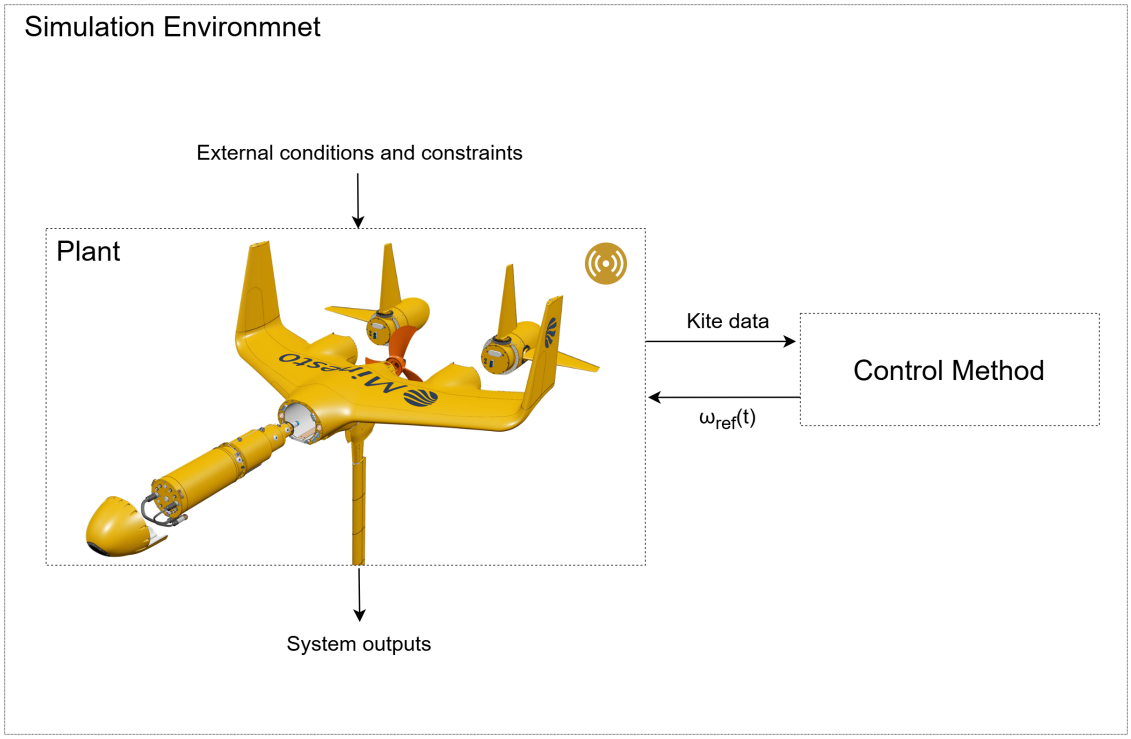
CHALMERS
UNIVERSITY OF TECHNOLOGY



Data Driven Turbine Control in Tidal Power Generation

Development and comparison of data driven turbine control algorithms in a simulated underwater kite environment

Master's thesis in Complex Adaptive Systems

Hampus Burenius
Rasmus Mårdberg

Department of Mechanics and Maritime Sciences

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

MASTER'S THESIS 2026

Data Driven Turbine Control in Tidal Power Generation

Development and comparison of data driven turbine control algorithms in a simulated underwater kite environment

Hampus Burenius
Rasmus Mårdberg

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
Gothenburg, Sweden 2026

Data Driven Turbine Control in Tidal Power Generation
Development and comparison of data driven turbine control algorithms in a simulated underwater kite environment
Hampus Burenius
Rasmus Mårdberg

Supervisor: Marco L. Della Vedova, Department of Mechanics and Maritime Sciences
Supervisor: Carl Larsson, Minesto AB
Supervisor: Magnus Fredriksson, Minesto AB
Examiner: Marco L. Della Vedova, Department of Mechanics and Maritime Sciences

Master's Thesis 2026
Department of Mechanics and Maritime Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: System diagram visualizing the kites interaction with the surrounding environment and the interaction with the developed turbine control algorithms.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Data Driven Turbine Control in Tidal Power Generation
Development and comparison of data driven turbine control algorithms in a simulated underwater kite environment
Hampus Burenius
Rasmus Mårdberg
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

Minesto's underwater kite extracts energy from ocean and tidal flows through a cross-flow motion and aims to convert as much energy as possible from the surrounding water-flow. To achieve this, designing a generator control signal that applies optimal torque on the turbine shaft is essential. This project aims to develop data-driven control systems for the kites generator that increases the power output relative to a baseline. ODE-based dynamical models of the kite and generator are derived to create a full simulation environment. The simulation is used to generate sensor data, as well as test and validate new control strategies. Two different approaches are investigated, a supervised predictive controller and a deep reinforcement learning controller where the control signal is generated by the learned policy. The supervised predictive controller approach involves creating a supervised dataset from the simulation and training a recurrent neural network to forecast future inflow water velocities. The knowledge of future inflow is then incorporated in the design of two new control methods, one that is built on the existing controller and another that is developed as a standalone control method. The deep reinforcement learning controller instead utilizes the simulation directly by iteratively stepping through the simulated environment and learning an optimal controller from the outcomes. This is achieved through the use of the state-of-the-art deep reinforcement learning algorithm Soft Actor-Critic. The solution also incorporates pre-training on generated data to validate a possible simulation-to-reality adaptation. Within the adopted simulation setting, the main finding is that predictive turbine control based on forecasted inflow can outperform a reactive baseline controller. The recurrent predictive controllers consistently improved generated power across unseen evaluation environments, indicating that short-term prediction and the periodicity of the kite motion are useful for control. The Soft Actor-Critic approach demonstrated learning capability but was more sensitive to reward design, partial observability, and tuning. Because the study relies on a simplified simulation model, the results should be interpreted as proof-of-concept and comparative evidence rather than as direct claims about real-system performance.

Keywords: TUSK, tidal power, underwater kite, power optimization, simulation, Soft Actor-Critic, Recurrent Neural Network.

Acknowledgements

We would like to thank Minesto and Magnus Fredriksson for outlining the project scope, driving the project forwards and creating a pleasant work environment. We would also like to thank our supervisor Carl Larsson at Minesto and our supervisor and examiner Marco L. Della Vedova for being supportive during the project. We appreciate the openness and trust in this project that allowed us to work independently and explore different solutions.

Hampus Burenius, Gothenburg, February 2026
Rasmus Mårdberg, Gothenburg, February 2026

Contents

List of Figures

List of Tables

1
Introduction

The transition towards renewable energy solutions has accelerated in the European Union's path to achieving climate neutrality by 2050 [8]. Among these renewable sources is the usage of tidal power, which is considered reliable for its predictability. Minesto's cross-flow underwater kites have shown promising potential for electricity generation from tidal streams and ocean currents, which makes efficient turbine control an important part of overall system performance.

1.1 Background

Minesto's sub-sea power plant kites aim to convert as much energy as possible from tidal and ocean currents. The wing uses hydrodynamic lift force and a steering control system to move the kite in a predetermined figure-eight trajectory, a so called Viviani's curve, pulling the turbine through the water at a relative water flow speed several times higher than the surrounding stream speed. To harvest as much energy as possible from the surrounding water flow, the onboard generator is controlled through a reference signal, such that it exerts a torque on the shaft that makes the turbine spin at an optimal speed. The kites relative velocity through the water and the blade tip speed of the turbine are considered highly valuable for maintaining an optimal tip-speed ratio (TSR) and maximize power generation. However, achieving this requires knowledge of the relative inflow velocity which is not directly measured.

To harvest as much energy as possible from the surrounding water flow, the onboard generator is controlled through a reference signal that determines the torque applied on the shaft and thereby the turbine speed. Since the extracted power depends strongly on the relation between turbine speed and relative inflow velocity, generator control has a direct influence on energy capture. Minesto currently uses a turbine control solution for this purpose. In this thesis, that solution is treated as a baseline reference, and the work investigates whether a data-driven controller that uses historical measurements and the periodic character of the kite motion can improve performance in a simulation setting.

The Dragon 4 (D4) kite, with a rated power of 100 kW, is used as the reference system for creating the simulation and developing the proposed controllers.

Prior work in the field mainly considers two kite models. Either a highly realistic free moving kite model, or a simplified path-locked model. In studies belonging to

the first group, highly realistic 6 degrees-of-freedom (DOF) kite models, often including multi-body tether models, are considered [45, 22, 21, 42, 33]. These models are then used to design path following kite control algorithms and contains heavy computations, resulting in slow simulation times. The second group includes work on simpler path following models, where the motion is constrained and internal tether dynamics are excluded. These models assume that the kite is locked to a predetermined, often a figure-eight path [9, 36, 23]. Modeling the kite-system like this moves the focus from spatial kite control to turbine control and allows for more efficient simulation, but at the cost of fidelity. The predetermined path models mentioned does often utilize the Viviani's curve due to its favorable mathematics, but at the cost of realism in the path obtained. However, a predetermined kite path has not previous been used to derive a dynamical kite-turbine model for simulation in time, but instead only for stationary point testing.

