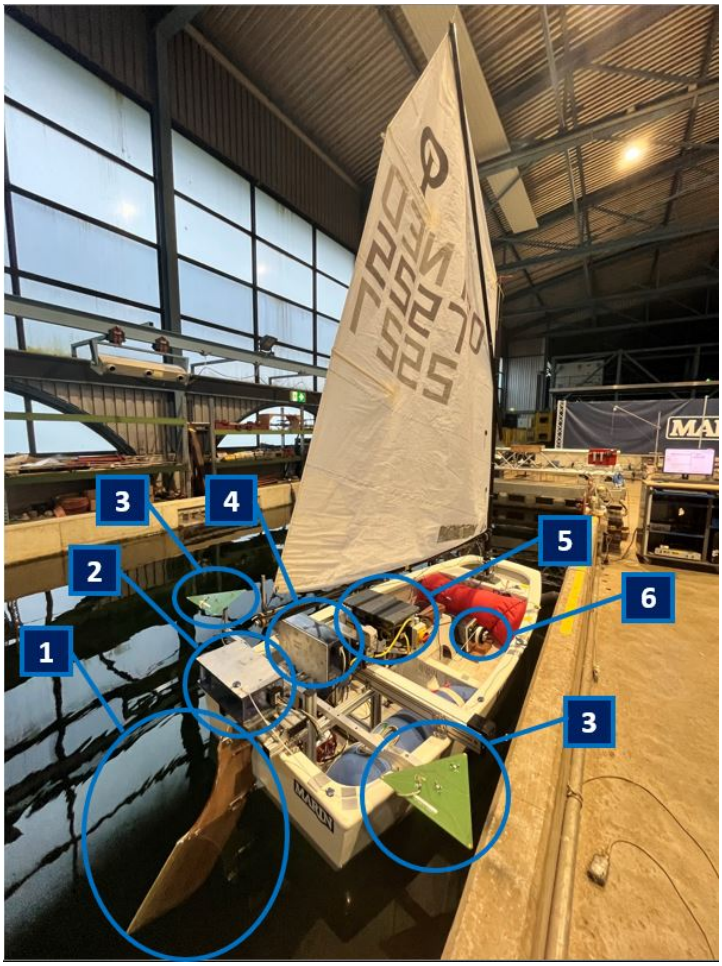


Figure 5.2: Robotized Optimist for autonomous sailing control, with: 1. Rudder 2. Servo motor to control the rudder angle 3. Triangles with a LED on each corner 4. Servo motor to move the ballast 5. Ballast/battery 6. Winch for sail control

The 3 available actions are moving the rudder (1), letting the sail in/out with the sheet, and moving the ballast (5) over the y-axis. Each of the actions have their own actuator on board, a servo motor for the rudder (2) and servo motor for the ballast (4). The sail setting is controlled using a winch (6). The triangles (3) contain 3 LED's each, besides those 6 LED's, there are 2 more positioned on the starboard side of the Optimist. This is done because the Optimist was operated in the basin and followed by a

carriage located on the starboard side when the Optimist is sailing upwind. The carriage features the Optotrak Certus HD : *'a Dynamic Measuring Machine for 3D and 6DOF motion tracking and analysis applications*' [83], which is able to track the positions and motions of infrared LED's (markers) on the Optimist in real-time at a high speed of approximately 495 frames per second (fr/s). Using at least 5 of the 8 LED's, the position and motions of the Optimist can be captured in real-time. The carriage also carries a wireless transmitter/receiver that can send and receive the data in real-time. The carriage following the Optimist is shown in Figure 5.3

Figure 5.3: The carriage following the Optimist„ containing the Optotrak Certus HD and communication hardware.

The basin environment is a controlled environment, meaning that the wind level, waves and current can be set to the desired levels. In the tests that were done, the current and wave levels were set to 0. The wind level is set to either 40% or 60%, since the data of these wind fields is available as described in section 5.1. The wind is generated by a wall of wind fans located on the north side of the basin, see Figure 5.4.

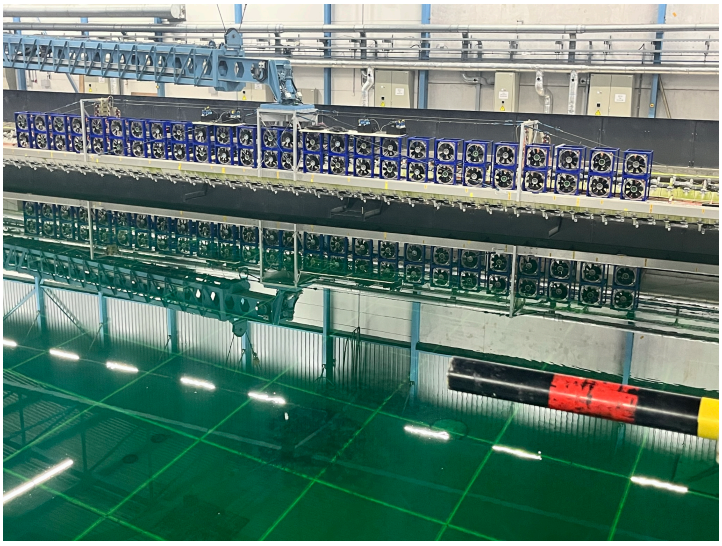


Figure 5.4: Wall of wind fans

A top view of the basin with the size of the inner boundaries is shown in Figure 5.5. A rope is placed at the inner boundaries to prevent the Optimist from hitting the sides of the basin.

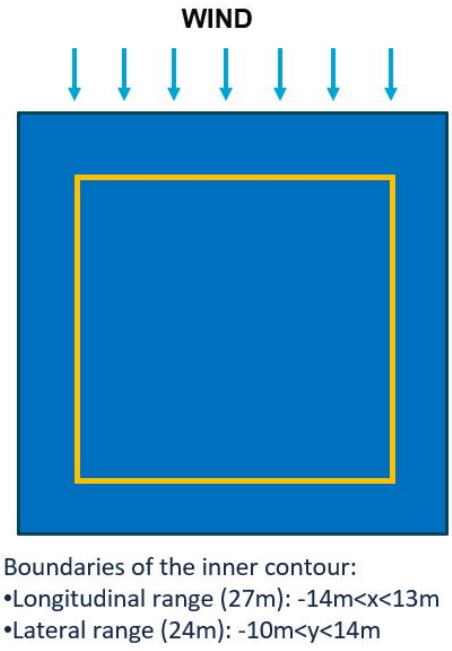


Figure 5.5: Top view and size of the inner boundaries of the basin

5.3. XMF model

The numerical model of the Optimist is made in MARIN's Extensible Modelling Framework (XMF) or xSimulation, as described in section 4.2. The set up of the complete XMF model of the Optimist is as follows:

1. **Environment:** A digital twin of the Offshore Basin.
2. **The Optimist Body:** Comprised of three separate bodies, each with its own mass and moment of inertia.
3. **Numerical Integrator:** Executes the time domain simulation using a second-order Runge-Kutta method (rk2).

Simulating the aerodynamic forces and the effect on the ships motions was not done in XMF before. This was implemented using lift and drag coefficients from Figure 2.9. The driving and side forces acting on the sail are then calculated using Equation 2.4. The leeway angle (λ) is assumed to be neglectably small. The application position, or centre of effort, of the sail forces is assumed to be in the middle of the sail area of the Optimist. The angle of attack taken as the difference between the apparent wind direction and the measured boom angle. The calculated aerodynamic forces and their application positions are then used in combination with the hydrostatic and hydrodynamic forces to solve the equations of motion in 6DOF. A snapshot of the Optimist sailing in the XMF environment is shown in Figure 5.6.

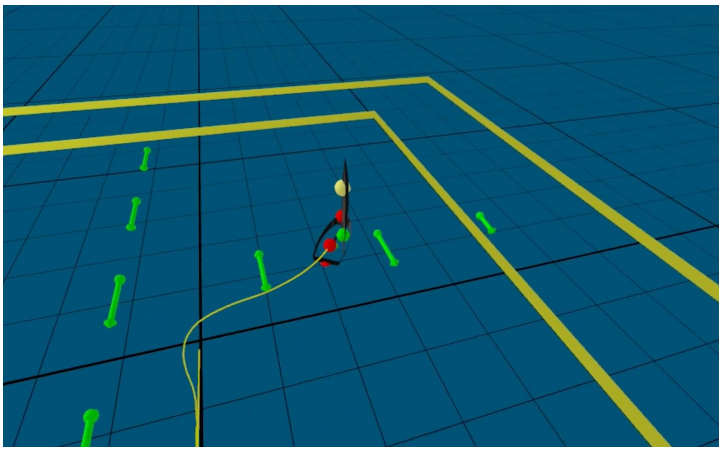


Figure 5.6: Optimist sailing in XMF environment

When XMF is running in connection with the RL environment, the necessary information is extracted from the XMF output and used as the 'state' to send to the agent, while the reward function calculates the reward based on that state. This is elaborated on in the next section.

5.4. RL environment & training set-up

The RL framework is based on the standardized RL Glue set-up, implementing the Stable Baselines3 library providing access to well established RL algorithms, as covered in subsection 3.2.2. The framework consists of the **environment**, the **agent**, and the **experiment**.

The **environment** is the XMF environment, an observation is made at every timestep of $\Delta t = 0.01s$. The environment extracts the relevant information to output as the state to the agent, while also calculating the reward for that state. The reward function used was defined as follows:

Reward Function:

- **Distance Reward:** $2 \times \frac{1}{|x_{ship} - x_{target}| + 1}$
- **Distance Penalty:** $-2 \times (x_{target} - x_{ship})$
- **Gybe Penalty:** -100
- **Actuator Efficiency Penalty:**

$$-5 \times \text{Rudder Rate}$$
$$-5 \times \text{Ballast Rate}$$
$$-5 \times \text{Sheet Rate}$$

Terminal Conditions:

- **Out of Time ($t > 360s$):** -1000
- **Out of Boundaries (see Figure 5.5):** -3000

- **Capsize Risk ($\phi > \frac{\pi}{5}$):** -3000
- **Reaching Target (x > 10):** $+2000$

To reduce the computational load, a *'skipped steps'* parameter was implemented and set to 49. This means that for every 50 steps ($= 0.5s$) the environment takes, the agent only receives a state and sends an action once, effectively making the agent operate at a higher time scale. The action and state space that were used to train the agents are shown in Table 3.1. All the components in the state space were normalized to range between [-1, 1]. The actions that the policy outputs are also all in the range [-1, 1], which were then scaled to each individual action range that was specified in the XMF environment (see Table 4.4).

The **agent** deployed across all experiments was the model-free, off-policy Soft Actor-Critic (SAC) algorithm. The hyperparameters were adopted from the collection available in RL Zoo. The SAC agent's architecture comprises two neural networks: the policy network (π) for the actor and the Q-function network (qf) for the critic. Both networks were structured with two hidden layers, each consisting of 64 units with the Tanh activation function. The agent utilized an 'MlpPolicy', which refers to a multi-layer perceptron policy. The complete setup and detailed specifications can be found on a dedicated webpage, the link and contents of the website are elaborated on in Appendix A: https://www.marin.nl/en/research/artificial-intelligence-applications.

The **experiment** controls the length of the training and the starting and stopping of episodes when terminal conditions (specified in the reward function) are reached. The experiment part of the framework saves the policy of the agent every 500 timesteps to be used for evaluation.

The use of this set up was validated by running four trainings of an agent with a simple environment set up, starting from a single initial condition:

- **Wind Setting:** 60%
- **X:** -10 m
- **Y:** 0 m
- **Yaw:** 0.5236 rad
- **Forward Speed:** 0.3 m/s

The length of the trainings was 400,000 timesteps. The agents were trained using a single compute node. This node featured dual Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz 10-core processors, 64 GB of RAM, and a 500 GB local disk, connected through Gigabit Ethernet and Infiniband HBA.

A summary of the results of the trainings is given in Table 5.1.

Test Name	Total Runtime	Episodes	Avg Runtime/Episode (s)	Success Rate
SAC_test_1	22h 46m	5083	$\approx$ 16.1	97.0%
SAC_test_2	23h 7m	5317	$\approx$ 15.6	97.6%
SAC_test_3	22h 53m	5166	$\approx$ 15.9	98.2%
SAC_test_4	20h 17m	4008	$\approx$ 18.1	95.8%

Table 5.1: Summary of SAC validation training results. The success rate column depicts the percentage of times that the agent has completed the task of reaching the target during the training.

The learning curves of all 4 agents are shown in Figure 5.7 .

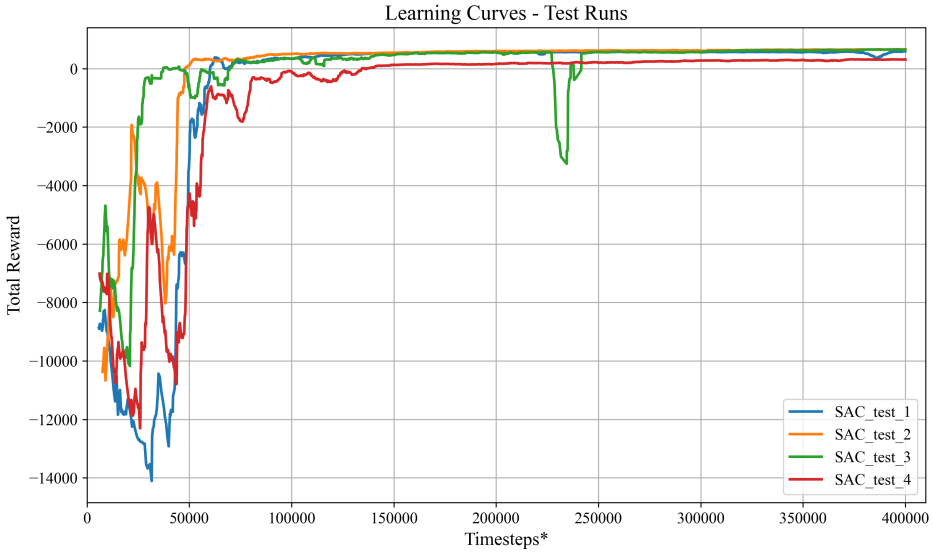


Figure 5.7: Total reward obtained over the timesteps during the test runs. *In order to present the learning progress of the reinforcement learning agents more clearly, a moving average of 50 episodes was applied to the total reward data.

The agents perform quite consistently in the trainings. Out of the 4 trainings, SAC_test_4, seems to learn a suboptimal policy with the lowest resulting rewards. To take a look at the differences in the learned policy between the agents, an outtake of trajectories taken by the agent in 6 episodes throughout the training of SAC_test_2 and SAC_test_4 are shown in Figure 5.8 and Figure 5.9 respectively.

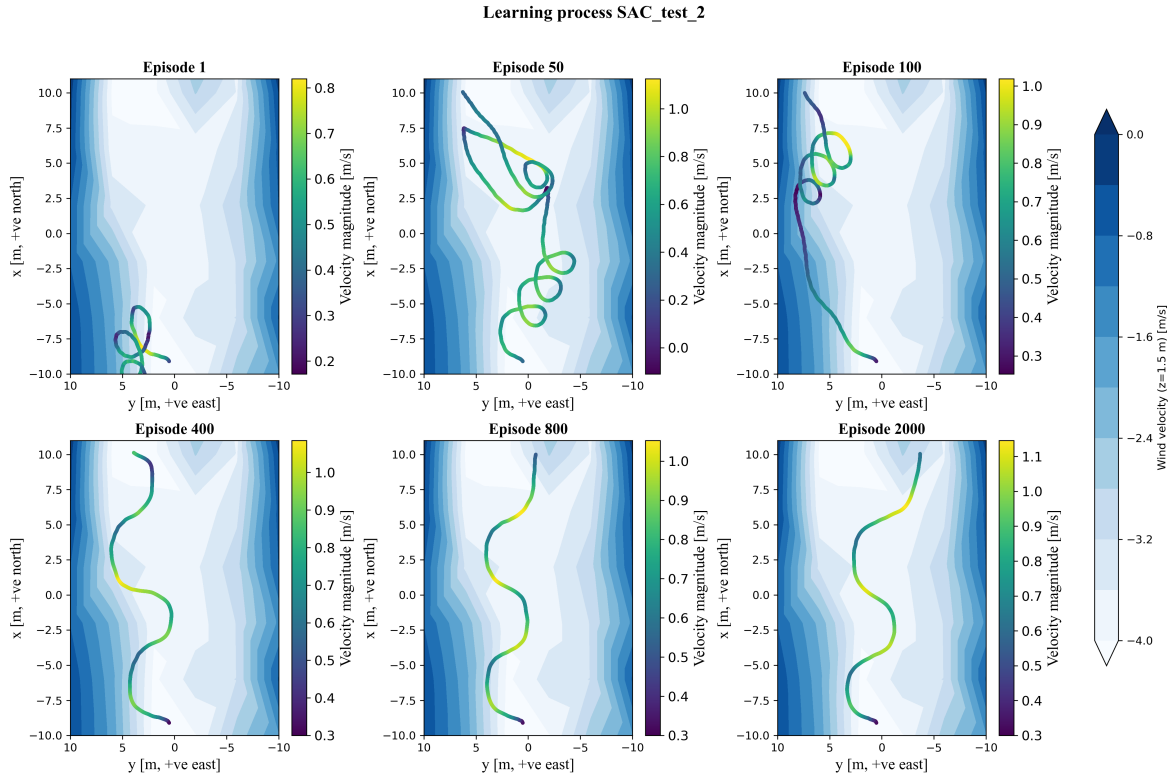


Figure 5.8: Learning progress SAC_test_2.