While there has been considerable contribution to wind turbine control, published work on turbine control for underwater kites is still relatively limited and has mainly focused on conventional strategies such as Maximum Power Point Tracking [23], which does not utilize stored data, nor kite-turbine models. The similar field of wind turbine control does however include contributions for turbine control algorithms utilizing data-driven algorithms. For example, different deep reinforcement learning algorithms have been used to optimize power production by controlling all control surfaces [37], and to reduce structural fatigue [46], both with promising results.

In this thesis, we propose a predetermined kite-path model with an improved path shape due to an extra path parameter. Namely, by exchanging the cylinder with a elliptical cylinder in the ordinary Viviani's curve, the cylinder radius is replaced by a minor and major ellipse axis, resulting in a more realistic kite path while keeping simulation times short. Furthermore, we propose two data-driven turbine controller solutions. The first one being a recurrent neural network (RNN) based predictive controller that incorporates the predicted relative velocity into a control method. The second is a deep reinforcement learning (DRL) approach where an agent learns turbine controls through interactions with the model to obtain an optimal turbine control policy.

1.2 Aim

The project aims to find a generator rotation speed reference signal that increases the power generated by the kite, despite constant water flow changes in the surrounding environment. By utilizing historical kite data, the project investigates whether predictive and reinforcement-learning-based control strategies can improve generator control relative to the baseline controller in a simulation setting, and what this reveals about predictive versus reactive control of periodic kite motion. A simplified model of the environment-kite-turbine system is to be created to allow a fair and realistic environment for controller comparison.

1.3 Limitations

The following limitations define the constraints of the project and scope. The limitations are divided into three subcategories: data-driven, control and simulation related limitations.

Data-driven

- The challenge in optimizing the power generation lies in the complex, nonlinear nature of the kites environment and how it behaves in the environment. Creating a model-based controller would require highly realistic mathematical models of the system. Deriving and simulating such a model is computationally heavy and out-of-scope for the project. A data-driven control approach is considered and adopted instead.
- The input feature space and sensor noise is determined by the sensors used in the real D4-kite system. The limited data features could impose restrictions on the controllers ability to find meaningful patterns and relations within the data.

Control method

- The current generator control unit (GCU), which regulates the generator speed given a reference signal, implies that the reference provided must take the GCU characteristics into account. A simplified model of the GCU is assumed to reduce system model and time complexity.

Simulation environment

- Creating an accurate, realistic, six degrees of freedom simulation environment of Minestos D4-kite is out of scope for the project given the limited access to kite-specific dynamics and kite-control system. Therefore, a simplified environment and kite dynamics model is considered sufficient for finding controller impact and performance.
- The simulation is only considering motion along a predetermined figure-eight flight path where the tether connecting the kite to the sea bed foundation is assumed to be a rigid body.
- The turbines effect, and in extension the controllers effect, on the the kite's motion is only considered in one dimension, tangent to the path.
- Results will only give insights in a simulation environment and not implemented on a real system as this project is considered an early stage development and investigation of potential generator control systems.

1.4 Specification of the issue being investigated

The importance of matching the turbine rotational speed ω_{turb} to the relative flow v_{rel} can be understood by the turbine equation,

$$P_{turb} = \eta \cdot \tfrac{1}{2}\rho A v_{rel}^3 C_p(\lambda), \tag{1.1}$$

where

$$\lambda = \frac{\text{turbine tip speed}}{\text{fluid speed}} = \frac{\omega_{turb}\, r_{turbine}}{v_{rel}} \tag{1.2}$$

is the tip speed ratio (TSR), η is the turbine efficiency, ρ is the fluid density, A is the turbine swept area and $C_p(\lambda)$ is the power coefficient. The equation reveals that the extractable power from the water, indirectly depends on λ through the power coefficient. The C_p curve is derived from computational fluid dynamics (CFD) modeling and the λ that corresponds to C_p optimum is unique for every turbine design. Hence, to harvest as much energy as possible from the surrounding water flow, it is of great importance to match the rotational speed of the turbine to the velocity of the water flow relative to the kite. If the turbine rotates too fast, the surrounding water flow will collide with the turbine blade putting additional drag on the turbine. If on the other hand, the turbine spins too slow, water flow will pass through the turbine without effective extraction [4]. Although the turbine power output has an optima for a specific λ, the project does not isolate the turbine to itself. Instead, the developed control reference signals seek to maximize power generation looking at the kite system as a whole to find the global optima.

To be able to obtain a data-driven generator reference control capable of improving the power generation, sensor measurement data from Minestos D4 kite is used. The data features are measured and stored in time-step intervals and consists of 16 measurements in total. Due to confidentiality policies, individual sensor signals are not described in detail. The measurements from the IMU, that is the accelerometer, gyroscope and magnetometer each have three values, one for each kite relative axis. Magnetometer measurements were excluded due to generator interference.

The problems to be answered in this paper can be narrowed down to the following research questions:

- Is a limited feature space sufficient for data-driven models to learn meaningful relationships?
- What is the significance of the periodic motion of the kite?
- Can a simplified system model be created with sufficient faithfulness to the real kite system?
- Are the models applicable to a real-world setting?
- How much can the power generation be increased compared to using the current systems baseline controller, when applying the new control for the reference signal?
- Is tracking of the turbine optimal TSR the best control objective for maximizing power output of the coupled kite system?

1.5 Thesis outline

Chapter 2 explains the theoretical conceptual parts on which the developed data-driven controllers are based on. This serves as the fundamental building block for the control method chapter. Chapter 3 explains the system modeling and how the

simulation environment was created. Chapter 4 contains the main methodology in developing the controllers, followed by the results obtained in Chapter 5, and a discussion on the findings in Chapter 6. Lastly, Chapter 7 summarize the thesis and indicates the direction of future work.

2
Theory

There is no universal framework for creating a data-driven control method, as the performance of a model strongly depends on the system at hand, its characteristics, constraints and the available data. Given the exploratory nature of the project seeking an improved generator control method, two methods were investigated as potential candidates. The first approach, the **supervised predictive controller**, aims to use an RNN to forecast the relative inflow velocity v_{rel} and use this knowledge inside a structured control law to compute ω_{ref}. The second approach, **soft actor-critic controller** interacts with the simulated environment to learn optimal control policies through trial and error by mapping data points to control signals. This chapter outlines the main theoretical components of the two controllers and serves as a conceptual foundation for the control methods developed in Chapter 4.

2.1 Supervised predictive controller

The supervised predictive controller method was developed under the assumption that knowledge of the relative inflow velocity v_{rel} would provide the key information needed for the generator control to give a near optimal reference signal. Not only does the turbines power optimum depend on v_{rel} as described in Section 1.4, it is also tied to the periodic motion of the kite. The supervised predictive controller method figures out how one can predict v_{rel} from the collected kite data and incorporate this knowledge in a control reference signal. This section explains the theory behind the RNN deep learning model used to process the kite time-dependent sequential data.

From a control-theoretic perspective, the supervised predictive controller can be viewed as a data-driven predictive control strategy: future inflow estimates are used to adjust the generator reference before the disturbance is fully reflected in the measured output. This places the approach closer to predictive control than to a purely reactive feedback law. It is not formulated as a full Model Predictive Control (MPC) problem with an explicit online optimization over a constrained horizon, but it shares the central MPC idea that predictions of future system behavior can be used to improve present control actions.

2.1.1 Recurrent neural networks

The recurrent feedback mechanism that presented itself in the evolution of artificial neural networks, and which forms the foundation of modern recurrent network archi-

tectures, originates from neuroscience and models of neuron activity. It captures the networks ability to maintain and process information from past states through feedback recurrent connections, introducing temporal dependencies and internal memory. One of the first network models to incorporate this was the McCulloch-Pitts neuron model which introduced nets with circles [24]. Further developments, including Frank Rosenblatts perceptron architecture and later the Hopfield network in 1982, extended the recurrent network ideas and models [34, 15].

The early developments to process temporal sequences and predict elements in a sequence include the Elman networks, or simple recurrent networks (SRN). The network introduced a three layer network with context units, which stored hidden layer activations by directly copying them and was thereafter fed back to the hidden unit. This way, the network could capture temporal sequences by updating the hidden unit with its previous state [7].

2.1.1.1 Modern recurrent neural networks

Although modern RNN architectures differ from early models, the fundamental property of recurrence through feedback connections is the same. Modern units with this property are referred to as recurrent units and are the building blocks of modern RNNs. Modern RNN architectures come in many forms. A standard structure can be described as consisting of the input, hidden and output vector x_t, h_t, y_t along with the network parameters θ. The network maps an input to an output through the hidden layer which has a feedback mechanism that allows for the storing of past information. A common representation of the RNN architecture is obtained by unfolding the network through time, as shown in Figure 2.1, where the hidden unit update is,

$$h_t = g(W_h h_{t-1} + W_x x_t + b) \tag{2.1}$$

and $g(\cdot)$ denotes the activation function.

Unlike deep feed-forward neural networks which consists of unique hidden neurons in hidden layers, RNNs reuse the same hidden units across time-steps. The hidden state h_t is propagated through time as the RNN is unfolding. Time t here should be interpreted as a sequence index rather than physical time and is used to show that the RNN is a dynamical system with a state-dependent evolution. While the RNN developed in this project is time-dependent, RNNs can model any sequential data such as words in sentences or symbols in a sequence.

2.1.1.2 The vanishing gradient problem

The early RNN models showed great promise, but suffered from the vanishing gradient problem which restricted them from learning long-term dependencies. The vanishing gradient problem is a consequence of unfolding the network with the standard hidden unit model described in Equation (2.1). Updating the network parameters θ through back propagation through time (BPTT), leads to a shrinking of the gradient

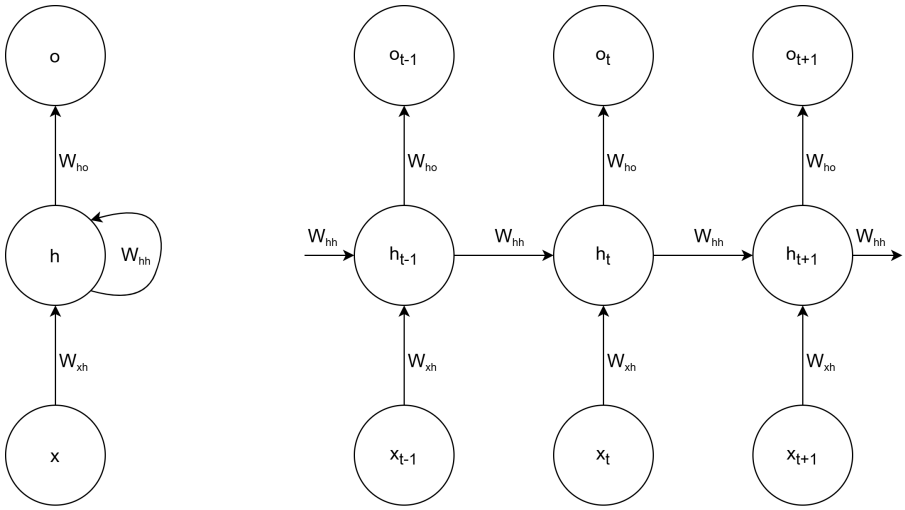


Figure 2.1: Compressed (left) and unfolded (right) standard modern RNN.

magnitude as the chain-rule update multiplications increase. Long-term dependencies are thereby difficult to learn, as the gradient has exponentially decreased by the time it reaches a distant dependent hidden state. As such, the standard unit had to be updated to allow the RNNs to learn long-term dependencies and resulted in what is know as a gated unit.

The first breakthrough was the Long Short-Term Memory (LSTM) hidden unit which was introduced by Hochreiter and Schmidhuber in their 1997 paper "Long Short-Term Memory" and introduced a gated memory cell that allowed information to persist and control what past information to erase and keep [14]. The Gated Recurrent Unit (GRU), originally introduced in 2014, was developed to improve both the memory capacity and ease of training [6]. The GRU hidden unit architecture was a simplification of the LSTM, reducing the number of gates and cell states, which meant less computational cost and faster training.

2.1.1.3 Gated recurrent unit

The GRU consists of a reset gate r_t and a update gate z_t. The gated design allows two key mechanisms to happen. The reset gate allows the unit to determine how much of the past information the hidden state should forget. Unlike a natural RNN hidden unit, the hidden state update does not only depend on the weight matrix connected to it. The scalar multiplication with r_t, determined by the sigmoid function, enables the unit update to forget past information stored. The resulting hidden state, often referred to as the candidate hidden state n_t, is sent as a proposing update for the hidden state.

Secondly, the update gate z_t gives the hidden unit the ability to control how much of the candidate hidden state to include. This very ability is what solves the vanishing gradient problem, as it lets the gate decide how much of the past hidden state can be copied over directly. The GRU architecture is seen in Figure 2.2 and the corresponding state update equations can be described as:

$$
\begin{aligned}
r_t &= \sigma\left(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}\right) \\
z_t &= \sigma\left(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}\right) \\
n_t &= \tanh\left(W_{in}x_t + b_{in} + r_t \odot \left(W_{hn}h_{t-1} + b_{hn}\right)\right) \\
h_t &= (1 - z_t) \odot n_t + z_t \odot h_{t-1}
\end{aligned}
\tag{2.2}
$$

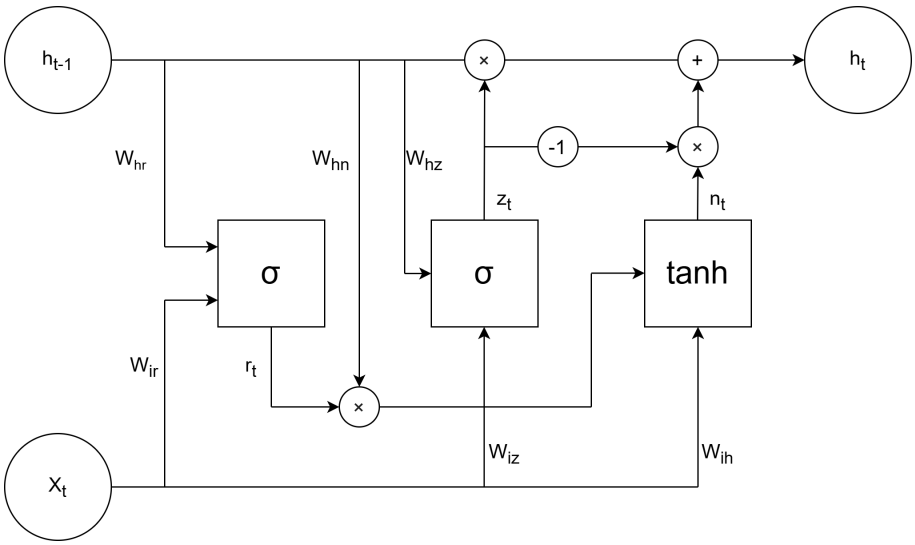


Figure 2.2: The GRU hidden unit architecture

2.2 Reinforcement learning

The information in this section is mainly gathered from [38], otherwise another citation is given. However, the equations presented in this section has been slightly modified to better align with those used in SAC literature, but has the same function.

Reinforcement learning (RL) can be described as an agent interacting with an environment and from the interactions continuously improving its abilities. It is at its core the theory of learning by mapping situations to actions, and in this way maximizing a reward. This process can be seen as a Markov Decision Process (MDP) where given a situation, more commonly named state s_t or observation o_t, the agent takes an action a_t which results in a reward r_t and a transition to the next state s_{t+1}. The transition distribution from one state to another is fully described as $p(s_{t+1}|a_t, s_t)$. The goal of the agent is to choose an action that maximizes the future cumulative reward given the current state. This is achieved by learning a policy $\pi(a|s)$ that given a state determines the optimal action. To further describe the theory behind RL the following terms needs to be known in further detail.

- **State** s_t is the state of the environment, in which the agent acts, at time-step t. State can be exchanged with observation o_t, where the observation is a subset of the state. Meaning that the observation does not contain all information about the state. If s_t is equal to o_t the environment is said to be fully observable, otherwise it is partially observable. In the case of a partially observable environment, the observation does not include all information of the past agent-environment interaction and the Markov property does not hold. Hence, the transition is not fully described as $p(o_{t+1}|a_t, o_t)$ [26, 38].

- **Action** a_t is the way the agent interacts with the environment and moves from one state to another. The action space can be both continuous and discrete and vary in number of dimensions. This means that one agent can be trained to control many actions simultaneously, for example aileron, elevator, rudder and engine thrusts of an airplane [43].

- **Reward** r_t is a numerical signal given to the agent each timestep depending on how it performs. The reward might be positive for good progress or performance or negative for the opposite. The reward should be shaped so that high reward is connected to good performance and vice versa. As mentioned, the agent tries to optimize the cumulative reward, that is the reward for all future time-steps. For this purpose, the discounted return is commonly used,

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.3}$$

  where γ is the discount on future rewards. When $0 < \gamma < 1$, future rewards are discounted, so rewards received later are worth less than rewards received sooner. Since future rewards are often unknown, value functions approximating the combined future value of a state are used. These are introduced below.

- **Episode.** Training of RL agents is usually done on episodic tasks, where there is a clear start and end to each episode. The start of an episode is usually a predetermined state or a randomized initial state. The end is defined by a time-step limit and termination conditions, which is further discussed in the implementation in Chapter 4.

- **Policy** controls what action the agent takes in a given state. The policy is state dependent and for each state the policy gives a probability for each possible action. The policy is mathematically written as $\pi(a|s)$, that is the probability of action a given state s. This policy can also be deterministic when used in some algorithms.