Looking at the learning process of SAC_test_2 in Figure 5.8, the agent learns an effective policy after about 400 episodes already. It explores different strategies like gybing to gain speed and/or using

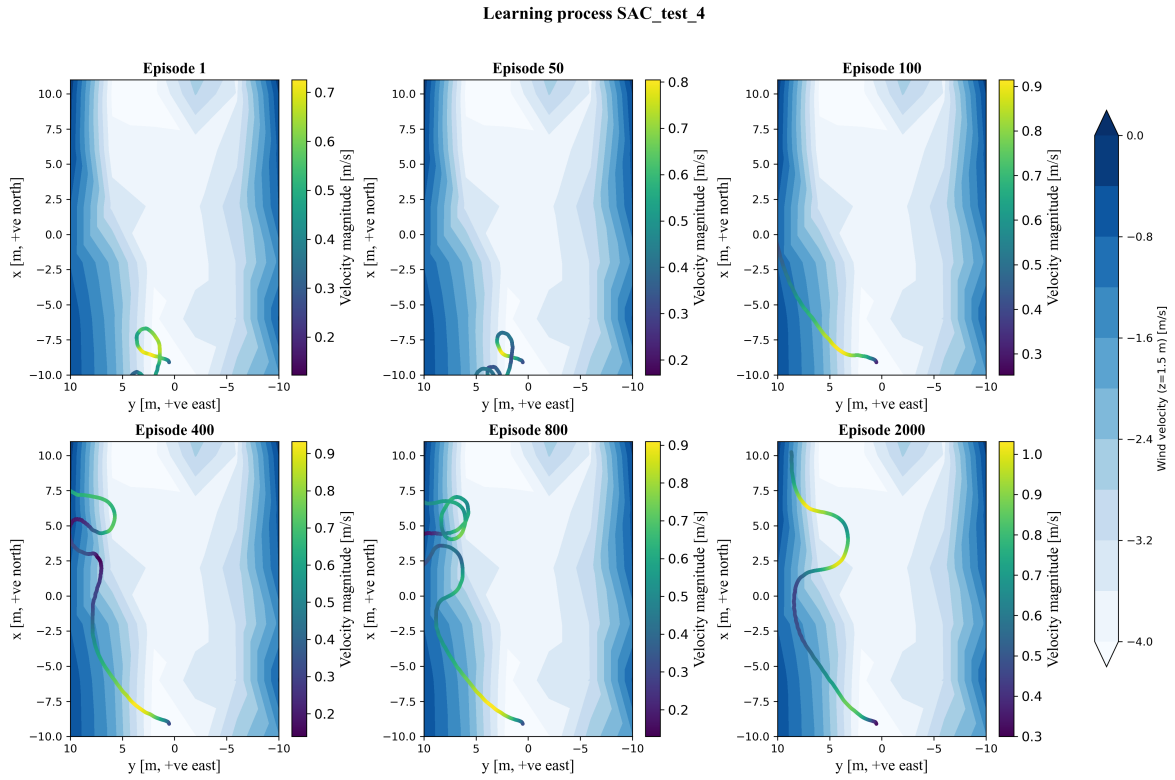


Figure 5.9: Learning process SAC_test_4

the sides of the basin where the wind speed is lower. Finally it learns to tack to go upwind, called 'beating' in sailing. This strategy results in the highest amount of reward at the end of the episode since the target is obtained the fastest (since the discounted reward parameter is set to $\gamma = 0.998$). The sub-optimal rewards of SAC_test_4 shown in Figure 5.7 can be explained by looking at the policy that it learned in Figure 5.9. It seems to utilize the lower wind speed areas on the side of the basin to sail upwind in a more direct manner instead of using tacks. When looking at the rudder angles throughout episode 2000 of both agents in Figure 5.10, it can be seen that while both agents are quite erratic with their rudder action. SAC_test_4 seems to have learned how to *scull* to propel itself into the wind. This can be seen mainly in the first 20 seconds of the episode where the rudder is moved quickly from side to side to it is basically using the rudder as a fin, which is considered 'cheating' in sail races, while also not being energy efficient. In section 5.8, an evaluation metric is described to catch this type of behavior.

After these validation runs, it is clear that the RL set up allows the agents to learn a policy that knows how to sail upwind in the XMF environment. Due to the randomness in the agent's exploration process, the learned policies can be quite different between agents that were trained using the same set up. This was considered when training the different types of agents to go in the basin. Trainings with the same set up were started at least four times to be able to evaluate and compare their learned policies. To do this, it is necessary to look at more than just the learning curves and trajectories over the episodes. Evaluation metrics are used to compare the agents and judge their performance and/or robustness, these are described in section 5.8.

Starting from the described set up, different types of agents were trained:

1. *SAC regular (SAC or SAC_REG)*
2. *SAC trained with Domain Randomization (SAC_RI)*
3. *SAC trained with Domain Randomization + Observation Noise (SAC_ON)*

Regular - The regular SAC agent will be trained with basically the same set up as the test runs, only the starting position (x,y) is changed to be the following:

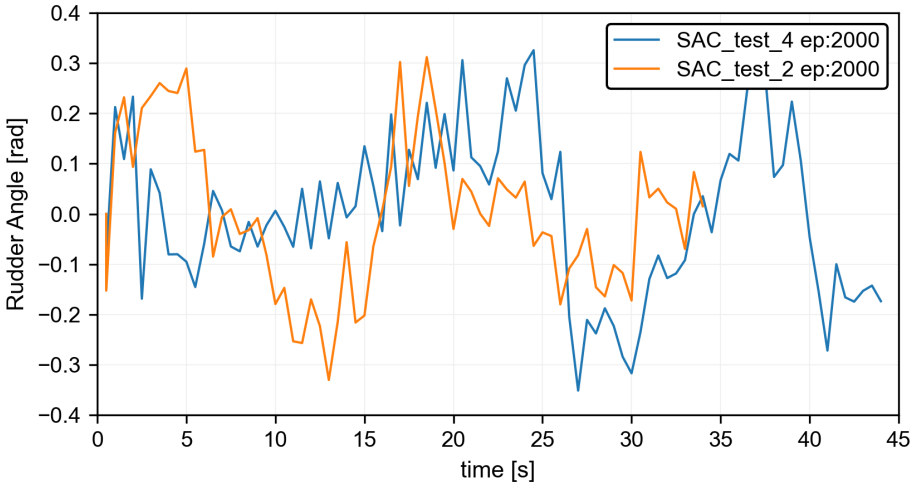


Figure 5.10: Rudder Angles of SAC_test_2 and SAC_test_4 during episode 2000

- **Wind Setting:** 60%
- **X:** -8.37m
- **Y:** -6.13m
- **Yaw:** 0.5236 rad
- **Forward Speed:** 0.3 m/s

Domain Randomization - The domain randomization is achieved by randomly varying the initial conditions of the environment during the training of the agent. The range of initial conditions that was utilized is as follows:

- **Wind Setting:** [40%, 60%]
- **X:** [-10.0, -6.0]m
- **Y:** [-8.0, 12.0]m
- **Yaw:** [-0.8, 0.8]rad
- **Forward Speed:** [0, 0.6]m/s

Observation Noise - Another technique used to improve the sim-to-real transfer was to implement basic noise models on the observed signals. Specifically, noise was introduced to the signals that the RL environment received from the XMF environment. This modification was done before extracting the state information to be sent to the agent. Consequently, the state is derived from these 'noisy' measurements, thereby attempting to approximate the real-world sensor noise. This approach aims to enhance the realism of the simulation, ensuring that the agent learns to operate effectively in the presence of the noise encountered in real-world scenarios. Two basic models to add the noise to the observed signals were used:

- *Proportional Gaussian Noise:* The first method introduces noise that is proportional to the magnitude of the original value. This approach is particularly useful for modeling scenarios where the uncertainty or variability in the measurement increases with the signal's intensity. For instance, sensors that are more erratic at higher readings might be well-represented by this model. The standard deviation of the noise is a fraction of the absolute value of the original signal, controlled by the percentage of noise. This type of noise was added to the following signals:
  - Forward speed
  - Rudder Angle
  - Ballast Position
  - Sheet Length

  - Boom Angle

- *Uniform Gaussian Noise:* The second method applies a uniform level of Gaussian distributed noise across the entire range of the signal. This is based on the range of the original signal. It represents scenarios where the sensor noise remains consistent regardless of the signal's intensity. This type of noise was added to the following signals:

  - X,Y position
  - Roll
  - Yaw
  - Wind Heading
  - Wind Speed

Both methods utilize a factor of 2 in the calculation of standard deviation to ensure that the noise value falls within the 95% confidence interval, assuming a normal distribution. A range of these noise models was incorporated when training the agents to increase the probability of modelling the reality more closely. The range included noise levels of: 1%, 4%, 10%, and 20%. The code to add the different levels of noise and the resulting noise signals can be found on the WEBSITE

The agents trained with the added noise were also trained with the randomized initial conditions. The resulting agents are evaluated in the simulation and put through a selection process to select the best performing agents to be tested in the basin. This process is described in detail in section 5.7.

5.5. State-of-the-art control

In order to assess the performance of the RL-controlled Optimist in comparison to the current state-of-the-art, the control system developed by the Southampton Sailing Robot team (SOTON) was integrated into the XMF-basin framework. SOTON developed a one meter long autonomous sailing robot with the goal to win the WRSC, which consists of four events: station keeping, cooperative area scanning, obstacle avoidance and the triangle race. This code is open-source, however, it was developed in the ROS framework, so this had to be adapted for application in the XMF-basin setup. Not all features were necessary to be modified to comply with the XMF-basin framework as there was one task to perform instead of four: sailing to an upwind target. Therefore, the necessary control features of SOTON were extracted and integrated with the XMF-basin framework. It will be referred to as 'SOTON' in the remaining work. The high-level control system is comprised of two key components: tack decision and goal heading. In the context of this system, the **tack decision** has been modified to be based on 'tacking zones,' which are placed 6 meters from each side of the basin, giving the Optimist time to execute the tacking maneuver. A tack command is triggered when the Optimist is sailing with its sail on the portside and enters the left tacking zone, and vice versa. Unlike the original SOTON code, which featured 4 tacking maneuvers with the optimal one determined through trial and error, the adapted approach in this case only features a basic tack maneuver. This was done because there are no waves in the basin, unlike in the rougher environments in which the one-meter-long sailing robot had to perform. This eliminates the need for additional tack maneuvers, as the Optimist will not encounter the same challenges. The **goal heading** calculation was taken from the original SOTON code, where the goal heading was calculated based on the relative angle to the target that was set. When the target is upwind, the goal heading is set to the 'beating angle' which SOTON put at $45°$.

The low-level control control consists of the sail and rudder control, which, besides making it compatible with the XMF environment, did not need any additional adaptations. The **sail setting** is controlled with a look-up table of the apparent wind direction (AWD). The values used are taken from tests that SOTON did with a Laser dinghy, which is similar in size to an Optimist. The sail settings are shown in Table 5.2.

The **rudder** utilizes PID control, which takes the error in heading to goal heading as input and outputs the rudder setting every 0.1 seconds. The tuning of the PID controller was done in the simulation, by running episodes with the following initial conditions:

- **Wind Setting:** (40%, 60%)
- **X:** -10 m
- **Y:** -8 m
- **Yaw:** 0.5236 rad

Table 5.2: Sail settings for various heading angles

AWD [deg]	Sail setting [0,1]
0	0
36	0
40	0.0519
53	0.0889
85	0.2222
105	0.4444
160	0.7555
178	1
180	1

- **Forward Speed:** 0.3 m/s

So the Optimist would run for 2 episodes in both the 40 and 60% wind setting from a single starting position. The first step of the tuning procedure was to find the optimal proportional (P) value by setting both the integral (I) and derivative (D) values to be 0. At the start, a low P-value of 0.1 was chosen and this was increased with increments of 0.1 until the rudder angle signal becomes unstable. This happened at a P-value of 0.4. At a P-value of 0.35, the Optimist showcased the most optimal behavior in both the 40% and 60% wind field. After this, small values for the I and D settings were tried to see if it would improve the performance further, but they made the Optimist less stable. Therefore, the PID-values employed for SOTON are P=0.35, I=0, D=0. To validate the expected functionality of SOTON, the resulting trajectory was compared to the data available in *'Adaptive Probabilistic Tack Manoeuvre Decision for Sailing Vessels'* by Lemaire et al. [46]. The overall behavior when sailing upwind looks to behave in a similar way. To assess the performance and robustness further, the same evaluation process and metrics employed for the RL agents was used, which are described in section 5.7, section 5.8, and section 5.9.

5.6. Measuring the reality gap

To quantify the modeling error in the simulation with respect to the reality in the basin, 5 runs were performed in the basin using open-loop controls, with the same deterministic rudder and sheeting actions.

The measured data of the basin runs was in turn used to replicate the runs in the simulation, using the measured initial conditions. As a result, the trajectories of the Optimist in the basin and in the simulation turned out as in Figure 5.11 below. The simulated run is showing the times of the tacking maneuver.

To verify that the same control actions were performed in both runs, the deterministic rudder and sheeting actions are shown in Figure 5.12, as well as the measured actions from 1 of the 5 runs in basin. However, since the actions are also part of the simulation, they are subject to some amount of modelling error as well. To gain an understanding of the extent of the differences between the measured actions in the simulation and the basin, the latency, RMSE and NRMSE of the actions are calculated.

The latency in the basin for the rudder angle is 2 timesteps (=0.2s), and 3 timesteps (=0.3s) for the sheet length. To compute the remaining error in the measured actions, the basin and sheet length signals are shifted to overlay the simulation signal as shown in Figure 5.13.

To get an idea of the size of the difference between measured actions, the shifted signals are used to calculate the Root Mean Squared Error (RMSE) and the Normalized Root Mean Squared Error (NRMSE) for the rudder angle and sheet length with the following expressions:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_{\text{basin},i} - X_{\text{simulated},i})^2} \tag{5.1}$$

$$\text{NRMSE} = \frac{\text{RMSE}}{\text{max} - \text{min}} \times 100\% \tag{5.2}$$

The average results over the 5 runs are given in Table 5.3.

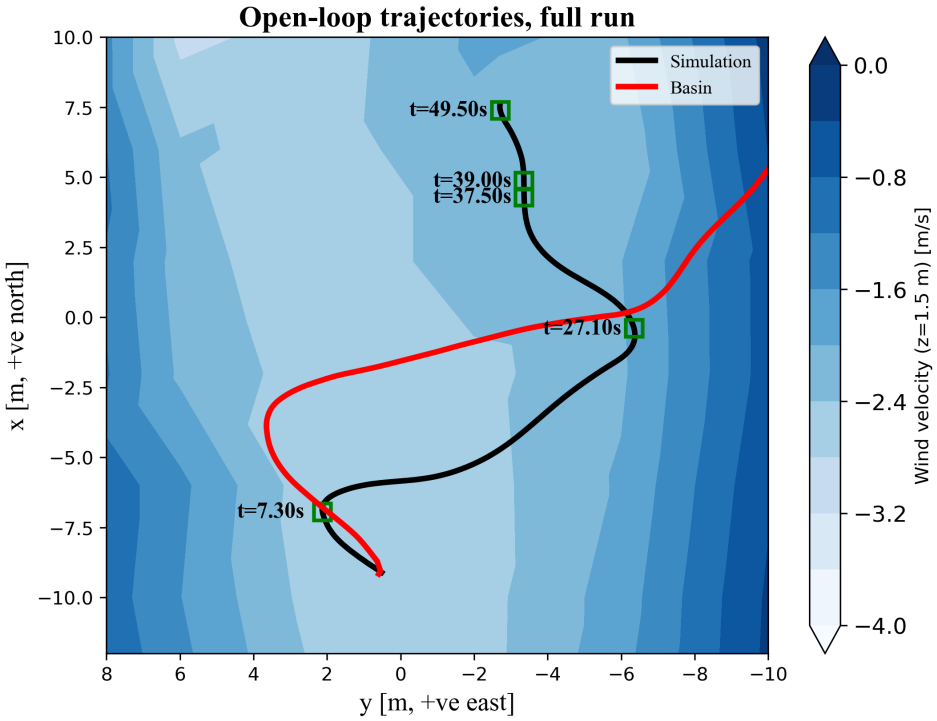


Figure 5.11: Open-loop controlled trajectories in the simulation and the basin.