- **Value functions** are used and estimated by RL algorithms to map states to values, determining how good a state is. The value of a state is measured as the future rewards expected (often expected return) from the current state if the agent takes actions according to a policy $\pi(a|s)$,

$$V^{\pi}(s) = \underset{(s_t)\sim\pi}{E}[R_t \mid s_t = s] = \underset{(s_t)\sim\pi}{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s\right], \text{ for all } s. \tag{2.4}$$

  Furthermore, value functions mapping state-action pairs to a value, so called action-value functions, are also commonly used. These gives the value for taking action a from state s and then following the policy $\pi(a|s)$,

$$Q^{\pi}(s,a) = \underset{(s_t,a_t)\sim\pi}{E}[R_t \mid s_t = s, a_t = a] = \underset{(s_t,a_t)\sim\pi}{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a\right], \text{ for all } s,\ a. \tag{2.5}$$

  From now on, $V^{\pi}(s)$ will be called value function and $Q^{\pi}(s,a)$ action-value function (often also referred to as state-action value function or Q-function in other literature). This nomenclature often differs in RL literature.

- **Optimal value function.** Since the value functions discussed above allows different policies to be compared, it is clear that there exists a **Optimal policy** $\pi^*(a|s)$ that has larger expected return (value function) then all other policies in all states. The existence of an optimal policy also means that there exists optimal value functions, that is the expected return from a given state following the optimal policy,

$$V^*(s) = \max_{\pi} V^{\pi}(s). \tag{2.6}$$

  Similarly, for the optimal action-value function,

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{2.7}$$

  Where the policy π that maximizes V^{π} and Q^{π} is the optimal policy.

- **Bellman equations.** While there exists RL algorithms that estimates value functions by interacting with the environment and collecting samples of actual returns from states (for example Monte Carlo reinforcement learning), most algorithms instead utilize that the value functions can be written as Bellman equations. This means that there exists a recursive relation between the value of the current state and that of its succeeding states,

$$V^{\pi}(s_t) = \underset{(r_t, s_{t+1})\sim\pi}{E}\left[r_t + \gamma V^{\pi}(s_{t+1})\right], \tag{2.8}$$

$$Q^{\pi}(s_t, a_t) = \underset{(r_t)\sim\pi}{E}\left[r_t + \gamma \underset{(s_{t+1}, a_{t+1})\sim\pi}{E}\left[Q^{\pi}(s_{t+1}, a_{t+1})\right]\right] \tag{2.9}$$

$$= \underset{(r_t)\sim\pi}{E}\left[r_t + \gamma \underset{(s_{t+1})\sim\pi}{E}\left[V^{\pi}(s_{t+1})\right]\right], \tag{2.10}$$

$$V^{*}(s_t) = \underset{a}{max} \underset{(r_t, s_{t+1})\sim\pi^{*}}{E}\left[r_t + \gamma V^{*}(s_{t+1})\right], \tag{2.11}$$

$$Q^{*}(s_t, a_t) = \underset{(s_t, a_t)\sim\pi^{*}}{E}\left[r_t + \gamma Q^{*}(s_{t+1}, a_{t+1})\right]. \tag{2.12}$$

  For example, in Equation (2.8) the value function for state s_t is the expected value, following the policy π, of the current reward r_t summed with the discounted value function for the next state s_{t+1}. The relation between the action-value and the value function seen in Equation (2.10), is important in the SAC implementation and will be seen again below.

- **The RL objective** is, as mentioned, to maximize the cumulative reward. The mathematical objective is usually to maximize the expected reward, that is, to maximize the value function [30],

$$J(\pi) = \underset{(r_t)\sim\pi}{E}\left[R_t\right] = V^{\pi}(s_t). \tag{2.13}$$

  To maximize this objective, RL algorithms tries to find the optimal policy,

$$\pi^{*}(a|s) = \underset{\pi}{argmax}\ J(\pi). \tag{2.14}$$

Traditional RL algorithms like Q-learning and SARSA, can for small state spaces estimate value functions for each state, or state-action pair, in an array or tabular layout. However, as the state and action spaces grow, memory limitations make this approach infeasible. Furthermore, the computations needed to be carried out each time-step to solve the Bellman equations for the optimal action is very large. One way to pass this obstacle is to approximate value functions and policies with deep neural networks to remove the need for large tabular representation, so called Deep Reinforcement Learning (DRL) [30].

2.3 Deep Reinforcement Learning

This section is based on the book on Deep Reinforcement Learning by Aske Plaat [30]. The field of DRL aims to explore the possibilities to use the theory of ordinary reinforcement learning in a setting that, prior to DRL, was not feasible. That is, learning in high-dimensional environments where states and/or action vectors are large and possibly continuous. This is made possible by introducing deep learning into the field of reinforcement learning, where value functions and policies are approximated with neural networks, making the need for tabularized state-value mappings obsolete and allowing generalized agent behavior outside the training region.

There are many different types of DRL algorithms developed that have different functionality and use cases. In this work, Soft Actor-Critic (SAC) is used and only SAC and its components will be described. DRL algorithms can be divided into two main groups, model-based and model-free algorithms. SAC and the most commonly used algorithms are of the model-free group. Furthermore, the model-free group can be divided into three sub-groups based on what the algorithms try to learn [1, 30].

- **Value-based** or often called Q-learning contains a group of algorithms that learns to approximate the optimal action-value functions $Q^*(s, a)$. The approximate action-value function $Q_\theta(s, a)$ is a deep neural network with parameters (weights) θ. With an approximated optimal action-value function, value-based algorithms directly take the optimal action at each state instead of finding the optimal policy from the RL objective, Equation (2.14),

$$a(s) = \underset{a}{argmax}\ Q_\theta(s, a). \tag{2.15}$$

  That is, the value of each possible action is compared to each other and the action with the largest value is chosen. This means that value-based algorithms can not have a continuous action space, only the state space can be continuous. These algorithms often use replay buffers, or experience replays, containing many old state, action, reward transition tuples. These are used together with newly acquired tuples to train the algorithm by changing the approximated action-value function, which counteracts that algorithms forget earlier training. Since value-based algorithms uses data (transition tuples) that was not obtained using the current policy (the current approximation of the optimal action-value function) these are usually off-policy methods.

- **Policy-based** are algorithms that given a randomized parametrized policy function $\pi_\phi(a|s)$ (usually represented as a neural network), continuously optimize the policy to approximate the optimal policy $\pi^*(a|s)$. Where, in contrast to value-based, the action for a given state lies in a continuous action space. The optimization of the parameters ϕ is done by sampling one or more transitions using the current policy, calculating the expected return of these and through gradient ascent pushing the approximated policy closer to the optimal

policy. Different policy-based algorithms use different amount of samples, but in contrast to value-based algorithms, the samples are all retrieved from the current policy. Therefore, these algorithms are on-policy methods.

- **Mixed algorithms** is the third type of model-free DRL algorithms. As the name suggests these are algorithms that incorporate methods from both the value- and policy-based algorithms. The group of mixed algorithms is in some texts included in the policy-based sub-group since they base their actions on a policy which is simultaneously updated. This is the group where the Soft Actor-Critic algorithm lives and it is further described in the following section.

2.4 Soft Actor-Critic

The following explanation of SAC is not complete and does not cover intermediate steps. For a deeper understanding, the reader can turn to the paper [1] and the original SAC papers [11, 12, 13]. This section about SAC is based on the three original SAC papers [11, 12, 13], citations is only given if another source is used.

SAC is one of many actor-critic type algorithms. Common for these types of algorithms is that they have parametrized policies and value functions. The actor part determines, and continuously improves, the policy $\pi_\theta(a|s)$, and consequently the action that the agent takes. This is done in a policy-based manner. The critic part instead continuously approximate a value function, for example $Q_\phi(s, a)$, towards the optimal value function in a value-based manner. This means that an actor-critic algorithm incorporates methods from both the value-based and policy-based subgroups [30].

2.4.1 Entropy

There are different methods to ensure proficient exploration versus exploitation in DRL algorithms. If the algorithm uses a deterministic policy, like purely value-based algorithms, noise can simply be added to the action. By tuning the distribution, variance and bias of the noise, the degree of exploration can be manipulated. However, in the case of SAC, where the policy is stochastic, the noise is already included in the random nature of the policy. But, when the policy is getting more and more optimized, its distribution gets thinner, hence reducing exploration. This is counteracted in SAC by introducing an entropy term [30, 12],

$$H[\pi_\theta(a|s)] = E[-log\ \pi_\theta(a|s)]. \tag{2.16}$$

The entropy term is a measure of how random the policy is. A more uniform policy gives larger entropy while an almost deterministic policy has entropy close to zero.

2.4.2 SAC objective

To give SAC its enhanced exploration versus exploitation properties, the original RL objective, Equation (2.13), is extended to include the entropy term. This means that SAC not only optimizes for maximal expected reward, but also for maximal randomness in the trained policy, which results in the maximum entropy objective,

$$J(\pi) = \sum_{t}^{\infty} \underset{(s_t,a_t,r_t)\sim\pi}{E} [r_t + \alpha H[\pi(a_t|s_t)]]. \tag{2.17}$$

That is, SAC tries to find the optimal policy,

$$\pi^* = \underset{\pi}{argmax} \sum_{t} \underset{(s_t,a_t,r_t)\sim\pi}{E} [r_t + \alpha H[\pi(a_t|s_t)]]. \tag{2.18}$$

Here, α is the temperature parameter which manages the relation between exploration and exploitation in SAC [13].