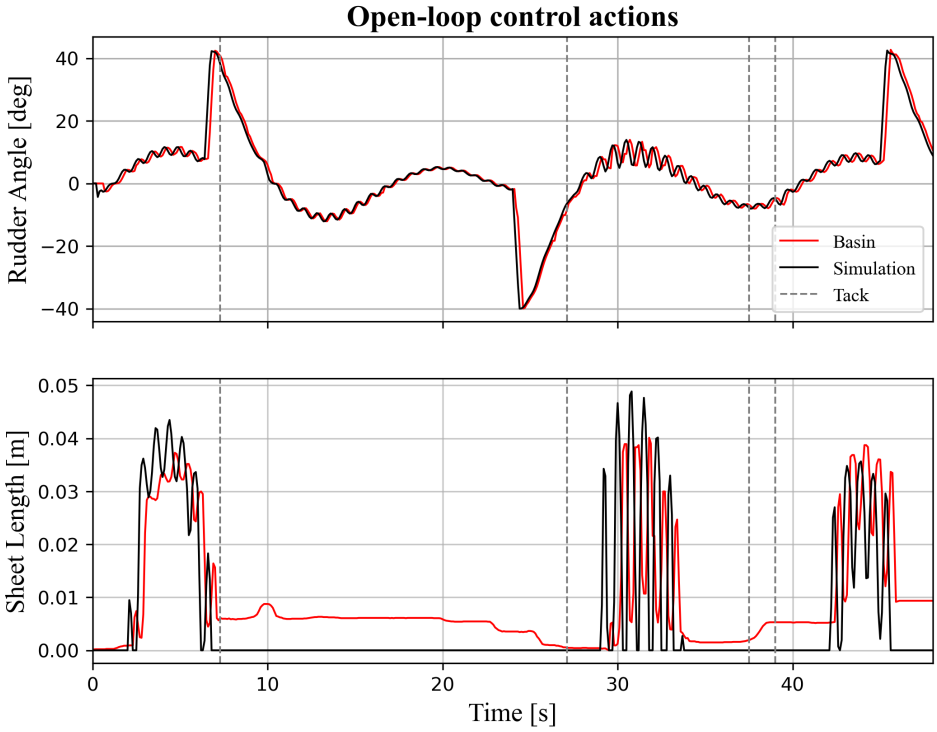


Figure 5.12: Measured rudder angle and sheet lengths in simulation and basin.

The error in the rudder angle is below 1% so it would not have significant impact on the performance in the basin. The action error of the sheet length is about 15% on average over the 5 runs.

It is clear that the trajectories deviated quite a bit, with errors accumulating over time. To get a better idea of the main sources of the errors, the basin run was split in time increments of 4 seconds, which is

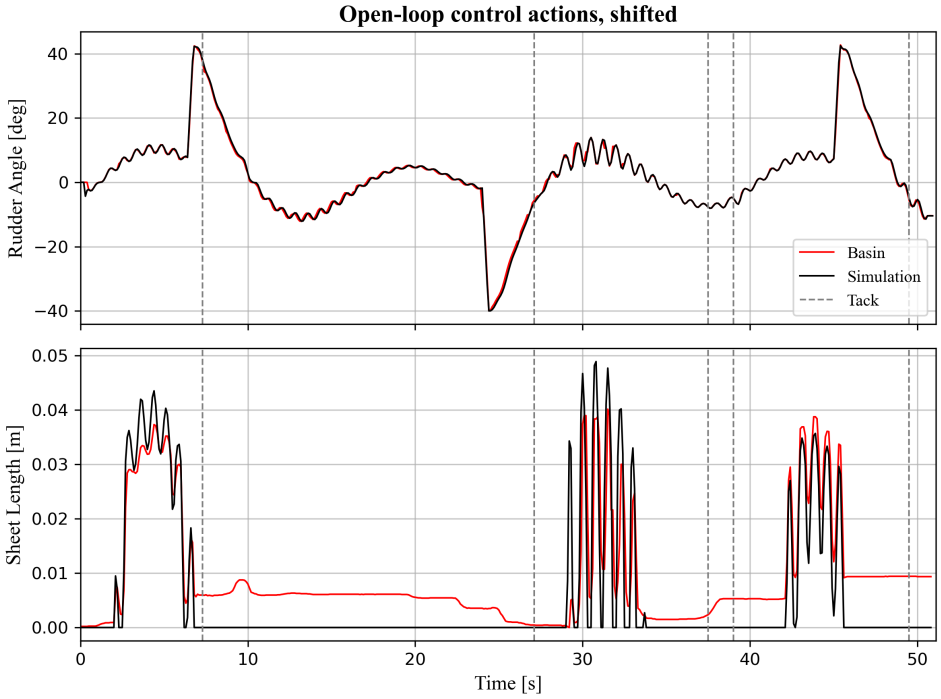


Figure 5.13: Shifted rudder and sheet length signals accounting for time delay in the basin.

Table 5.3: Average errors in the measured rudder and sheet actions.

	RMSE	NRMSE
Rudder Angle (θ_r)	0.74 $\pm$ 0.08 deg	0.89% $\pm$ 0.10%
Sheet Length (l_s)	7.54 $\pm$ 0.60 mm	14.98% $\pm$ 0.75%

approximately the average time it takes to move one body length for the Optimist. The initial conditions were obtained from the basin data and used as input for the simulation every 4 seconds for all 5 runs. The resulting trajectory of 1 of them is shown in Figure 5.14.

By systematically evaluating the data at each timestep (dt = 0.01 seconds) within each 4-second increment across the 5 runs, the Euclidean distance between the positions of the Optimist in the basin (denoted as $\mathbf{p}_{\text{basin}}$) and the simulated positions (denoted as $\mathbf{p}_{\text{sim}}$) was calculated. The positions are represented as 2-dimensional coordinates with the x and y values.

For each 4-second increment, the Euclidean distance for a single timestep between the basin position ($\mathbf{p}_{\text{basin},t} = (x_{\text{basin},t}, y_{\text{basin},t})$) and the simulated position ($\mathbf{p}_{\text{sim},t} = (x_{\text{sim},t}, y_{\text{sim},t})$) is calculated as follows:

$$d_t = \sqrt{(x_{\text{basin},t} - x_{\text{sim},t})^2 + (y_{\text{basin},t} - y_{\text{sim},t})^2} \tag{5.3}$$

Where:

- d_t represents the Euclidean distance at a single timestep t.
- $(x_{\text{basin},t}, y_{\text{basin},t})$ are the coordinates of the Optimist in the basin at timestep t.
- $(x_{\text{sim},t}, y_{\text{sim},t})$ are the coordinates of the simulated position at timestep t.

After calculating the Euclidean distance for each timestep within a 4-second increment, we compute the mean of these distances to get the mean position error for that increment, denoted as $\overline{d}$, as follows:

$$\overline{d} = \frac{1}{N} \sum_{t=1}^{N} d_t \tag{5.4}$$

Where:

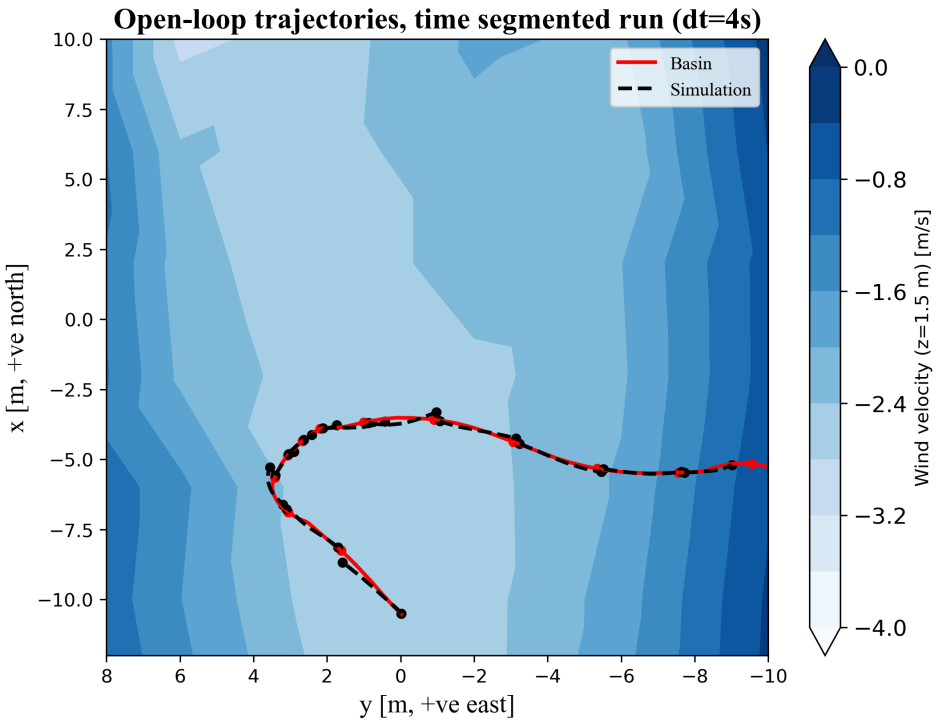


Figure 5.14: Basin run compared to the time segmented runs performed in the simulation.

- N is the total number of timesteps within a single 4-second increment.

Mean errors of the ship motions and velocities from the simulation to the basin were calculated in a similar manner, but calculating the RMSE as in Equation 5.1 and getting the average over the 4-second time increment instead of the Euclidean Distance.

5.7. Evaluation of agents in simulation

Considering the limited testing time in the basin, a careful process for selecting the best agents was set up. The training for each agent type was started at least 4 times as explained in section 5.4. The exploration process is random, therefore, although the set-up for two agents is exactly the same, they can still learn different policies. Thus, the first step is to select the agents to evaluate, since it is a waste to evaluate agents that got stuck in local optima during the training process, resulting in low overall returns. Once these agents are filtered out, the agents that are left have many policies to choose from because the policy is saved every 500 timesteps during the 500,000-timestep training period. This leaves a pool of 5,000 policies per agent type. To narrow down the options, a pre-selection is made based on the learning curve again. The training period during which the agent consistently achieves the highest rewards is determined by calculating the rolling average over a window of 100 timesteps. To capture the variability and stability of rewards within this window, the 5th and 95th percentiles of the rolling average are plotted. A smaller spread between the 5th and 95th percentiles indicates a more consistent and stable performance, while a larger spread suggests greater variability in the agent's rewards. Generally the region with the highest and most stable rewards could be found at the end of the training process.

Depending on the size of the stable region of the agent. Usually this range was about 50,000 timesteps. This still leaves 500 policies, to bring this down, the evaluation was done every 5000 timesteps instead of 500, leaving 50 policies to evaluate.

The evaluation process consists of combinations of the following initial conditions:

- **Wind Setting:** (40%, 60%)
- **X-Position:** -8 m
- **Y-Positions:** (-6, 0, 8) m

- **Yaw:** (-0.8, -0.2, 0.6) radians
- **Forward Speed:** (0.2, 0.6) m/s

These initial conditions cover several starting positions in the basin, as well as the starting speed and wind settings. This results in an evaluation of 36 episodes for each agent.

The results of the evaluated policies are examined using specific metrics to help identify the best performing and most robust agents, while also evaluating them for any risky behavior that should be avoided in the basin. The evaluation metrics are described in detail in section 5.8.

5.8. Evaluation Metrics

The metrics used for the evaluation and their respective abbreviations are as follows:

- **Success Rate (SR):** Percentage of episodes in which the agent successfully reaches the goal.
- **Time to Complete (TTC):** Average time taken by the Optimist to reach the goal in an episode.
- **Actuator Usage (AU):** Rudder, sail, and ballast usage, expressed as the average rates per episode.
- **High-Risk Behavior (HR):** Instances where the robot engages in high-risk situations (e.g., gybing, roll > 45), the amount of the risk is assessed using a step function.
- **Energy Ratio (ER):** Ratio of energy put into the rudder and ballast actuators to the wind energy utilized by the agent over the course of the completed episodes.

A weighted average of these metrics is calculated to derive an overall fitness score for the agent, facilitating a quick way to compare multiple agents. The weights ($w1$, $w2$, $w3$, $w4$, and $w5$) allow customization of the importance assigned to each metric in the overall assessment:

$$\text{Fitness} = w1 \cdot \text{SR} - w2 \cdot \text{TTC} - w3 \cdot \text{AU} - w4 \cdot \text{HR} - w5 \cdot \text{ER}$$

The weight assigned to each evaluation metric reflects its relative importance in assessing the agent's performance. In this work, the highest priority is accorded to HR due to its critical role in preventing any risk of making water and/or capsizing in the basin. Gybing, while less catastrophic, introduces highly dynamic behavior that could potentially damage the actuators. Consequently, HR receives the highest weight.

The second-highest weight is assigned to SR, emphasizing its significance in identifying the most robust agents. Agents with a high SR demonstrate resilience and consistent goal attainment when presented with the different initial conditions.

Following is the ER, where a considerable weight is placed. ER is instrumental in selecting agents that have learned policies effectively utilizing wind energy to propel themselves in successful episodes.

AU is given the second-lowest weight, acknowledging the importance of limiting actuator rates. While there are already constraints on actuator rates, this will give an idea on how effectively the agent manages and utilizes the actuators within the rate constraints.

Lastly, TTC carries the lowest weight. While this can identify the fastest agents, other aspects such as HR, SR, ER, and AU are more critical in assessing the robustness and safety, which are deemed of higher importance than the speed in this case.

The resulting weights that were used are:

- **SR:** w1 = 3
- **TTC:** w2 = 0.5
- **AU:** w3 = 1
- **HR:** w4 = 3.5
- **ER:** w5 = 2

The values for the SR, TTC, and AU are fairly straightforward to calculate in the evaluation. The success rate is the percentage of episodes that reached the target, as mentioned before. The TTC is the average time it took to reach the target over these episodes. The AU is computed by finding the mean of the sum of the actuator rates of the rudder, ballast and sheet across all episodes. The calculation of the used values for ER and HR require more detailed explanations, which are provided in the following sections.

5.8.1. Energy Ratio

The mean energy ratio ($\overline{ER}$) of all episodes is calculated by determining the average rudder, ballast and wind energy of the successful episodes in the evaluation. The mean rudder and ballast energy fromEquation 5.8 and Equation 5.10 respectively, is the energy that was put into moving the rudder and the ballast, this is divided by the wind energy that was utilized by the Optimist to propel itself forward, see Equation 5.11:

$$\overline{ER} = \frac{\overline{E_R} + \overline{E_B}}{\overline{E_W}} \tag{5.5}$$

In general, the power of each component is calculated at each timestep, after which it is integrated over short time intervals. This is done to smoothen out the effects of instantaneous power inputs, as these should not dominate the calculated energy ratio. By keeping the time interval relatively small it will still capture changes in the system's dynamics over time. The time interval (Δt_i) for integration is determined by the average time it takes to move one body length. The body length of the Optimist is: $LPP = 2.041m$. The average speed (obtained from a run in the 40% wind field) is about 0.5m/s, so the integration time interval is calculated as:

$$\Delta t_i = \frac{LPP}{\overline{V_S}} = \frac{2.041\,\text{m}}{0.5\,\text{m/s}} \approx 4\,\text{s} \tag{5.6}$$

Rudder Energy

The rudder energy (E_R) is computed through a series of steps involving the rudder area (A_R), rotational velocity of the rudder (ω_R), net torque (τ), rudder power (P_R), and integration over time intervals (Δt_i). The lift and drag coefficients (C_L, C_D) that were used were obtained from the Rudder2020 data that was also used in the XMF model of the Optimist, shown in Figure 4.5.

The rudder area (A_R) is determined by multiplying the average chord length ($\bar{c}$) by the rudder height (h_R):

$$A_R = \bar{c} \times h_R$$

With:

$$\bar{c} = 0.250m$$
$$h_R = 0.3m$$

The output data contains the rudder angle (θ) at each timestep, the rotational velocity of the rudder (ω_R) is the rate of change of the rudder angle with respect to time (t):

$$\omega_R = \frac{\Delta\theta}{\Delta t}$$

The lift (F_L) and drag (F_D) forces on the rudder at each time (t) are calculated using lift and drag coefficients (C_L, C_D) interpolated from the Rudder2020 data as shown in Figure 4.5. The angle of attack (α) should be the angle of the oncoming flow of the water on the rudder, however, there is no information available of the direction of the water flow information so the rudder angle (θ) was used to approximate the angle of attack of the flowing water:

$$F_L = \frac{1}{2} \cdot \rho_w \cdot V_S(t)^2 \cdot A_R \cdot C_L(\alpha)$$
$$F_D = \frac{1}{2} \cdot \rho_w \cdot V_S(t)^2 \cdot A_R \cdot C_D(\alpha)$$

With:

$$\rho_w = 1025.0kg/m^3$$
$$\alpha \approx \theta$$

The net torque (τ) acting on the rudder is determined by the rudder's radial distance ($|r|$) and the lift and drag forces:

$$\tau = |r| \cdot (|F_L| \cos(\theta) + |F_D| \sin(\theta))$$

With:

$$|r| = 0.299m$$

Rudder power (P_R) is the product of the net torque and the rotational velocity:

$$P_R = \tau \times \omega_R$$

Finally, the total rudder energy (E_R) of 1 episode is obtained by integrating P_R over the time intervals Δt_i and summing those values:

$$E_R = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} P_{R,ij} \Delta t_{ij} \right) \tag{5.7}$$

With:

n : # of time steps within each time interval Δt_i

m : # of time intervals Δt_i in 1 episode

The average rudder energy over the number of episodes N in the evaluation is then calculated as follows:

$$\overline{E_R} = \frac{1}{N} \sum_{k=1}^{N} E_R \tag{5.8}$$

With:

N : # of episodes in the evaluation

For each completed episode, the sum of all these values is calculated. The average of the total rudder energy over all the episodes in the evaluation is then used to calculate the energy ratio in Equation 5.5.

Ballast Energy

The ballast energy (E_B) is computed through a series of steps involving the ballast position (x_B), ballast mass (m_B), ballast velocity (V_B), ballast acceleration (a_B), force (F_B), displacement (D_B), work done (W_B), and integration over time intervals (Δt_i).

The ballast velocity (V_B) is calculated as the difference in ballast position over time:

$$V_B = \frac{\Delta x_B}{\Delta t}$$

The ballast acceleration (a_B) is the rate of change of velocity:

$$a_B = \frac{\Delta V_B}{\Delta t}$$

The force acting on the ballast (F_B) is determined by multiplying the mass of the ballast by the acceleration:

$$F_B = m_B \cdot a_B$$

With:

$$m_B = 10kg$$

The ballast displacement (D_B) is the absolute difference in sequential ballast positions:

$$D_B = |\Delta x_B|$$

The work done by the ballast (W_B) is the product of force and displacement:

$$W_B = F_B \times D_B$$

Finally, the ballast energy of each time interval is obtained by integrating W_B over the time intervals. The sum of those values over the episodes gives the total ballast energy used in 1 episode:

$$E_{B,i} = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} W_{B,ij} \Delta t_{ij} \right) \tag{5.9}$$

With:

n : # of time steps within each time interval Δt_i

m : # of time intervals Δt_i in 1 episode

The average ballast energy over the number of episodes N in the evaluation is then calculated as follows:

$$\overline{E_B} = \frac{1}{N} \sum_{k=1}^{N} E_B \tag{5.10}$$

With:

N : # of episodes in the evaluation

Wind Energy

Next, the wind energy is computed through a series of steps involving the sail area (A_s), apparent wind speed (V_A), angle of attack (α), lift and drag coefficients (C_L, C_D), lift (F_L) and drag (F_D) forces, sail drive force (F_R), wind power (P_W), and integration over time intervals (Δt_i).

The lift and drag coefficients used in the calculations are obtained from the paper *'Design of a Foiling Optimist'* [24], see Figure 2.9. Andersson et al. obtained the coefficients from tests done with an Optimist sail at 1/4 scale in a wind tunnel, their test set-up shown in Figure 2.8.

The sail area of the Optimist is the same as has been used in the XMF model:

$$A_s = 3.3\,\mathrm{m}^2$$

The output of XMF for the wind heading is given in 'going-to definition', which means that a wind heading between $270°$ and $90°$ angles means the Optimist is going upwind, and between $270°$ and $90°$ is downwind. To comply with the coefficients in Figure 2.9, the apparent wind heading (β_A) is transformed to a going-from definition, in the range $[-180°, 180°]$. This means that upwind is $0°$ and downwind $(-)180°$. With that, the apparent wind speed (V_A) is calculated as follows:

$$V_A = \sqrt{V_{tw}^2 + V_S^2 + 2 \cdot |V_{tw}| \cdot |V_S| \cdot \cos(\beta_A)}$$

The angle of attack (α) is the taken as incoming wind on the sail by subtracting the boom angle (β_B) from the apparent wind heading. The coefficients are interpolated from Figure 2.9 and the lift and drag force on the sail are calculated at all timesteps t:

$$F_L = \frac{1}{2} \cdot \rho_a \cdot V_A^2 \cdot A_s \cdot C_L(\alpha)$$

$$F_D = \frac{1}{2} \cdot \rho_a \cdot V_A^2 \cdot A_s \cdot C_D(\alpha)$$

With:

$$\rho_a = 1.225 kg/m^3$$

$$\alpha \approx \beta_A - \beta_B$$

Next, using the force diagram and equations from subsection 2.1.4, specifically Equation 2.4, the sail drive force is calculated as follows:

$$F_m = F_L \cdot \sin(\alpha) - F_D \cdot \cos(\alpha)$$

The calculation is simplified by assuming that the leeway angle (λ) is negligibly small. F_m represents the forward driving force generated by the wind on the sail.

The wind power (P_W) is obtained by multiplying the drive force by the forward speed of the Optimist:

$$P_W = |F_m| \cdot V_S$$

Finally, the total wind energy (E_W) of 1 episode is obtained by integrating P_W over the time intervals and summing all the values of the episode:

$$E_W = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} P_{W,ij} \Delta t_{ij} \right)$$

With:

n : # of time steps within each time interval Δt_i

m : # of time intervals Δt_i in 1 episode

Similar to the ballast and rudder energy values, the resulting E_W is a value of the wind energy over the time interval Δt_i. The mean wind energy over the number of episodes N is calculated as follows:

$$\overline{E_W} = \frac{1}{N} \sum_{k=1}^{N} E_W \tag{5.11}$$

With:

N : # of episodes in the evaluation

The average of the total wind energy over all the episodes in the evaluation is then used to calculate the energy ratio in Equation 5.5.

5.8.2. High Risk Behavior

To assess the agents' high-risk behaviors, the focus is on two aspects: **gybing** and **heeling**. Preventing gybes is important within the basin environment due to the highly dynamic behavior involved in a gybe, which could increase the risk of capsizing and damage to the actuators. The heeling angle is used to identify and mitigate agents that take too much risk to prevent capsizing and making water in the basin.

The number of gybes are counted over all the episodes and averaged over the total time of the full evaluation.

To assess the capsize risk, a step function is used to assign a function value to the heeling angle output from XMF at every timestep. The step function is given in Equation 5.12. The maximum heeling angle (θ_{max}) is calculated based on the stability parameters of the Optimist from the XMF model in Equation 5.13. This step function is designed to apply no penalty for heeling angles up to 20% of the maximum heeling angle. This conservative threshold is chosen to prevent heeling angles in the basin that could potentially lead to water intake and damage to the systems onboard the Optimist.

$$f(\theta, \theta_{\text{max}}) = \begin{cases} 0, & \text{if } \theta \leq 0.2 \cdot \theta_{\text{max}} \\ \exp\left(\frac{\theta - 0.2 \cdot \theta_{\text{max}}}{0.2 \cdot \theta_{\text{max}}}\right) - 1, & \text{if } 0.2 \cdot \theta_{\text{max}} < \theta \leq \theta_{\text{max}} \end{cases} \tag{5.12}$$

With θ_{max} as the maximum heeling angle calculated as:

$$\theta_{\text{max}} = \arctan\left(\frac{GM}{BM}\right) \approx \arctan\left(\frac{0.713m}{0.556m}\right) \approx 52^\circ \tag{5.13}$$

With:

$$BM = 0.556\,\text{m}$$
$$KG = 0.305\,\text{m}$$
$$KM = 0.943\,\text{m}$$
$$GM = KM - KG = 0.713\,\text{m}$$

The resulting heeling value function is shown in Figure 5.15.

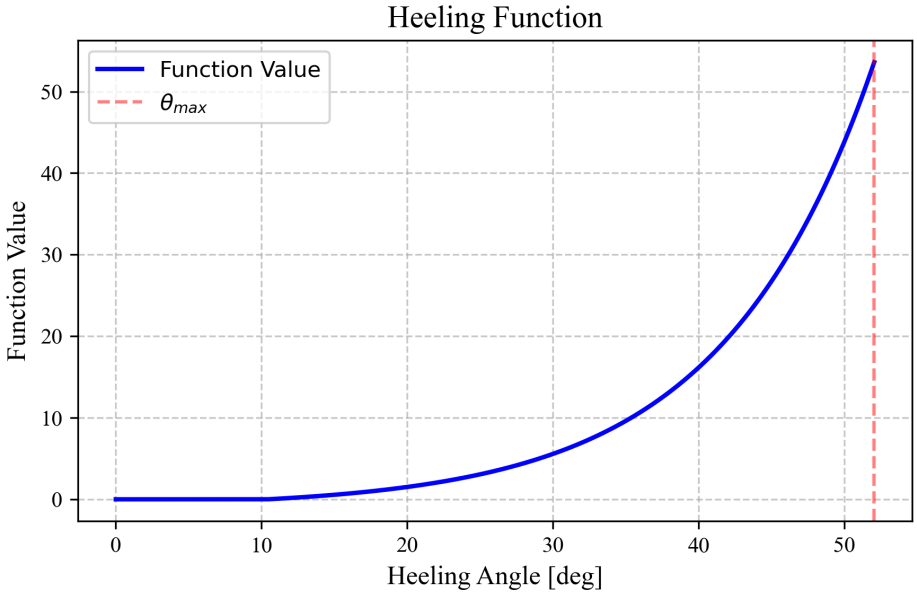


Figure 5.15: Heeling function to assess the capsize risk behavior in the evaluation of the agents.

5.9. Evaluation of agents in the basin

To test the agents in the basin, the set up as described in section 5.2 was used. The allotted basin time was about 2 hours for 4 days, for a total of 8 hours. This time was used to run the open-loop control tests, the state-of-the-art control and the different types of RL agents resulting from the selection process as described in section 5.7. Before starting any tests, the measurement systems were calibrated. The rudder, ballast and sheet were zeroed. The position measurement system was calibrated as well, making sure that the carriage followed the Optimist so that at least 5 LEDs were kept in the frame and the global position could be recorded. Once all the systems were verified to work, preliminary sanity checks were performed using the open-loop controls. This led to the discovery that the action that was sent for the rudder angle had to be reversed for example, due to the configuration of the basin signal being the other way around. Apart from this adjustment, the integration of the basin with the XMF environment and the control over the Optimist worked as desired. The fans were set at levels of either 40% or 60% and a buffer period of about 2 minutes was allowed for the wind to stabilize before proceeding with the tests.

Safety was of the highest importance during the testing procedure. A basin crew of at least 3 people was always present to oversee the systems in the basin, the hardware on the Optimist and to control the carriage. Two members of the crew were present in the basin at all times during the tests. They could intervene at any time to make sure that the Optimist did not capsize, hit the carriage or perform any other type of dangerous behavior. Ropes were also placed at the borders of the basin to stop the Optimist from hitting the sides or the fans at the north side. They were also important as there was no system in place to bring or sail the Optimist back to the desired starting position so this was done manually.

To use the limited time in the basin effectively, it was not feasible to perform all of the same runs as in the simulation for a single agent, described in section 5.7. Instead, 3 different starting positions with both wind levels were used for each agent with the aim to repeat runs at least 2 times with the same initial conditions to investigate the consistency and reliability of the results. The basin is divided up with a coordinate system of letters and numbers. The starting positions were on the height of 'D', which corresponds to $x \approx -10$ and at either 4, 8 or 12, corresponding to $y \approx [7.5, 0, -7.5]$ respectively. The starting heading was -30° when starting at D4 and 30° at D8 or D12.

The procedure for a test in the basin was as follows:

1. The Optimist was moved to its starting position by the two people in the basin.
2. Information of the test was documented, such as: the type of control/agent, the starting position and the current wind level.
3. The controls file of the agent to test was loaded into the system, checking if the connection with the basin was in place.
4. Another person started the recording, which in turn sounded a horn signalling to the people in the basin to start the run.
5. The run was started by giving the Optimist a small push.
6. The run continued until the Optimist either sailed out of bounds or reached the target, which was set at x=10.*
7. At that point, the Optimist was grabbed by the people in the basin, bringing it back to the starting point of the next run.

* In some cases the runs had to be stopped early. For example, when the carriage was unable to follow the Optimist. It was unable to follow either because the Optimist would sail too fast or when the Optimist would gybe. Another case was the Optimist heeling too far, making water.

6 Results

The results section is split into two parts: the reality gap and the results of the transfer from simulation to reality.

6.1. What is the extent of the reality gap?

In the following, the results of the open loop run in the simulation (XMF environment) and the basin are compared.

First, the results of the calculation of the mean position error with Equation 5.4 for all the time incremented runs are shown in Figure 6.1.

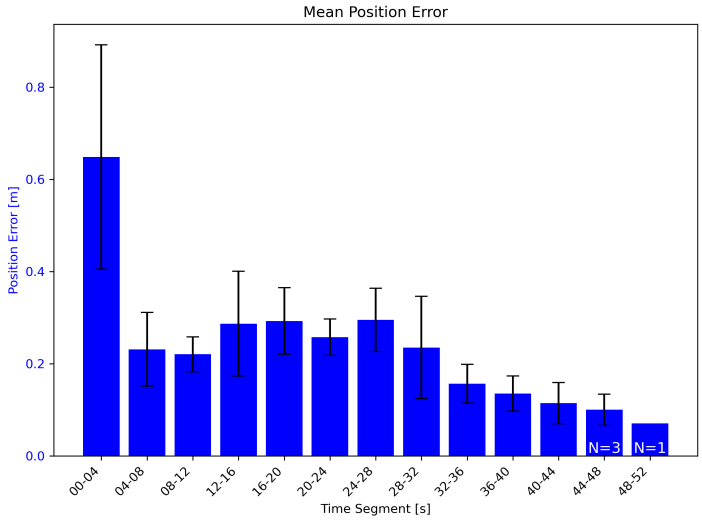


Figure 6.1: Mean positon error over 5 runs in each 4-second time segment. The nr of runs in the last two time segments is less since the durations of the runs were not consistent.