2.4.3 Replay Buffer

As mentioned above, value-based, off-policy algorithms usually incorporate replay buffers (replay pool in SAC paper [13]). The replay buffer is used to store prior transitions that the agent has seen. The transitions are stored as tuples containing the current state, action, reward and next state (usually also the termination boolean). That is, $(s_t, a_t, r_t, s_{t+1}) \cup D \rightarrow D$ where D is the replay buffer. The buffer is extended with a new tuple at each step the agent takes and has a maximum length, after which is achieved, a first in first out strategy is adopted [27]. There are different types of replay buffers that has different functionality and use cases, which are further discussed in Chapter 4.

2.4.4 Critic

The critic used in SAC is modeled as a deep neural network with parameters θ, meaning that the critic is a network representation of the value function $Q_\theta(s_t, a_t)$, which given a state-action pair returns the corresponding value. The new maximum entropy objective gives raise to an extended version of the value function in Equation (2.13), which now includes the entropy term,

$$V^\pi(s_t) = \underset{(s,a,r)\sim\pi}{E} \left[\sum_{k=t}^{\infty} \gamma^k (r_k + \alpha H[\pi(a_k|s_k)]) \right]. \tag{2.19}$$

This value function can be used to re-derive the action-value function in Equation (2.10),

$$Q^\pi(s_t, a_t) = r_t + \gamma \underset{\substack{(a_{t+1})\sim\pi \\ (s_{t+1})\sim D}}{E} [Q^\pi(s_{t+1}, a_{t+1}) - \alpha \, log \, \pi(a_{t+1}, s_{t+1})], \tag{2.20}$$

which is approximated from states s_{t+1} from the replay buffer and actions a_{t+1} from the current policy $\pi_\phi(a_{t+1}|s_{t+1})$.

To optimize the critic's value function, a mean square error loss function between the parameterized action-value function $Q_\theta(s_t, a_t)$ and an action-value function approximation bootstrapped from the next state $t+1$ according to Equation (2.20) is used,

$$J_Q(\theta) = \underset{\theta}{min}\ E\left[\frac{1}{2}(Q_\theta(s_t, a_t) - Q^\pi(s_t, a_t))^2\right] = \tag{2.21}$$

$$\underset{\theta}{min} \underset{\substack{(s_t, a_t, r_t, s_{t+1}) \sim D \\ (a_{t+1}) \sim \pi}}{E} \left[\frac{1}{2}(Q_\theta(s_t, a_t) - (r_t + \gamma(Q_{\theta_{target}}(s_{t+1}, a_{t+1}) - \alpha\, log(\pi_\phi(a_{t+1}|s_{t+1}))))^2\right]. \tag{2.22}$$

In the loss function above, $Q_{\theta_{target}}(s_t, a_t)$ is a target action-value function. The parameters θ_{target} of the target slowly follows the parameters of $Q_\theta(s_t, a_t)$, meaning that the target action-value function can be seen as an earlier copy of the current action-value function. This is done to ensure stability during training. Furthermore, to ensure stable training, SAC simultaneously trains two separate action-value functions using the loss function in Equation (2.22), both having separate target networks. In the loss function, Equation (2.22), above, SAC uses the smallest (least value for the (s_t, a_t) pair) $Q_{\theta_{target}}(s_t, a_t)$ of the two when training the action-value function networks. This loss function is then used with stochastic gradient decent to improve the two parametrized action-value function $Q_\theta(s_t, a_t)$.

2.4.5 Actor

The actor of the SAC algorithm is modeled as a deep neural network outputting a mean and covariance, which are used to form the stochastic policy,

$$\pi_\phi(a_t|s_t) = \mu_\phi(s_t) + \sigma_\phi(s_t)\mathcal{N}(0, 1). \tag{2.23}$$

To train this policy, the SAC objective, see Equation (2.17), could initially be thought to be maximized directly,

$$J(\pi_\phi) = \underset{\phi}{max} \underset{(s_t, a_t) \sim \pi}{E} [Q_\theta(s_t, a_t) - \alpha\, log\, \pi_\phi(a_t|s_t)]. \tag{2.24}$$

However, this is not possible due to the policy being stochastic (objective can not be differentiated). Therefore, the author tries to minimize the Kullback–Leibler (KL) divergence instead for a similar result,

$$J_\pi(\phi) = \underset{\phi}{min}\ D_{KL}\left(\pi_\phi(a_t|s_t)\ ||\ \frac{exp\ (\frac{1}{\alpha}Q_\theta\ (s_t, a_t))}{Z(s_t)}\right), \tag{2.25}$$

where $Z(s_t)$ is needed to normalize the numerator to be a probability density function, that is the area sums to unity. The reason behind the choice of objective is that minimizing the Kullback-Leibler (KL) divergence between the current policy and the new action-value function will achieve a policy with higher expected return and/or higher entropy. This can be seen when extending the new objective,

$$J_\pi(\phi) = \min_{\phi} \underset{\substack{(s_t) \sim D \\ (a_t) \sim \pi}}{E} \left[log\, Z(s_t) + \, log\, \pi_\phi(a_t|s_t) - \frac{1}{\alpha} Q_\theta(s_t, a_t) \right], \tag{2.26}$$

which is identical to the SAC objective in Equation (2.24) if the normalizing distribution $Z(s_t)$ is ignored and the function multiplied with α. However, to use this objective function the dependence on the parameters ϕ must be removed from the expectation. That is, the expectation over actions sampled from the policy $(a_t) \sim \pi_\phi$. This is done through the reparameterization trick where actions are instead sampled from a policy with deterministic mean μ_ϕ and added independent noise $\epsilon \sim N(0, 1)$, scaled with a deterministic variance σ_ϕ [1]. This policy is presented above in Equation (2.23). Here it is rewritten as deterministic but depends on the independent noise to match what is seen in SAC literature,

$$a_t = f_\phi(\epsilon_t; s_t) = \mu_\phi(s_t) + \sigma_\phi(s_t)\, \epsilon_t. \tag{2.27}$$

With this inserted into the objective, Equation (2.26), the complete policy objective is received,

$$J_\pi(\phi) = \min_{\phi} \underset{\substack{(s_t) \sim D \\ (\epsilon_t) \sim \mathcal{N}(0,1)}}{E} \left[\alpha\, log\, \pi_\phi(f_\phi(\epsilon_t; s_t)|s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t)) \right]. \tag{2.28}$$

This policy objective function is then used to perform stochastic gradient decent on the policy parameters ϕ. As mentioned in the actor section, SAC trains two action-value functions, in the objective from Equation (2.28) the smaller of the two action-value function $Q_\theta(s_t, a_t)$ is used to ensure stability.

2.4.6 Entropy parameter setting

The entropy parameter α determines the ratio between exploration and exploitation by increasing or decreasing the importance of the entropy term in the objective function. However, with SAC, this parameter can be either constant or automatically adapted. The automatically adjusted method adds another optimization problem where the average entropy is managed to lay above a minimum entropy and to yield the highest reward. Neither the objective function for the entropy parameter, nor its derivation is shown in this work. The reader is referred to the original SAC paper on this topic [13].

2.4.7 Algorithm

Now that all major functions of the SAC algorithm has been covered, their usage can be described. After initializing all five neural networks (one policy network, two action-value functions and two corresponding target networks) and the replay buffer, the agent can chose its first action based on the initial state. This first action is essentially random since the policy network is random. The action causes the agent to transition to the next state in the environment resulting in a reward. The state, action, reward, next state tuple is appended to the empty buffer.

Then, the action-value function networks can be updated with the loss function described by Equation (2.22), using the data in the buffer. After the action-value networks has been updated, the policy network is updated with the policy objective function, see Equation (2.28), and the data in the buffer. Following that, the entropy parameter is updated, assuming that adaptive entropy parameters is used. Finally, the target action-value function networks are updated through polyak averaging. The gradient decent/accent updates uses mini-batches taken from the buffer, meaning that a randomized subset of old and new transition tuples are used to update the networks. This cycle is repeated for a set amount of steps.

When the training is completed, the agent is comprised of three useful networks, the policy (actor) and the two action-value networks (critic). When using the network in a live, non-training setting, the actor is simply used alone to determine the optimal action in the given state. It is however possible to let the agent continuously train during normal operation.

3
System Modeling

This chapter explains the system modeling workflow. The fixed kite flight path is derived to then be incorporated into the kite kinematics. Kite reference frames are described and used to calculate forces acting on the kite-system. Dynamical models are obtained for the coupled kite-generator system and sensor reading calculations are performed. Lastly, the water current function used is described to create a fully solvable state space model.

3.1 Simulation System modeling

This section describes the mathematical modeling of the kites simulation environment. The kite kinematics are derived from the governing geometrical equations of the path and combined with external forces, to create a dynamical model of the kites position and velocity along the path. A state-space model is introduced, representing the whole system dynamics and forming the framework for the simulation and generator control design. An overview of the system components and interactions can be seen in Figure 3.1.

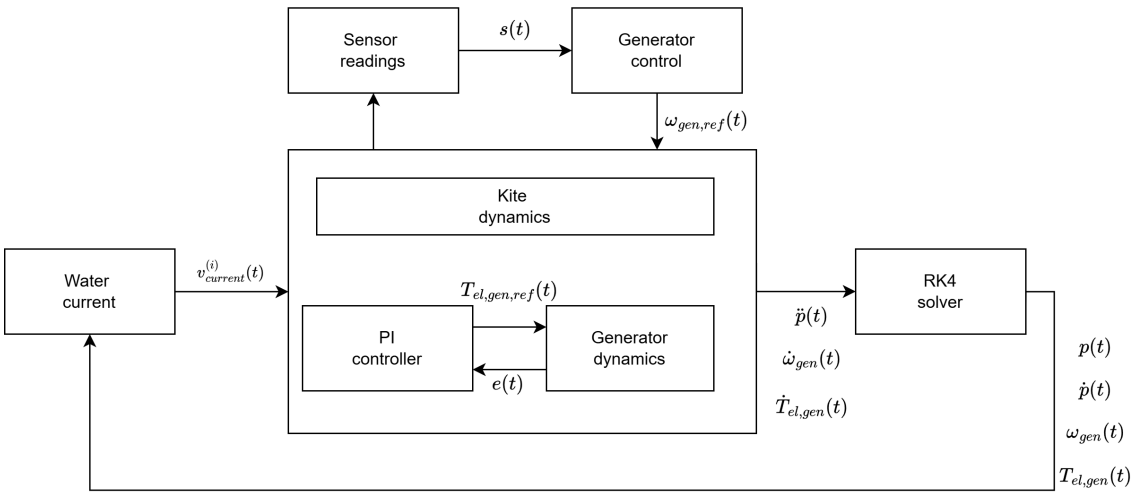


Figure 3.1: Block diagram overview of full system model.

3.1.1 Kite figure-eight path

The Viviani's curve, a figure-eight shaped curve defined as the intersection between a sphere and a cylinder tangent to it, forms the basis of the kite flight path. While

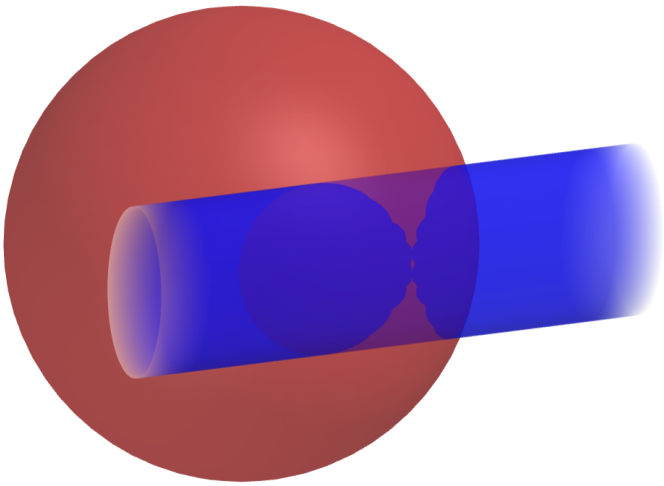


Figure 3.2: The intersection curve between the red sphere and blue elliptic cylinder describes the assumed kite path.

other related work has adopted the Viviani's curve, see [23, 36, 9], with the assumption that it describes a realistic kite path, this paper modifies the standard Viviani's curve to better describe the motion of the real kite. This modified path curve is described in Section 3.1.1.2 and a descriptive illustration is presented in Figure 3.2. The need for a more realistic path was concluded by comparing the simulation to real-world kite data.

3.1.1.1 Cylindrical Viviani's curve

The cylindrical Viviani's curve can be used to describe the kite path by setting the radius of the sphere R equal to the tether length. The sphere is centered at the origin and the cylinder is tangent to it with radius r. The equation for a sphere and cylinder in cartesian coordinates is,

$$x^2 + y^2 + z^2 = R^2, \tag{3.1}$$

$$(x - x_0)^2 + (z - z_0)^2 = r^2. \tag{3.2}$$

Equation (3.1) represents the sphere centered at the origin, and Equation (3.2) represents a cylinder parallel to the y-axis with radius r, centered at $(x_0, z_0) = (R-r, 0)$ and tangent to the sphere at $(R, 0, 0)$ for $r \leq R$.

The coordinates are parameterized and expressed as a function of an independent position variable p. Looking at the x-z plane, a circle with radius r can be expressed as,

$$\begin{cases} x(p) = (R - r) + r\cos(p), \\ z(p) = r\sin(p), \end{cases} \qquad p \in [0, 2\pi]. \tag{3.3}$$

The equation for y is obtained by inserting Equation (3.3) into Equation (3.1),

$$y(p) = 2\sqrt{r(R-r)}\,\sin\left(\tfrac{p}{2}\right). \tag{3.4}$$

The position equations as a function of p becomes,

$$\begin{cases} x(p) = (R-r) + r\cos(p), \\ y(p) = 2\sqrt{r(R-r)}\,\sin\left(\tfrac{p}{2}\right), & p \in [0, 4\pi]. \\ z(p) = r\sin(p), \end{cases} \tag{3.5}$$

Note that the $\frac{p}{2}$ term in $y(p)$ makes one period for the kite be 4π, meaning that for every full figure-eight traversal, two rotations around the cylinder are performed.

3.1.1.2 Elliptic Viviani's curve

In contras to the ordinary Viviani's curve, where a cylinder intersects a sphere, the cylinder is now exchanged for an elliptic cylinder with two semi-axis, a and b. The equation for the sphere stays the same while the cylinder is instead described as,

$$\frac{(x-x_0)^2}{a^2} + \frac{(z-z_0)^2}{b^2} = 1. \tag{3.6}$$

The elliptic cylinder is parallel to the y-axis and has it center at $(x_0, z_0) = (R-a,\ 0)$, to ensure that the elliptic cylinder tangents the sphere at $(R, 0, 0)$.

In a similar fashion to the cylindrical case, the parameterized p dependent on x and z is,

$$\begin{cases} x(p) = (R-a) + a\cos(p), \\ z(p) = a\sin(p), \end{cases} \qquad p \in [0, 2\pi]. \tag{3.7}$$

The equation for y is obtained by inserting equation (3.7) into equation (3.1),

$$y(p) = \pm\sqrt{R^2 - \left[(R-a) + a\cos(p)\right]^2 - b^2\sin(p)^2}. \tag{3.8}$$

The full parametrized curve is then,

$$\begin{cases} x(p) = (R-a) + a\cos(p), \\ y(p) = \pm\sqrt{R^2 - \left[(R-a) + a\cos(p)\right]^2 - b^2\sin(p)^2}, & p \in [0, 2\pi]. \\ z(p) = a\sin(p), \end{cases} \tag{3.9}$$

To get a continuous and differentiable curve, the $\pm$ sign needs to be removed. This is done by setting $y(p)$ to be positive for one rotation about the cylinder and then negative for the next,

$$y(p) = \begin{cases} \sqrt{R^2 - \left[(R-a) + a\cos(p)\right]^2 - b^2\sin(p)^2}, & \text{when } p \in [0, 2\pi] \\ -\sqrt{R^2 - \left[(R-a) + a\cos(p)\right]^2 - b^2\sin(p)^2}, & \text{when } p \in (2\pi, 4\pi). \end{cases} \tag{3.10}$$

Figure 3.3 shows the cylindrical and elliptic cylinders. While both intersect the sphere at the same point $(58, 15.5)$, the elliptical cylinder protrudes further into the sphere, giving the intersecting curve a larger range in the y-axis and a shorter range along the x-axis. This can be seen in Figure 3.4 where both curves are illustrated. From this figure, it is clear that the elliptic Viviani's curve gives a shorter range in y and x.