The mean position error of the first time increment of 4 seconds is relatively high, which is due to the start-up procedure in the basin. For this reason, the first 4 seconds of each run were excluded from the modeling error analysis. The remaining 4 second increment runs (total N=54) were used to create a correlation matrix using Pearson's correlation coefficient. Besides the mean position error, the RMSE of speed (x), speed (y), yaw, roll and pitch were calculated for each increment. They were compared to the average values of the wind speed and heading, the pitch, roll and yaw rates, the rudder rate, speed (x) and speed (y). The resulting correlation matrix is shown in Figure 6.2

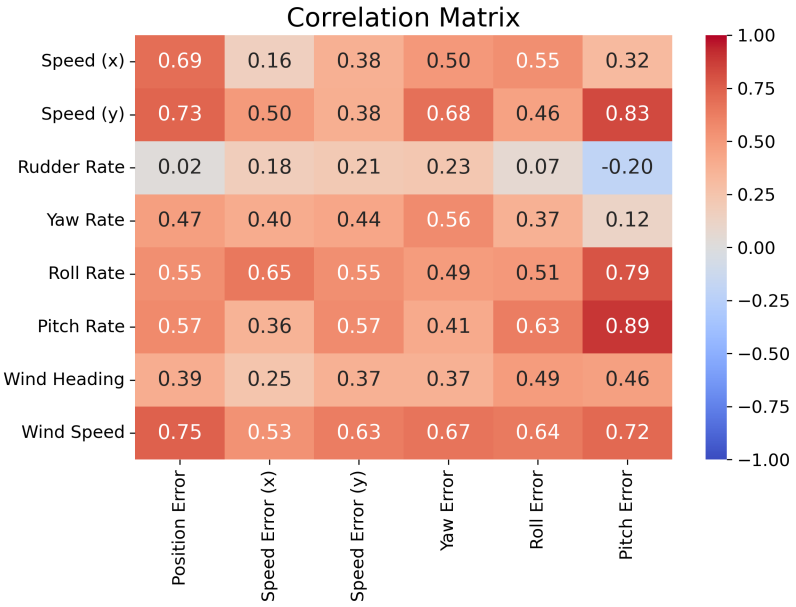


Figure 6.2: Correlation matrix

There are quite a few positively correlated relationships, with the pitch error showing the strongest correlation with the pitch and roll rate, and speed (y). However, it is hard to make out what is affecting what since most relationships seem to be somewhat correlated. Since the wind speed is positively correlated with every single error, it can be seen as sort of an 'amplitude' of the dynamic system, which is logical when looking at sailing. Therefore, to make things more clear, partial correlation for each pair is computed with the wind speed as the control variable, mitigating the effects of wind speed. The resulting partial correlation matrix is shown in Figure 6.3.

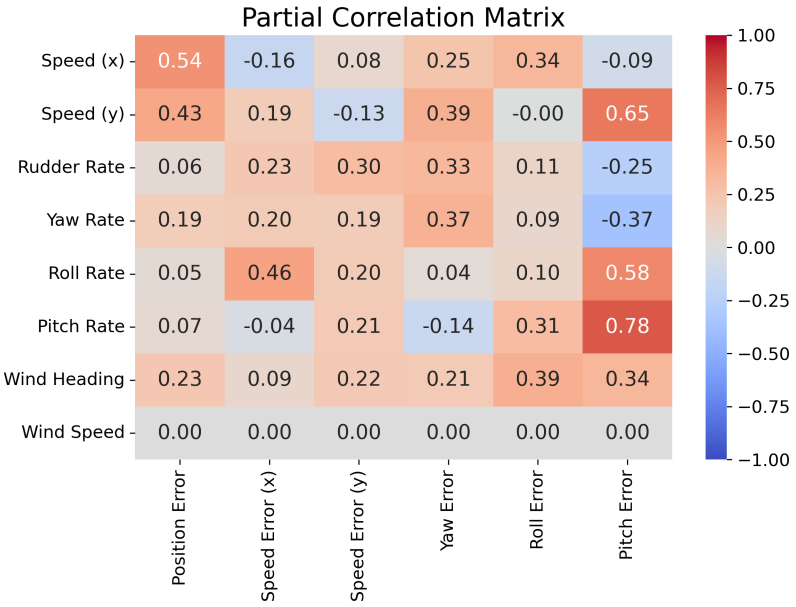


Figure 6.3: Partial correlation matrix, mitigating the effects of the wind speed.

The examination of the partial correlation matrix reveals that the pitch error continues to exhibit the most prominent positive correlations. This implies that pitch error could be a significant contributor to the overall modeling error. Other correlations worth noting are the speed in x- and y-direction with

the position error and between the speed in x-direction and the roll rate, which has a value of r=.46, suggesting a moderate correlation.

The average calculated errors over all time segments for all 5 runs are shown in Table 6.1 below.

Table 6.1: Average RMSE and standard deviation over all the time segments of the 5 runs. of the position, speed in x and y, and the roll, pitch and yaw motions.

	Average error
Position	0.2 ± 0.06 m
Speed (x)	0.03 ± 0.02 m/s
Speed (y)	0.03 ± 0.01 m/s
Roll	0.75 ± 0.31 deg
Pitch	0.35 ± 0.08 deg
Yaw	8.64 ± 4.87 deg

The average RMSE of the yaw error notably high, with a high standard deviation as well, suggesting large variability over the time segments. Figure 6.4 compares the yaw motion of one of the time segmented runs to the yaw motion measured in the basin, it is clear that the simulated yaw motion deviates from the basin quite fast, especially in between the first and second tack maneuver. This is also where the wind speed is the strongest. To assess the apparent contribution of pitch error to the

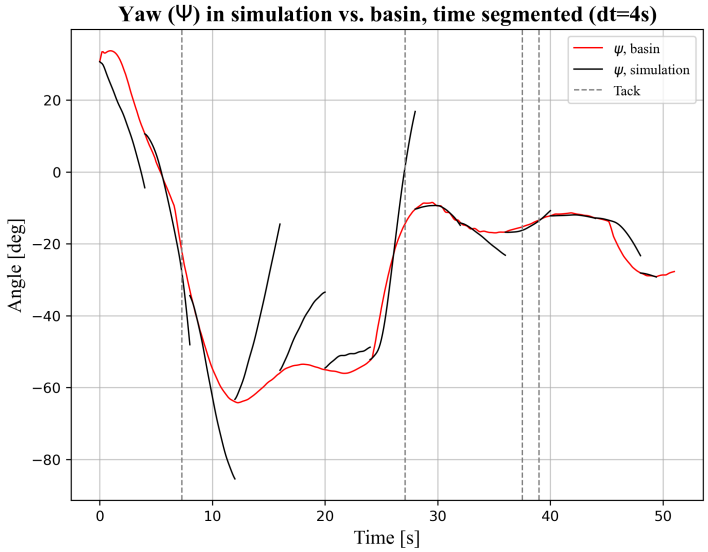


Figure 6.4: Segmented yaw motion from the simulation and the yaw motion from the basin

overall modeling error further, the pitch signal of the time segmented run in the simulation is compared to the corresponding basin runs. One of the results is shown in Figure 6.5. Overall, the simulation seems to overestimate the amplitude of the pitch motion. Especially during and between the tacking maneuvers, this can also be seen when looking at the average RMSE pitch error over all the runs in Figure 6.6. The time segments that contain a tack maneuver (in the simulation) are shown in green. The RMSE is higher in between the tack maneuvers. The speed in y direction and the roll rate are plotted as well to show the moderate positive correlation. The mean position error is interesting to look at as well, showing the correlations with speed in x and y-direction in Figure 6.7. The mean position error is highest during the second tack maneuver. The relatively large error bars suggest that there is considerable variability in the position errors during the time segments.

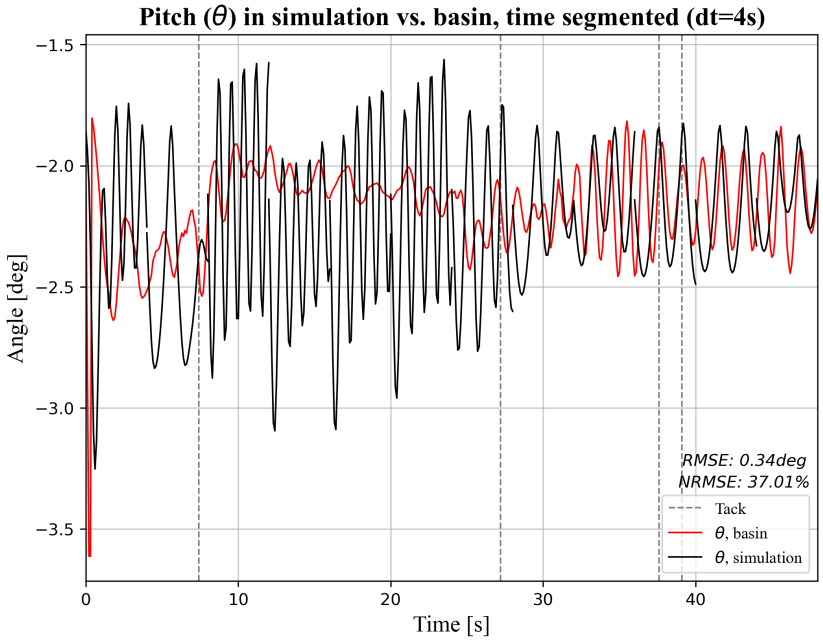


Figure 6.5: Segmented pitch motion from the simulation and the pitch motion from the basin

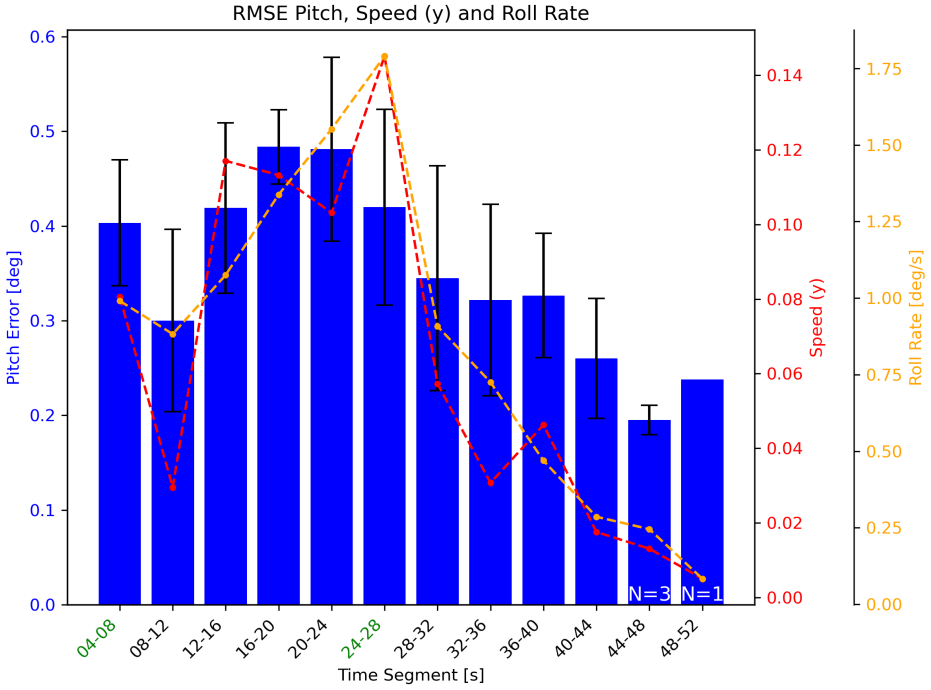


Figure 6.6: Mean pitch errors over the time segments, compared to the average measured speed in y-direction and the roll rate. The time segments that contain a tack maneuver (in the simulation) are shown in green.

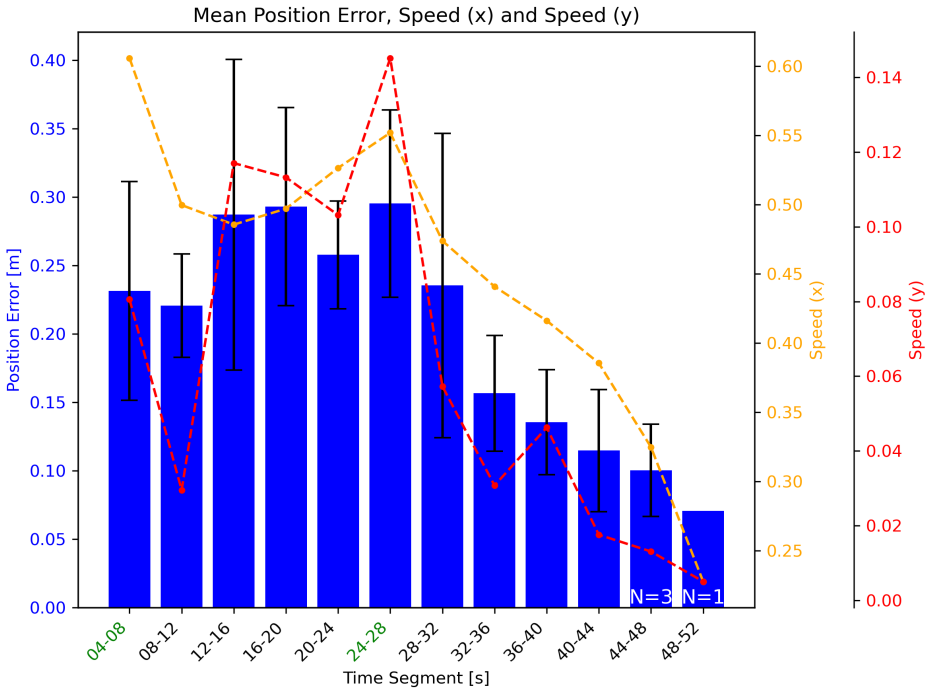


Figure 6.7: Mean position errors over the time segments, compared to the average measured speed in x- and y-direction. The time segments that contain a tack maneuver (in the simulation) are shown in green.

6.2. What agents were selected to be tested in the basin after the evaluation in the simulation?

As outlined in section 5.7, the agents with the highest overall fitness scores from each trained type were selected for testing in the basin. The following sections will discuss these specific agents, they will be referred to as follows:

- **SAC** : SAC regular
- **SAC_RI_3** : SAC trained with Domain Randomization
- **SAC_RI_4** : SAC trained with Domain Randomization (exactly the same set up as SAC_RI_3)
- **SAC_ON_01** : SAC trained with Domain Randomization + Observation Noise 1%
- **SAC_ON_04** : SAC trained with Domain Randomization + Observation Noise 4%
- **SAC_ON_10** : SAC trained with Domain Randomization + Observation Noise 10%

SAC_RI_3 and SAC_RI_4 are both included in the analysis as they both performed comparably in the simulation evaluation. All of the setups used to train these agents and the data on the learning process can be found on the supplementary website, the link can be found in Appendix A. The time it took to train each of the agents is given in Table 6.2 below.

Table 6.2: Summary of training times and number of episodes of the agents

Name	Duration	Nr. of Episodes
SAC_REG	21 hours 19 min	4062
SAC_RI_3	23 hours 1 min	3821
SAC_RI_4	23 hours 10 min	3795
SAC_ON_01	20 hours 39 min	4067
SAC_ON_04	18 hours 37 min	3062
SAC_ON_10	21 hours 52 min	3021
Average	*21 hours 26 min*	*3638*

The average computation time per episode is approximately 20 seconds. A run in the basin took about 3 minutes on average, including time to bring the Optimist back to its starting position and getting the systems ready. That means that the average training time of 3638 episodes would take ≈ 180 hours in the basin, corresponding to 22.5 work days of 8 hours, not including breaks. Such a substantial time commitment underscores the critical role and efficiency of simulation in the training process.

6.3. How did the selected agents perform in the basin compared to the simulation?

The average success rates and the times to complete these successful runs in the simulation and in the basin are summarized in Figure 6.8 and Figure 6.9. The figures compare the performances of the chosen SAC agents in the task of reaching the upwind target in both simulation and basin (real-world) environments.

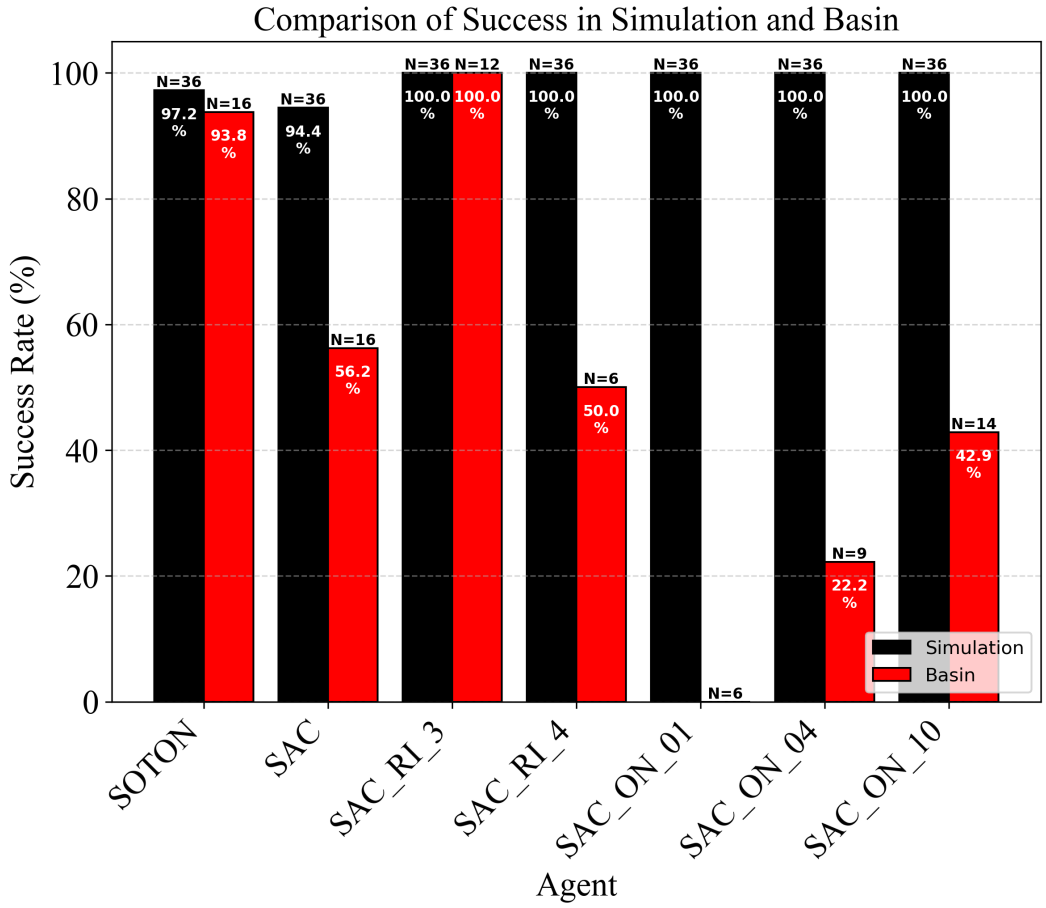


Figure 6.8: Success rate of the agents in the simulation and basin.

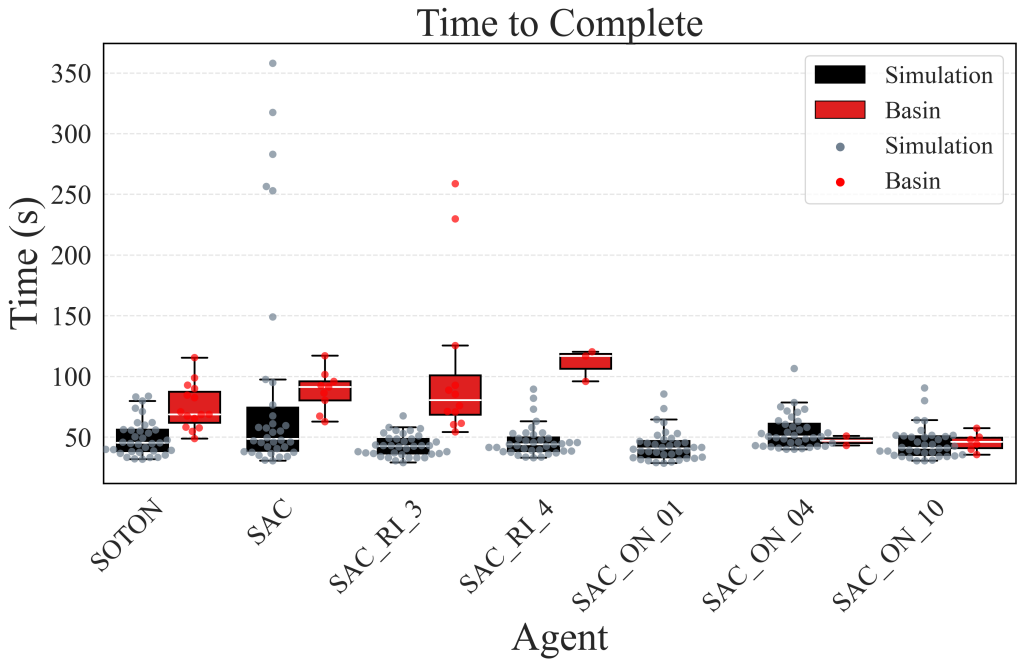


Figure 6.9: Completion times of the agents in the simulation and the basin over all the successful runs.

First, the SOTON agent, representing the state-of-the-art classic sailing control, performed impressively in both the simulation and basin tests. In each environment it was able to complete the task all but 1 times, resulting in success rate of 97.2% in the simulation environment over 36 trials and 93.8% across 16 trials in the basin environment. The average completion time from simulation to the basin is a bit slower, as observed for most agents. The only basin run that did not succeed ran out of time, as the Optimist was not able to generate enough speed to sail upwind, the trajectory is shown in Figure 6.10. Even in the failed run, the SOTON agent showed robust behavior in the sense that it was still sailing within the boundaries and not engaging in risky behavior. This underscores the robustness and reliability of classic control methods when transferring from sim-to-real and in its performance in general.

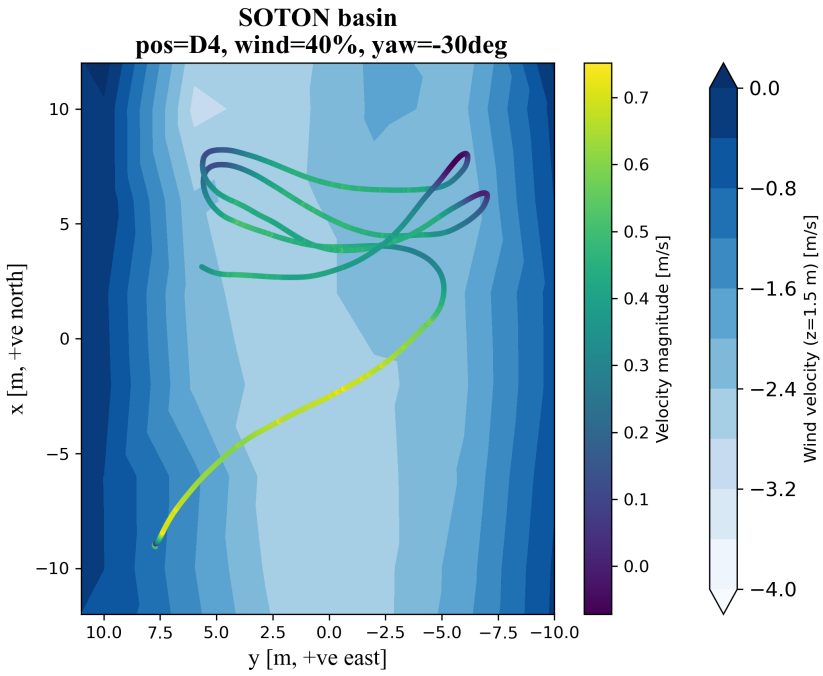


Figure 6.10: SOTON's single failed run, showing that the speed loss after the tacks is detrimental.

The SAC regular agent, achieved a success rate of 100% in the simulated trials, but with varying completion times. These are due to the different initial conditions it was evaluated on, which where not encountered during training. Surprisingly, it was still able to complete the task every time. In the basin however, it encountered more of a challenge, with its success rate falling to 56.2% over 16 trials. The agent was able to do quite well in the 40% wind field in the basin, with most runs succeeding from the different starting positions. The policy it had learned did seem to profit from some 'cheating' behavior in the basin, once it gained a bit of speed it would turn into the wind and use quick movements of the rudder (*sculling*) and/or the ballast (*rocking*) to propel itself. This could explain why most of the failures in the basin occurred in the 60% wind field, as the wind was too strong to succeed by trying to go directly into the wind. The only times it was able to complete the task in the 60% wind field was from the same starting position as which it was trained at, with 2 out of the 3 runs being successful as shown in Figure 6.11. Overall, the regular SAC agent is not very robust to the changes in the environments.

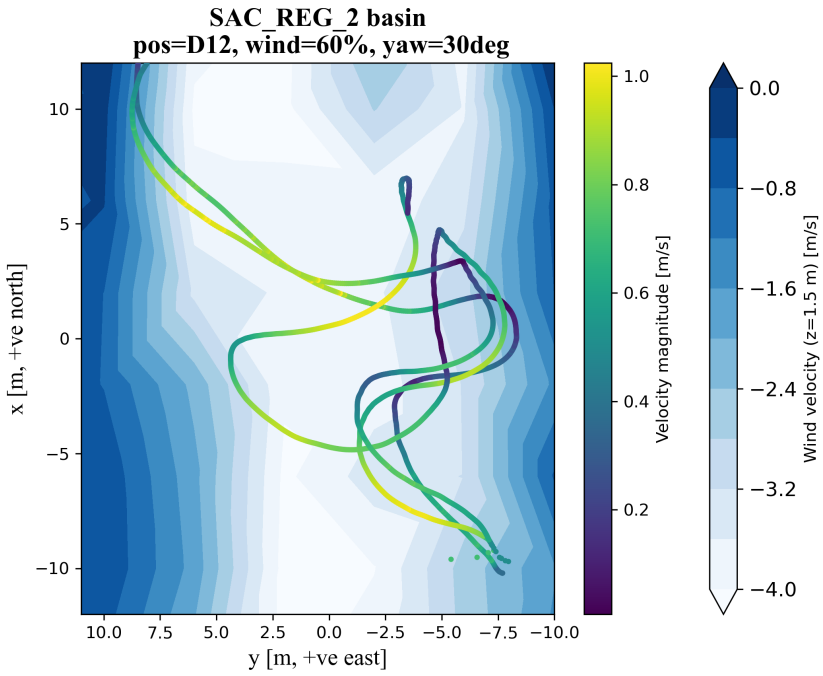


Figure 6.11: 3 basin runs of SAC_REG from D12 with 60% wind.

A particularly intriguing aspect of the study is the comparison between SAC_RI_3 and SAC_RI_4, both of which were subjected to the same domain randomization training. Despite identical training set-ups and comparable performances in the simulation, their outcomes in the basin were quite different. SAC_RI_3 maintained an impressive success rate of 100% over 12 trials, mirroring its simulation performance. Conversely, SAC_RI_4, which had demonstrated a 100% success rate in simulation, saw its basin success rate halved to 50% across 6 trials. A Fisher's Exact Test yielded a p-value of 0.025, indicating a statistically significant difference in performance between the two agents. This discrepancy between SAC_RI_3 and SAC_RI_4 highlights the inherent variability in learned policies by RL agents, even when identical training set ups are employed. It also shows that, even when using the extensive evaluation process in the simulation, the RL agent performance in the real-world is hard to predict.

Upon examining SAC_RI_3's performance in the basin more closely, its results were quite impressive reaching a success rate of 100%. To fairly compare the success rates of SAC_RI_3 and SOTON, Fisher's Exact Test was applied, yielding a high p-value of 1.0. This outcome indicates that there is no statistically significant difference in the success rates between the two models at the conventional significance levels. Furthermore, except for two outliers, the completion times seem to be generally comparable to the SOTON agent. This is underscored using the Mann-Whitney U test, resulting in a p-value of 0.252, meaning that the distributions of the data are similar.

When taking a closer look at the two runs by SAC_RI_3 that took more time to complete, they were both started with the same initial conditions (D4, 40% wind) and they both ended up taking around 250 seconds to complete the task. During these runs, the Optimist got stuck in the wind but was able to steadily recover itself and resume sailing to the finish line. Supplementary videos of the runs are available for reference on the website, see Appendix A.

SAC_RI_4 succeeded in half the runs in the basin, with all of the successful runs being in the wind field of 40%. In the failed runs in the 60% wind field, it didn't fail horribly. It was able to sail quite well, staying within the boundaries and demonstrating that it was trying to go upwind by beating. However, it is unable to make enough progress going upwind due to drifting after a tack maneuver in the failed runs, thus running out of time. This can be seen in Figure 6.12 and in the supplementary videos.

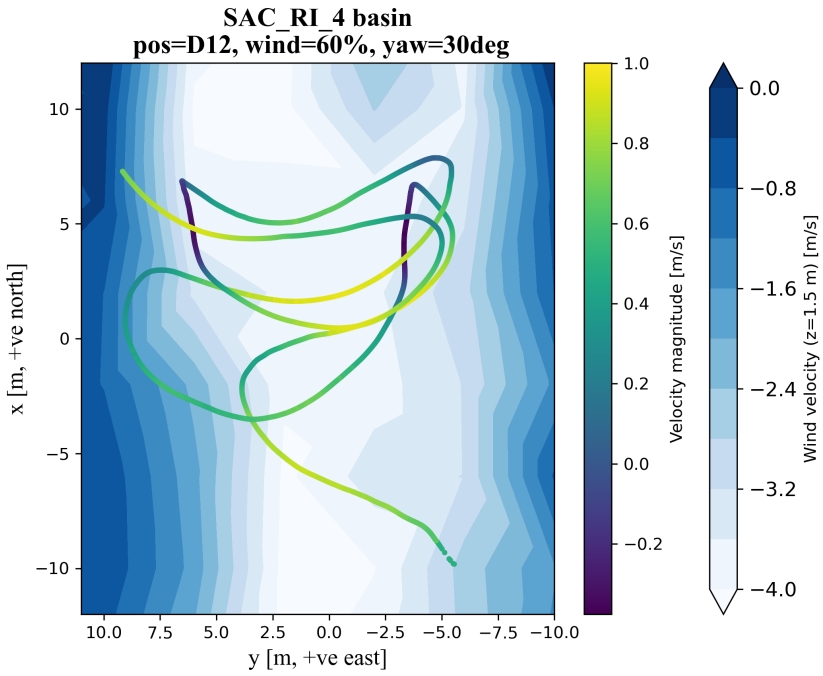


Figure 6.12: 1 basin run of SAC_RI_4 from D12 with 60% wind level.

Finally, the agents trained with additional noise added during the trainings all achieved a success rate of 100% in the simulation. However, the performances in the basin environment drastically dropped, with SAC_ON_01 failing to succeed in any of the 6 trials, and not showing an ability to tack in any of the runs and going out of bounds quickly, indicating a complete inability to generalize its simulation training to the real-world conditions encountered in the basin. SAC_ON_04's success rate dropped to a critical 22.2% over 9 trials, and SAC_ON_10 declined to 42.9% in 9 trials.

Similar to SAC_ON_01, SAC_ON_04 also did not demonstrate the ability to tack. The only two successful runs occurred in a 60% wind field starting from the corner position (D12) as shown in Figure 6.13. It manages to gain speed in the areas with high winds, after which it slowly turns into the wind in the areas with a lower wind speeds to use the gained speed to progress towards the finish line. This last part is occasionally aided by some cheating like sculling. Although this strategy was not robust — failing to reach the target from other starting conditions and in the other attempt in Figure 6.13 — it did result in a fast average completion time on the occasions when it was successful.

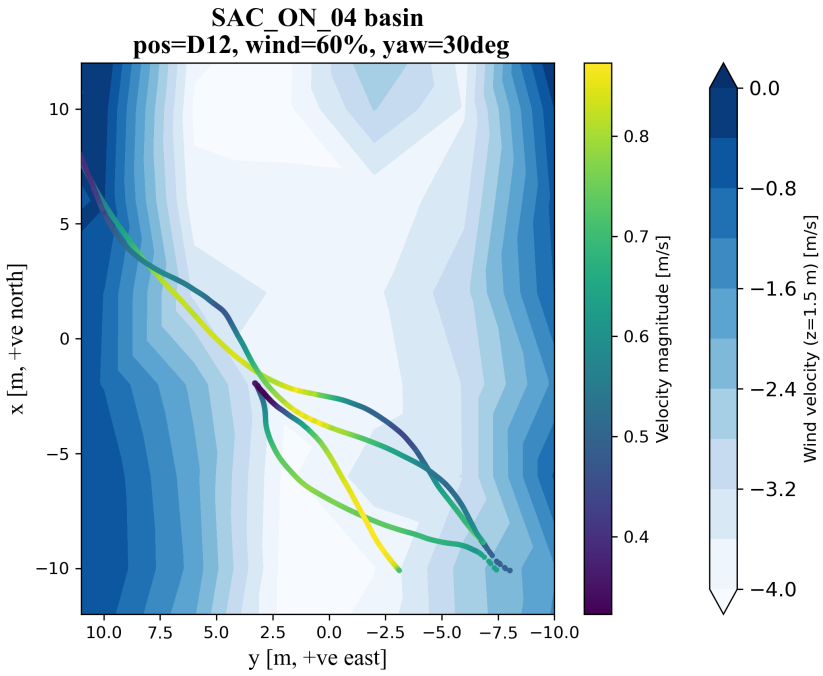


Figure 6.13: 3 basin runs of SAC_ON_04 from D12 with 60% wind level.

Finally, SAC_ON_10 was also only able to succeed in the higher wind speeds in the 60% wind field. This agent did show the ability to tack, but did seem to avoid tacking if possible, as shown in Figure 6.14. In one of those runs, it utilizes the same type of strategy as described for SAC_ON_04 before. In the other two, it does choose to tack and reaches the finish line on the other side. The agent uses its ballast to get through the wind when tacking, resulting in tacks that are quite smooth and don't result in a major loss of speed. Therefore, this strategy does result in a fast average completion time for this agent.