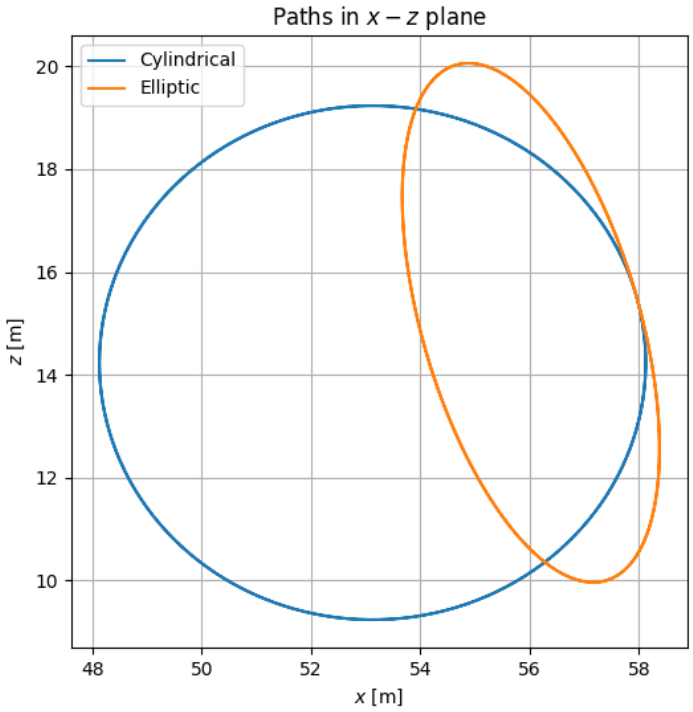


Figure 3.3: Side view of the Viviani's curve obtained using a cylindrical (blue) and elliptic (orange) cylinder. In this case the semi-axis $a = 2$ of the elliptic cylinder is smaller then $b = 5.2$ giving the elliptic cylinder an oval shape.

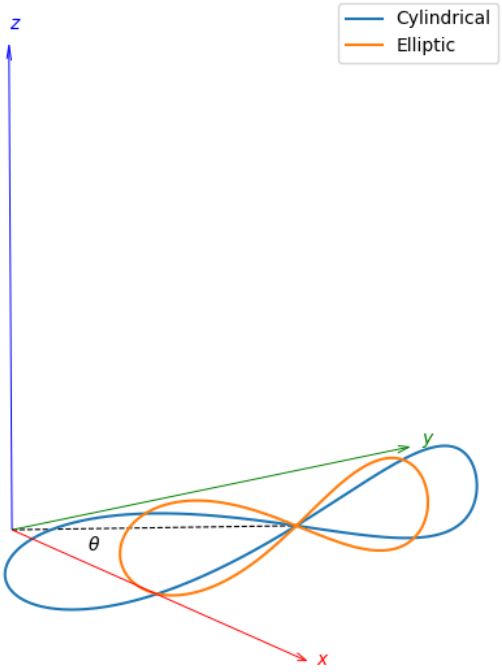


Figure 3.4: Cylindrical (blue) and elliptic (orange) Viviani's curves rotated θ degrees about the y-axis.

3.1.1.3 Position vector

The location of the kite on the predetermined path is defined by the position vector denoted as $\vec{r}(p)$. The position equations, given by Equation (3.5), is rotated around the y-axis to place the path at an angle θ above the sea bed,

$$\vec{r}(p) = R_y(\theta) \begin{bmatrix} x(p) \\ y(p) \\ z(p) \end{bmatrix}. \tag{3.11}$$

The derivative of the position vector $\vec{r}(p)$ with respect to p gives the tangent vector of the path at position p, and the second derivative determines how the tangent vector changes with p,

$$\vec{r}_p = \frac{\partial \vec{r}(p)}{\partial p}, \tag{3.12}$$

$$\vec{r}_{pp} = \frac{\partial^2 \vec{r}(p)}{\partial p^2}. \tag{3.13}$$

3.1.1.4 Path singularity

The equation used to obtain the y coordinate for the elliptic path, Equation (3.10), is continuous and smooth for all $p \in \mathcal{R}$. This means that all $p \in \mathcal{R}$ corresponds to a $y(p)$ coordinate. However, to obtain the spatial velocity and acceleration along the y-axis, the derivative of $y(p)$ needs to be found. Since there exist a discontinuity at

$p = 2n\pi,\ n \in \mathcal{R}$, the derivative of $y(p)$ is not well defined at these points, resulting in large unpredictable derivatives close to the singularity. The singularity occurs at the point where the two loops of the figure-eight meet, at $y = 0$.

To counteract unrealistic velocities and acceleration at or near the singularity, the first and second derivative of $\vec{r}(p)$ prior to the singularity points are used as the kite travels over the singularity. Which, if the region of constant $\vec{r}(p)$ derivatives is small, and the kite path velocity does not reach zero (risk of long travel time over singularity), likely results in smooth and realistic kite velocity $\vec{r}_p$ and acceleration $\vec{r}_{pp}$ at or near the singularities.

3.1.2 Kite Kinematics

To model the dynamics of the system, the position parameter p is described as a function of time, with position $p(t)$ and velocity $\dot{p}(t)$ along the curve with radians as units. To obtain the linear position and velocity in the inertial frame, the time derivatives of $\vec{r}(p(t))$ are calculated. Linear velocity $v(t)$ is obtained by taking the first time derivative of $\vec{r}(p(t))$ and applying the chain rule,

$$\vec{v}(t) = \frac{d}{dt}\vec{r}(p(t)) = \frac{d\vec{r}}{dp}\frac{dp}{dt},$$

$$\vec{v}(t) = \dot{\vec{r}} = \vec{r}_p\dot{p}. \tag{3.14}$$

This means that the linear velocity in the inertial frame can be found by multiplying the tangent of the path with the angular velocity along the path. Since the tangent of the path has units $[m/rad]$ and angular velocity is measured in $[rad/s]$, we have that the linear velocity is in $[m/s]$.

The linear acceleration in the inertial frame is found by taking the time derivative of the linear velocity. Applying product and chain rule gives,

$$\vec{a}(t) = \frac{d}{dt}\vec{v}(t) = \frac{d}{dt}(\vec{r}_p\dot{p}) = \frac{d\vec{r}_p}{dt}\dot{p} + \vec{r}_p\ddot{p} = \frac{d}{dp}\frac{dp}{dt}\frac{d\vec{r}}{dp}\dot{p} + \vec{r}_p\ddot{p} = \frac{d^2\vec{r}}{dp^2}\dot{p}^2 + \vec{r}_p\ddot{p},$$

$$\vec{a}(t) = \ddot{\vec{r}} = \vec{r}_{pp}\dot{p}^2 + \vec{r}_p\ddot{p}. \tag{3.15}$$

The acceleration has a tangential component along the path and a curvature component bending the velocity so that the kite stays in the figure-eight shape.

3.1.3 Reference frames

To simplify the derivations of the forces acting on the kite, three main reference frames were used. One inertial frame static at the seabed anchor point, one frame fixed to the center of gravity and rotation of the kite and one aligned with the relative water flow as perceived by the kite.

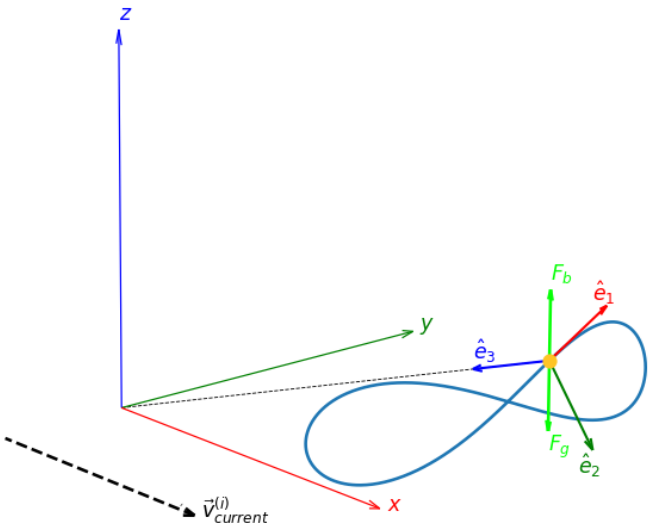


Figure 3.5: Figure-eight path followed by the kite (blue). The inertial (x, y, z) and path basis $(\hat{e}_1, \hat{e}_2, \hat{e}_3)$ shown in relation to each other and the kite trajectory. Furthermore, the absolute current direction in the inertial frame is shown (black).

3.1.3.1 Inertial frame

The frame used to derive the system kinematics with position $\vec{r}(p(t))$, velocity $\vec{v}(t)$, and acceleration $\vec{a}(t)$ vectors is refereed to as the inertial frame (i). The origin coincide with the tether anchor point and has axis x, y and z as illustrated in Figure 3.5. The x-axis points in the direction of water current, z-axis points upward and y-axis follows from the right hand rule.