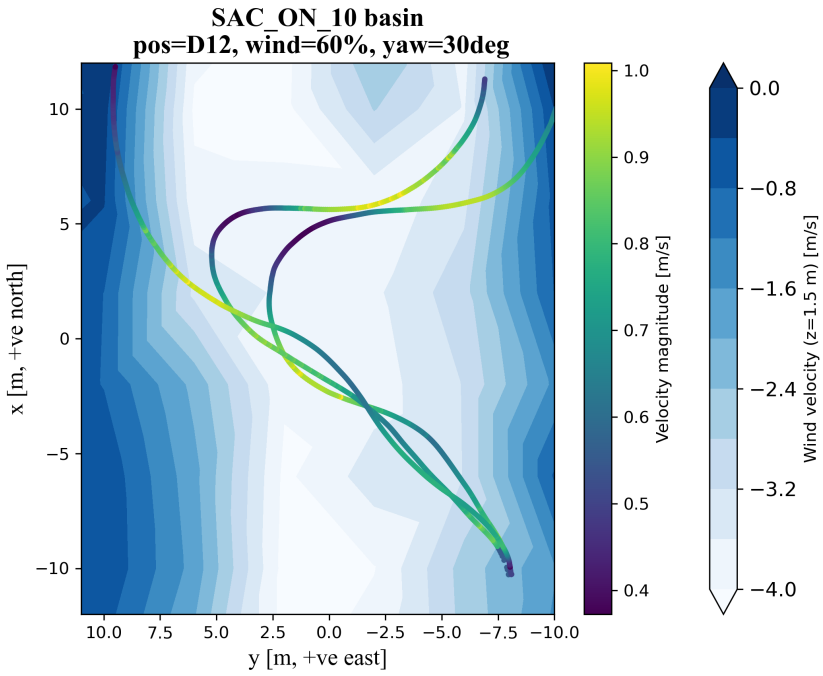


Figure 6.14: 3 basin runs of SAC_ON_10 from D12 with 60% wind level.

Overall, the basin runs of the agents trained with added noise suggest that the noise that was added during training hinder the agent's ability to adapt to real-world scenarios in a robust manner. The agents seem to take more risks to try and reach the finish line as fast as possible, resulting in many failed runs.

In summary, the variance in success rates from simulation to basin highlights the complex nature of real-world adaptability. SAC_RI_4's unexpected drop in the basin, despite the same domain randomization training as SAC_RI_3, underscores the lack of a guarantee for consistent outcomes. Moreover, the lack of robustness showed by the agents trained with different levels of noise questions the efficacy of the observation noise approach. It emphasizes that the exact parameters of their application require careful calibration to ensure successful real-world application. Furthermore, these findings suggest that while DR does seem to enhance robustness to some degree, the variability in outcomes makes it hard to predict with certainty how well an agent will perform in real-world settings. This unpredictability in performance indicates that there is still much to explore and understand about the optimal training methods for these agents.

In Figure 6.15, the energy ratios of all the successful runs of the agents are summarized. The lower the energy ratio, the better, since it represents the ratio of rudder and ballast energy used compared to the wind energy harnessed in the episode.

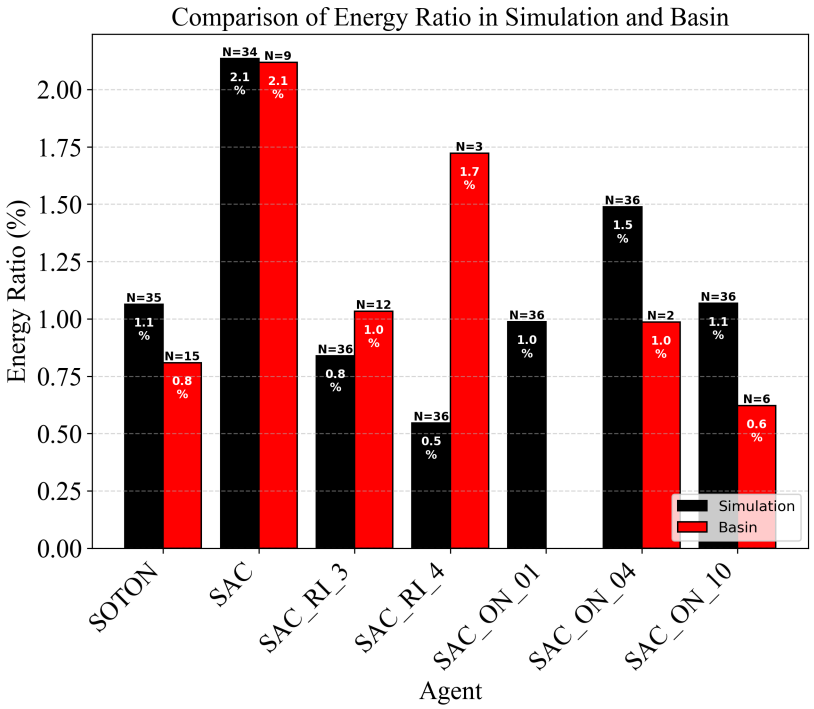


Figure 6.15: Energy ratio of all the agents in the simulation and basin. The lower the energy ratio, the more effective agents are in utilizing the wind energy.

As expected, all the energy ratios are quite low, suggesting that all the agents that are successful in reaching the target mainly use the wind energy to propel themselves. For SOTON, SAC, and SAC_RI_3, the values when going from sim-to-real are quite comparable for the agents. SAC has the highest energy ratio, indicating that it used its rudder and/or ballast to propel itself more than the wind compared to the other agents, which lines up with the observations that were described before. SAC_ON_10 reaches the lowest energy ratio in the basin, which can also be explained by the observations in the basin: the runs that it did succeed in, the agent was able to reach the target quite fast, harnessing the wind power in an effective way and managing to tack without losing too much speed by utilizing the ballast in an effective way.

7

Concluding Remarks

In conclusion, this thesis has addressed the research gap in the application of RL in domains characterized by low control authority and the dynamic, stochastic nature of environments like ASC, with a focus on the challenges of robustness and stability during sim-to-real transfer. Despite the convenience and efficiency of simulations for rapid data collection and iterative development, the transition from a simulated environment to real-world application remains a major challenge. The study presented aimed to bridge the measured modelling errors by demonstrating the viability of RL for controlling underactuated systems, specifically ASC, through a practical, small-scale project. By developing and implementing a control system using RL for a small sailboat first in a simulation and then transferring it to a real-world setting, this work provides valuable insights into the process and outcomes of the sim-to-real transfer.

First off, the research gap was addressed by measuring and analyzing the modelling errors of the used simulated environment. This way, the findings of the study increase the understanding of the extent to which these discrepancies can impact the performance and reliability of the RL agents. Quantifying the reality gap in this study addressed a critical yet often overlooked aspect in the field of sim-to-real transfer. By doing this, the findings of this research can be placed in the bigger picture, providing a benchmark for comparing different approaches and techniques.

In an attempt to bridge the reality gap, several types of RL agents were trained. For the RL framework, the goal was to prevent overoptimization of the problem at hand. This was done to show the potential of RL in complex control problems in general. Therefore, the RL framework and trainings were set up using established and benchmarked methods such as RL Glue [6], Stable Baselines3 [65] and RL zoo [66]. Using this set up, different types of RL agents were trained applying sim-to-real transfer techniques such as domain randomization and training with added noise.

To evaluate the performances of the RL agents, extensive evaluation metrics were used such as the success rate, time to complete, and the energy ratio. These same metrics were used to evaluate the performance of state-of-the-art control to compare. The metrics aid in answering the research question:

> *How robust, stable, and effective are RL strategies for sailing upwind when trained in a simulated model and applied in real life?*

Despite the advances in technology and methodology, this study confirms that sim-to-real transfer remains a significant challenge in the deployment of robust and stable RL agents. The dynamic and unpredictable nature of real-world environments, especially in scenarios with low control authority like sailing, introduces complexities that simulations struggle to accurately replicate. This disparity resulted in most RL agents failing to translate their good performances in the simulation to the real-world. The performances that did succeed were mostly effective, they managed to find ways to have runs that would match or even beat the performance of the state-of-the-art in terms of completion time. The domain randomization strategy demonstrated its effectiveness in enhancing sim-to-real transfer, resulting in policies trained in the simulation that exhibited greater robustness in the basin compared to those trained without this strategy, with SAC_RI_3 reaching a 100% success rate over 12 runs in different conditions. Yet, the stability of the trained algorithms is called into question by the performance of SAC_RI_4, which achieved only a 50% success rate. Nevertheless, the strategy employed

by SAC_RI_4 appeared more robust compared to other trained agents. Its failures were not due to risky behavior or going out of bounds, but rather due to running out of time, a challenge similar to the one encountered in the single failed attempt by SOTON. The addition of noise during training did not achieve the desired impact. The implementation of noise models was not specifically optimized or tailored to the system, indicating that mere addition of basic noise models is insufficient for training policies that improve the sim-to-real transfer.

However, it's not all rough seas. The study identified and demonstrated that certain RL agents, despite the reality gap, were able to learn robust and effective policies. This shows that with the right strategies and adjustments, the hurdle of sim-to-real transfer can be overcome. While it is clear that the current state of technology and methods is not perfect, this work has enhanced the understanding of RL in ASC and complex control problems alike. By quantifying the reality gap and using benchmarked RL methods, it lays a solid foundation for future explorations.

8

Recommendations

This chapter provides a brief overview of the recommendations to advancing the research initiated in this study.

1. **Refinement of Simulation Models:** Building on the quantification of the reality gap provided by this study, future research can focus on reducing the reality gap by improving the simulation. It would be most interesting to use the findings from this study on the reality gap, identify the areas where the modelling errors are largest and improve the simulation in those parts. For example, by reducing the pitch and yaw error. It was also clear that in higher wind speeds, all of the errors got larger. Therefore, it could be interesting to improve the fidelity of the simulation in areas with higher winds, or during tacking maneuvers for example. That way, the reality gap could be reduced where needed without having to sacrifice too much computational power and slowing down the training process for RL.

2. **Improving Sim-to-Real Transfer Techniques:** Besides improving the simulation, further research on the techniques applied to improve the sim-to-real transfer could help improving the transfer. The domain randomization was kept fairly limited, only considering randomizing the initial conditions. Further research could look into utilizing dynamics randomization for example. This was shown in the literature review to improve sim-to-real transfer in other complex control applications such as USV.

3. **Improving Hardware:** To replicate the real-world scenarios more accurately, the hardware limitations observed in this study should be addressed. With a 15% error in sheet length and the inability of simulations to account for slack, there is a clear need for direct control over the boom angle.

4. **Predictive Analysis for Real-World Performance:** Another potential interesting area of future study is the development of methods to better predict the performance of agents in the basin environment. The difference in the basin performances between agents SAC_RI_3 and SAC_RI_4 raises important questions about the underlying differences in their strategies. Investigating these differences could yield predictive indicators that identify robust performance before actual transfer to reality, streamlining the sim-to-real process.

5. **Broader Implications for the Maritime Industry:** The broader context of this research extends to its potential impact on the maritime industry. As the industry navigates towards sustainability goals, RL-controlled systems present a promising avenue for innovation. It is recommended that future studies not only continue to push the boundaries of sim-to-real transfer in this specific problem, but also consider specific use cases in the maritime domain where RL can offer benefits. Such applications could play a big part in achieving more efficient and sustainable maritime operations.

References

[1] E. Commission. “Reducing emissions from the shipping sector.” (2018), [Online]. Available: https://climate.ec.europa.eu/eu-action/transport-emissions/reducing-emissions-shipping-sector_en#documentation (visited on 06/02/2023).

[2] IMO. “Un body adopts climate change strategy for shipping.” (2018), [Online]. Available: https://www.imo.org/en/MediaCentre/PressBriefings/Pages/06GHGinitialstrategy.aspx (visited on 06/06/2023).

[3] T. Chou, V. Kosmas, M. Acciaro, and K. Renken, “A Comeback of Wind Power in Shipping: An Economic and Operational Review on the Wind-Assisted Ship Propulsion Technology,” *Sustainability 2021, Vol. 13, Page 1880*, vol. 13, no. 4, p. 1880, Feb. 2021, ISSN: 2071-1050. DOI: 10.3390/SU13041880.

[4] C. Chen, F. Ma, J. Liu, R. R. Negenborn, Y. Liu, and X. Yan, “Controlling a cargo ship without human experience using deep Q-network,” *Journal of Intelligent and Fuzzy Systems*, vol. 39, no. 5, pp. 7363–7379, 2020, ISSN: 18758967. DOI: 10.3233/JIFS-200754.

[5] C. Chen, X. Q. Chen, F. Ma, X. J. Zeng, and J. Wang, “A knowledge-free path planning approach for smart ships based on reinforcement learning,” *Ocean Engineering*, vol. 189, Oct. 2019, ISSN: 00298018. DOI: 10.1016/j.oceaneng.2019.106299.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, Massachusetts: The MIT Press, Nov. 2018, ISBN: 9780262039246.

[7] B. Zhu, J. Xu, T. Du, M. Foshey, B. Li, and A. Schulz, “Learning to Fly: Computational Controller Design for Hybrid UAVs with Reinforcement Learning,” *ACM Trans. Graph*, vol. 38, no. 4, p. 42, 2019. DOI: 10.1145/3306346.3322940.

[8] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, “Champion-level drone racing using deep reinforcement learning,” *Nature*, vol. 620, no. 7976, pp. 982–987, Aug. 2023, ISSN: 1476-4687. DOI: 10.1038/s41586-023-06419-4.

[9] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, “End-to-End Race Driving with Deep Reinforcement Learning,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2070–2075, Jul. 2018, ISSN: 10504729. DOI: 10.1109/ICRA.2018.8460934.

[10] J. Tan, T. Zhang, E. Coumans, *et al.*, “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots,” *Robotics: Science and Systems*, Apr. 2018, ISSN: 2330765X. DOI: 10.15607/RSS.2018.XIV.010.

[11] W. Zhao, J. P. Queralta, and T. Westerlund, “Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: A Survey,” in *IEEE Symposium Series on Computational Intelligence, SSCI*, Institute of Electrical and Electronics Engineers Inc., Dec. 2020, pp. 737–744, ISBN: 9781728125473. DOI: 10.1109/SSCI47803.2020.9308468.

[12] A. Irpan. “Deep reinforcement learning doesn’t work yet.” (2018), [Online]. Available: https://www.alexirpan.com/2018/02/14/rl-hard.html (visited on 04/02/2023).

[13] R. Yu, Z. Shi, C. Huang, T. Li, and Q. Ma, “Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle,” *Chinese Control Conference, CCC*, pp. 4958–4965, Sep. 2017, ISSN: 21612927. DOI: 10.23919/CHICC.2017.8028138.

[14] J. Kimball, *Physics of Sailing*. CRC Press, Dec. 2009, ISBN: 9780429193361. DOI: 10.1201/9781420073775.

[15] P. Van der Steen, “Ship Motion Prediction for the Ampelmann System,” Ph.D. dissertation, TU Delft, 2016.

[16] D. Santos, A. Negreiros, J. Jacobo, L. Goncalves, A. Silva Junior, and J. Silva, “Gain-scheduling PID low-level control for robotic sailboats,” in *Proceedings - 15th Latin American Robotics Symposium, 6th Brazilian Robotics Symposium and 9th Workshop on Robotics in Education, LARS/SBR/WRE*, 2018, pp. 118–123, ISBN: 9781538677612. DOI: 10.1109/LARS/SBR/WRE.2018.00035.

[17] R. Steinegger, “The symmetry of wings and sails,” *ZHAW Zürcher Hochschule für Angewandte Wissenschaften*, Sep. 2019. DOI: 10.21256/ZHAW-18571.

[18] F. Fossati, S. Muggiasca, and I. M. Viola, “Wind tunnel techniques for investigation and optimization of sailing yachts aerodynamics,” English, in *2nd High Performance Yacht Design Conference*, 2006, pp. 105–113.

[19] *Velocity Prediction Program (VPP) - ORC*. [Online]. Available: https://orc.org/organization/velocity-prediction-program-vpp.

[20] J. Kerwin and H. Irving Pratt, “A Velocity Prediction Program for Ocean Racing Yachts Revised to June, 1978,” *The Society of Naval Architects and Marine Engineers, SNAME, Report 78-11 Massachusetts Institute of Technology, MIT, Department of Ocean Engineering, Ocean Race Handicapping Project*, 1976.

[21] G. S. Hazen, “A model of sail aerodynamics for diverse rig types,” *The Society of Naval Architects and Marine Engineers, SNAME, March 22, 1980. George S. Hazen Yacht Design & Consulting*, 1980. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3Af8f5c069-9337-4124-aa56-8524f03b47fc.

[22] J. Gerritsma, R. Onnink, and A. Versluis, “Geometry, resistance and stability of the Delft Systematic Yacht hull series,” in *International Shipbuilding Progress, ISP*, vol. 28, Delft: TU Delft, Faculty of Marine Technology, Dec. 1981.

[23] A. B. Philpott, R. M. Sullivan, and P. S. Jackson, “Theory and Methodology Yacht velocity prediction using mathematical programming,” *European Journal of Operational Research*, vol. 67, pp. 13–24, 1993.

[24] A. Andersson, A. Barreng, E. Bohnsack, *et al.*, “Design of a Foiling Optimist,” *Journal of Sailing Technology*, vol. 3, no. 01, pp. 1–24, Sep. 2018, ISSN: 2475-370X. DOI: 10.5957.jst.2018.06.

[25] J. Keuning, K. Vermeulen, and E. J. Ridder, “A Generic Mathematical Model for the Maneuvering and Tacking of a Sailing Yacht,” in *SNAME 17th Chesapeake Sailing Yacht Symposium*, Jun. 2005. DOI: 10.5957/CSYS-2005-012.

[26] H. Saoud, M. D. Hua, F. Plumet, and F. Ben Amar, “Modeling and Control Design of a Robotic Sailboat,” in *Robotic Sailing 2013*, Springer, Cham, 2014, pp. 95–110. DOI: 10.1007/978-3-319-02276-5{_}8.

[27] Y. Masuyama and T. Fukasawa, “Tacking Simulation of Sailing Yachts With New Model of Aerodynamic Force Variation During Tacking Maneuver,” *Journal of Sailing Technology*, vol. 2, no. 10, pp. 1–34, Oct. 2011, ISSN: 2475-370X.

[28] M. Buehler, C. Heinz, and S. Kohaut, “Dynamic Simulation Model for an Autonomous Sailboat,” in *International Robotic Sailing Conference 2018*, Southampton, 2019.

[29] J. D. Setiawan, D. Chrismianto, M. Ariyanto, C. W. Sportyawan, R. D. Widyantara, and S. Alimi, “Development of Dynamic Model of Autonomous Sailboat for Simulation and Control,” *7th International Conference on Information Technology, Computer, and Electrical Engineering, ICITACEE 2020 - Proceedings*, pp. 52–57, Sep. 2020. DOI: 10.1109/ICITACEE50144.2020.9239150.

[30] A. Wolniakowski and M. Czarna, “A Framework for Model Sailing Simulation in Gazebo,” *2022 26th International Conference on Methods and Models in Automation and Robotics, MMAR 2022 - Proceedings*, pp. 81–86, 2022. DOI: 10.1109/MMAR55195.2022.9874293.

[31] R. Tedrake, *Underactuated Robotics*. 2023. [Online]. Available: https://underactuated.csail.mit.edu.

[32] R. Stelzer and T. Pröll, “Autonomous sailboat navigation for short course racing,” *Robotics and Autonomous Systems*, vol. 56, no. 7, pp. 604–614, Jul. 2008, ISSN: 0921-8890. DOI: 10.1016/J.ROBOT.2007.10.004.

[33] Y. Tipsuwan, P. Sanposh, and N. Techajaroonjit, “Overview and control strategies of autonomous sailboats—A survey,” *Ocean Engineering*, vol. 281, p. 114 879, Aug. 2023, ISSN: 0029-8018. DOI: 10.1016/J.OCEANENG.2023.114879.

[34] H. Erckens, G. A. Büsser, C. Pradalier, and R. Y. Siegwart, “Avalon: Navigation strategy and trajectory following controller for an autonomous sailing vessel,” *IEEE Robotics and Automation Magazine*, vol. 17, no. 1, pp. 45–54, Mar. 2010, ISSN: 10709932. DOI: 10.1109/MRA.2010.935792.

[35] M. Zyczkowski and R. Szlapczynski, “Collision risk-informed weather routing for sailboats,” *Reliability Engineering & System Safety*, vol. 232, p. 109 015, Apr. 2023, ISSN: 0951-8320. DOI: 10.1016/J.RESS.2022.109015.

[36] H. Saoud, M. Hua, F. Plumet, and F. Ben Amar, “Routing and course control of an autonomous sailboat,” in *European Conference on Mobile Robots, ECMR 2015 - Proceedings*, 2015, ISBN: 9781467391634. DOI: 10.1109/ECMR.2015.7324218.

[37] D. Sidoti, K. R. Pattipati, and Y. Bar-Shalom, “Minimum Time Sailing Boat Path Algorithm,” *IEEE Journal of Oceanic Engineering*, vol. 48, no. 2, pp. 307–322, Apr. 2023, ISSN: 15581691. DOI: 10.1109/JOE.2022.3227985.

[38] A. da Silva Junior, D. Dos Santos, A. de Negreiros, J. Silva, and L. Gonçalves, “High-level path planning for an autonomous sailboat robot using Q-learning,” *Sensors (Switzerland)*, vol. 20, no. 6, 2020. DOI: 10.3390/s20061550.

[39] L. Liu, C. Wang, H. Gao, D. Shen, and Y. Liao, “High-Level Path Planning of Unmanned Sailboat for Sailing Championship and Innovative Education,” *Proceedings of 2022 IEEE International Conference on Unmanned Systems, ICUS 2022*, pp. 1557–1562, 2022. DOI: 10.1109/ICUS55513.2022.9986804.

[40] F. Plumet, H. Saoud, and M. D. Hua, “Line following for an autonomous sailboat using potential fields method,” *OCEANS 2013 MTS/IEEE Bergen: The Challenges of the Northern Dimension*, 2013. DOI: 10.1109/OCEANS-BERGEN.2013.6607961.

[41] Y. Deng, X. Zhang, and G. Zhang, “Line-of-Sight-Based Guidance and Adaptive Neural Path-Following Control for Sailboats,” *IEEE Journal of Oceanic Engineering*, vol. 45, no. 4, pp. 1177–1189, Oct. 2020, ISSN: 15581691. DOI: 10.1109/JOE.2019.2923502.

[42] C. Pêtrès, M. Romero-Ramirez, and F. Plumet, “Navigation with obstacle avoidance of an autonomous sailboat,” in *Field Robotics - Proceedings of the 14th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines, CLAWAR 2011*, 2012, pp. 86–93, ISBN: 9789814374279.

[43] F. Plumet, C. Pêtrès, M.-A. Romero-Ramirez, B. Gas, and S.-H. Ieng, “Toward an Autonomous Sailing Boat,” *IEEE Journal of Oceanic Engineering*, vol. 40, no. 2, pp. 397–407, 2015. DOI: 10.1109/JOE.2014.2321714.

[44] D. Ulysse, R. Niklas, and K. Jakob, “Energy Efficient Self-Steering Mechanism for an Autonomous Sailing Vessel,” in *OCEANS 2019 - Marseille*, Marseille, 2019, pp. 1–6, ISBN: 9781728114507. DOI: 10.1109/OCEANSE.2019.8867310.

[45] N. A. Cruz and J. C. Alves, “Navigation performance of an autonomous sailing robot,” *2014 Oceans - St. John’s, OCEANS 2014*, Jan. 2015. DOI: 10.1109/OCEANS.2014.7003227.

[46] S. Lemaire, Y. Cao, T. Kluyver, *et al.*, “Adaptive Probabilistic Tack Manoeuvre Decision for Sailing Vessels,” *Proceedings of the International Robotic Sailing Conference*, pp. 31–39, 2018.

[47] N. A. Cruz and J. C. Alves, “Auto-heading controller for an autonomous sailboat,” *OCEANS’10 IEEE Sydney, OCEANSSYD 2010*, 2010. DOI: 10.1109/OCEANSSYD.2010.5603882.

[48] R. Stelzer, T. Pröll, and R. I. John, “Fuzzy logic control system for autonomous sailboats,” *IEEE International Conference on Fuzzy Systems*, 2007, ISSN: 10987584. DOI: 10.1109/FUZZY.2007.4295347.

[49] G. Zhang, J. Li, C. Liu, and W. Zhang, “A robust fuzzy speed regulator for unmanned sailboat robot via the composite ILOS guidance,” *Nonlinear Dynamics*, vol. 110, no. 3, pp. 2465–2480, Nov. 2022, ISSN: 1573269X. DOI: 10.1007/S11071-022-07763-2/FIGURES/16.

[50] Y. Briere, F. L. Cardoso Ribeiro, and M. A. Vieira Rosa, “Design methodologies for the control of an unmanned sailing robot,” *IFAC Proceedings Volumes*, vol. 42, no. 18, pp. 58–65, Jan. 2009, ISSN: 1474-6670. DOI: 10.3182/20090916-3-BR-3001.0026.

[51] J. He, L. Xiao, and J. Jouffroy, “Towards heading control of an autonomous sailing platform through weight balancing,” in *IFAC Proceedings Volumes*, vol. 9, 2012, pp. 392–397, ISBN: 9783902823601. DOI: 10.3182/20120919-3-IT-2046.00067.

[52] Q. Sun, W. Qi, H. Liu, Z. Sun, T. L. Lam, and H. Qian, “OceanVoy: A hybrid energy planning system for autonomous sailboat,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 2481–2487, Oct. 2020, ISSN: 21530866. DOI: 10.1109/IROS45743.2020.9341591.

[53] M. L. Puterman, *Markov Decision Processes*. Wiley, Apr. 1994, ISBN: 9780471619772. DOI: 10.1002/9780470316887.

[54] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, “Reinforcement Learning for Control: Performance, Stability, and Deep Approximators,” *Annual Reviews in Control*, vol. 46, pp. 8–28, Jan. 2018, ISSN: 1367-5788. DOI: 10.1016/J.ARCONTROL.2018.09.005.

[55] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, “How to train your robot with deep reinforcement learning: lessons we have learned,” *International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, Apr. 2021, ISSN: 17413176. DOI: 10.1177/0278364920987859.

[56] R. Bellman, *Dynamic Programming*. Dover Publications, 1957, ISBN: 9780486428093.

[57] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning 1992 8:3*, vol. 8, no. 3, pp. 279–292, May 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698.

[58] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning 1992 8:3*, vol. 8, no. 3, pp. 229–256, May 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992696.

[59] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” *HotNets 2016 - Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pp. 50–56, Nov. 2016. DOI: 10.1145/3005745.3005750.

[60] Y. Yang, K. Caluwaerts, A. Iscen, T. Zhang, J. Tan, and V. Sindhwani, “Data Efficient Reinforcement Learning for Legged Robots,” in *Proceedings of the Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, vol. 100, PMLR, Jun. 2020, pp. 1–10.

[61] B. Recht, “A Tour of Reinforcement Learning: The View from Continuous Control,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 253–279, May 2019, ISSN: 25735144. DOI: 10.1146/ANNUREV-CONTROL-053018-023825.

[62] L. Smith, I. Kostrikov, and S. Levine, “Demonstrating a Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning,” *Robotics: Science and Systems*, 2023.

[63] T. Lesort, N. Díaz-Rodríguez, J.-F. Goudou, and D. Filliat, “State Representation Learning for Control: An Overview,” *Neural Networks*, vol. 108, pp. 379–392, Dec. 2018. DOI: 10.1016/j.neunet.2018.07.006.

[64] T. Suda and D. Nikovski, “Deep Reinforcement Learning for Optimal Sailing Upwind,” in *2022 International Joint Conference on Neural Networks (IJCNN)*, vol. 2022-July, IEEE, Jul. 2022, pp. 1–8, ISBN: 978-1-7281-8671-9. DOI: 10.1109/IJCNN55064.2022.9892369.

[65] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-Baselines3: Reliable Reinforcement Learning Implementations,” *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[66] A. Raffin, *RL Baselines3 Zoo*, https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[67] J. Hwangbo, J. Lee, A. Dosovitskiy, *et al.*, “Learning agile and dynamic motor skills for legged robots,” *Science Robotics*, vol. 4, no. 26, Jan. 2019, ISSN: 24709476. DOI: 10.1126/scirobotics.aau5872.

[68] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, “High-Dimensional Continuous Control Using Generalized Advantage Estimation,” *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, Jun. 2015.

[69] S. T. Havenstrøm, A. Rasheed, and O. San, "Deep Reinforcement Learning Controller for 3D Path Following and Collision Avoidance by Autonomous Underwater Vehicles," *Frontiers in Robotics and AI*, vol. 7, p. 566 037, Jan. 2021, ISSN: 22969144. DOI: 10.3389/FROBT.2020.566037/BIBTEX.

[70] D. Reda and H. Y. Ling, "Learning to Brachiate via Simplified Model Imitation," *SIGGRAPH Conference Proceedings*, vol. 1, 2022. DOI: 10.1145/3528233.3530728.

[71] N. Wang, Y. Wang, Y. Zhao, Y. Wang, and Z. Li, "Sim-to-Real: Mapless Navigation for USVs Using Deep Reinforcement Learning," *Journal of Marine Science and Engineering*, vol. 10, no. 7, 2022. DOI: 10.3390/jmse10070895.

[72] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," *IEEE International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012, ISSN: 21530858. DOI: 10.1109/IROS.2012.6386109.

[73] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking Deep Reinforcement Learning for Continuous Control," *33rd International Conference on Machine Learning, ICML 2016*, vol. 3, pp. 2001–2014, Apr. 2016. DOI: 10.5555/3045390.3045531.

[74] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3207–3214, Sep. 2018, ISSN: 2159-5399. DOI: 10.1609/AAAI.V32I1.11694.

[75] T. N. Larsen, H. Ø. Teigen, T. Laache, D. Varagnolo, and A. Rasheed, "Comparing Deep Reinforcement Learning Algorithms' Ability to Safely Navigate Challenging Waters," *Frontiers in Robotics and AI*, vol. 8, p. 738 113, Sep. 2021, ISSN: 22969144. DOI: 10.3389/FROBT.2021.738113.

[76] A. Laud and G. DeJong, "The influence of reward on the speed of reinforcement learning: An analysis of shaping," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, Washington DC, 2003.

[77] F. Muratore, M. Gienger, and J. Peters, "Assessing Transferability from Simulation to Reality for Reinforcement Learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 4, pp. 1172–1183, Apr. 2019, ISSN: 19393539. DOI: 10.1109/TPAMI.2019.2952353.

[78] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep Drone Racing: From Simulation to Reality with Domain Randomization," *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 1–14, Feb. 2020, ISSN: 19410468. DOI: 10.1109/TRO.2019.2942989.

[79] D. Wada, S. Araujo-Estrada, and S. Windsor, "Sim-to-Real Transfer for Fixed-Wing Uncrewed Aerial Vehicle: Pitch Control by High-Fidelity Modelling and Domain Randomization," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 735–11 742, Oct. 2022, ISSN: 23773766. DOI: 10.1109/LRA.2022.3205442.

[80] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005, ISBN: 0262201623.

[81] F. H. H. A. Quadvlieg and S. Rapuc, "A pragmatic method to simulate maneuvering in waves," in *SNAME Maritime Convention*, Oct. 2019.

[82] V. Ferrari, R. Tonelli, A. Kisjes, and R. Hallmann, "Manoeuvring experiments, mathematical model and sensitivity analysis for test-case ferry," *Trends in Maritime Technology and Engineering Volume 1*, pp. 327–335, Jun. 2022. DOI: 10.1201/9781003320272-36.

[83] *Legacy Products: NDI's 40-Year History and Transition*. [Online]. Available: https://www.ndigital.com/products/legacy-products/.

A

Supplementary Content

In the spirit of promoting transparency, reproducibility, and collaborative innovation, this project is open source. MARIN has a dedicated webpage which contains a comprehensive repository of data and resources related to this thesis:

https://www.marin.nl/en/research/artificial-intelligence-applications

Below is an overview of the contents available on this webpage:

- **data/**: Contains datasets from simulations and basin tests.
- **videos/**: Video files of all of the basin tests.
- **scripts/**: Post-processing and analysis scripts.
- **agents/**: Trained agents and their configuration files.

To support users in navigating these resources effectively, the webpage also provides instructions on the following:

- How to access and download the datasets, videos, scripts, and trained agents.
- Guidelines for using the post-processing and analysis scripts, including prerequisites for their execution and tips for troubleshooting common issues.
- Contact information for MARIN's research team for inquiries and questions.