3.1.3.2 Path frame

To describe the kites position along the path with respect to the inertial frame (i), the path reference frame (p) is defined. The path reference frame takes inspiration from the one used in [9] and those used to model aircraft dynamics. It has its origin at the center of mass of the kite. The basis has its first basis vector $\hat{e}_1$ along the tangent of the path, see Figure 3.5. $\hat{e}_3$ points towards the origin of (i) and is parallel to the tether, assuming the tether is rigid and can be approximated by a straight line. The third basis vector $\hat{e}_2$ completes the orthonormal basis and is found by taking the cross product of $\hat{e}_1$ and $\hat{e}_3$,

$$\hat{e}_1 = \frac{\vec{r}_p}{||\vec{r}_p||}, \tag{3.16}$$

$$\hat{e}_3 = -\frac{\vec{r}(p)}{||\vec{r}(p)||}, \tag{3.17}$$

$$\hat{e}_2 = \hat{e}_3 \times \hat{e}_1. \tag{3.18}$$

The path reference frame derived forms an orthonormal basis expressed in the inertial frame. Put together in a 3×3 matrix gives a direction cosine matrix where each

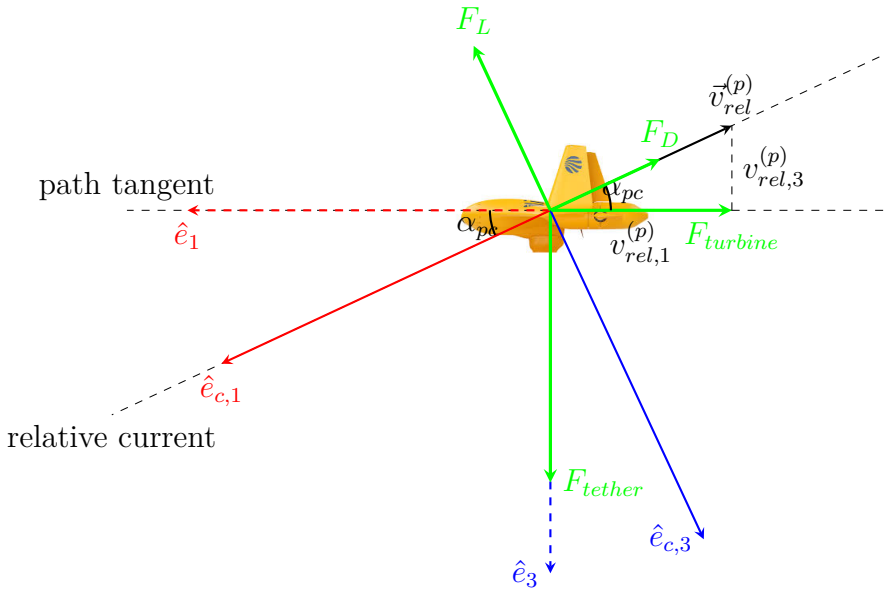


Figure 3.6: Two-dimensional side view of the kite illustrating the relation between the path and current frames as well as the hydrodynamical and tether forces acting on the kite. $\hat{e}_1$ point in the direction the kite is moving in while $\hat{e}_{c,1}$ points in the opposite direction of the kite relative flow direction. The current frame is rotated α_{pc} degrees about the $\hat{e}_2$-axis in relation to the path frame. The remaining unit vectors $\hat{e}_2$ and $\hat{e}_{c,2}$ point into the paper.

column is the coordinate of a path axis expressed in the inertial frame. As such, to transfer vectors from the path to the inertial frame, the rotation matrix R_{pi} is used,

$$R_{pi} = \begin{bmatrix} \hat{e}_1 & \hat{e}_2 & \hat{e}_3 \end{bmatrix}. \tag{3.19}$$

Since the axes $\hat{e}_i$ forms an orthonormal basis, R_{pi} is an orthogonal matrix and the rotation matrix R_{ip} can be found,

$$R_{pi}^T \cdot R_{pi} = I \rightarrow R_{ip} = R_{pi}^{-1} = R_{pi}^T. \tag{3.20}$$

3.1.3.3 Relative current frame

The water current vector $\vec{v}_{current}^{(i)}$ determines the direction and magnitude of the water flow and is assumed to be in the positive x direction. By subtracting the water current vector with the linear kite velocity, the relative flow vector $\vec{v}_{rel}^{(i)}$ is found,

$$\vec{v}_{rel}^{(i)} = \vec{v}_{current}^{(i)} - \vec{v}_{kite}^{(i)}. \tag{3.21}$$

This vector describes how the surrounding water moves from the kites perspective, which is of interest when calculating the hydrodynamical and turbine induced forces acting on the kite. The third reference frame, the relative current frame (c), was introduced to simplify these calculations. The relative current frame is found by a rotation about the $\hat{e}_2$ axis such that its first basis vector $\hat{e}_{c,1}$ is aligned and parallel

to the relative water flow, pointing in the opposite direction, see Figure 3.6. The rotation is described as,

$$R_{pc}(-\alpha_{pc}) = \begin{bmatrix} \cos\alpha_{pc} & 0 & \sin\alpha_{pc} \\ 0 & 1 & 0 \\ -\sin\alpha_{pc} & 0 & \cos\alpha_{pc} \end{bmatrix}. \tag{3.22}$$

The angle of attack, α_{pc}, is obtained as the angle between the relative flow velocity in the path frame and the $\hat{e}_1$-axis. The changed sign of α_{pc} is used to get counter-clockwise rotation as the positive rotational direction. The relative flow velocity in the path frame is obtained as,

$$\vec{v}_{rel}^{(p)} = R_{pi}\vec{v}_{rel}^{(i)} = \begin{bmatrix} v_{rel,1}^{(p)} \\ v_{rel,2}^{(p)} \\ v_{rel,3}^{(p)} \end{bmatrix}. \tag{3.23}$$

Hence,

$$\alpha_{pc} = atan2(-v_{rel,3}^{(p)}, -v_{rel,1}^{(p)}). \tag{3.24}$$

3.1.4 Forces

The forces acting on the kite are the gravitational force F_g, buoyancy force F_b, tether force F_{tether}, turbine force $F_{turbine}$ and the hydrodynamical forces F_a. Each force vector is calculated in the frame in which the specific force is the simplest to calculate.

The gravitational and buoyancy forces are calculated in the inertial frame since they are parallel to the z-axis. The gravitational force is simply dependent on the mass m of the kite and the gravitational constant g and points straight down in negative z direction,

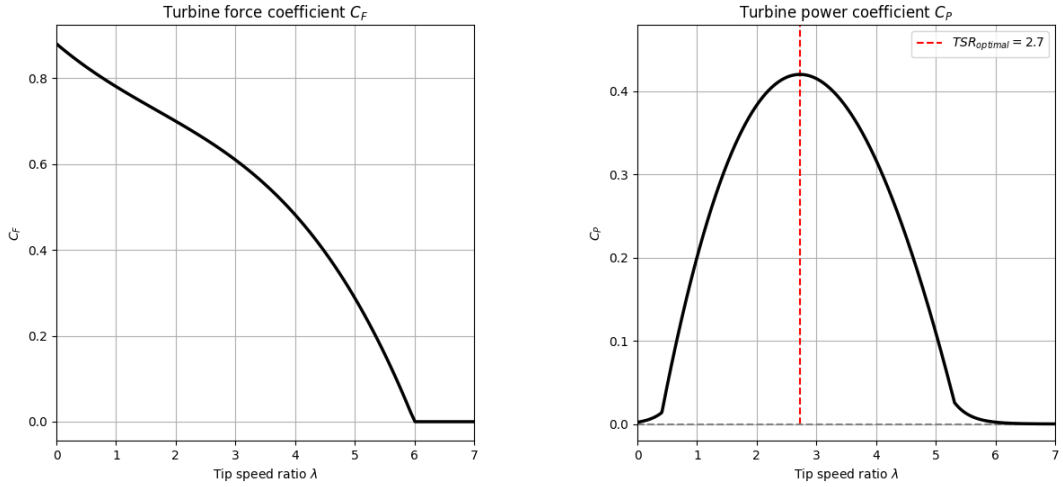
$$\vec{F}_{grav}^{(i)} = mg \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}. \tag{3.25}$$

The bouynacy force is according to the Archimedes' principle equal to the force of the weight of the water displaced by the submerged kite, and acts upwards in positive z direction,

$$\vec{F}_{bouy}^{(i)} = \rho V g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{3.26}$$

ρ is the density of salt water and V is the volume displaced by the submerged kite.

The force induced on the kite by the turbine point in negative $\hat{e}_1$ direction and is therefore calculated in the path frame. To obtain the turbine force $F_{turbine}$ the following equation is used,



(a) Force coefficient.

(b) Power coefficient.

Figure 3.7: Turbine specific dimensionless coefficient maps, mapping a TSR value λ to a C_f and C_p. In figure **(b)** the optimal TSR value, maximizing C_p, is ~ 2.7, indicated by the red dashed line.

$$F_{turbine} = \frac{1}{2}\rho A_{turbine}(-v_{rel,1}^{(p)})^2 C_p(\lambda). \tag{3.27}$$

Here, $-v_{rel,1}^{(p)}$ is the positive relative current that is traversing the turbine, $A_{turbine}$ is the swept area of the blades and $C_p(\lambda)$ is the non-dimensional power coefficient for the turbine. $C_p(\lambda)$ depends on the tip speed ratio obtained in Equation (1.2). In a similar fashion, the power extracted by the turbine can be calculated as,

$$P_{turbine} = \frac{1}{2}\rho A_{turbine}(-v_{rel,1}^{(p)})^3 C_f(\lambda). \tag{3.28}$$

The difference from above is that $C_f(\lambda)$, the dimensionless thrust coefficient, is used instead of $C_p(\lambda)$ and that the cube root of the velocity is used. The dimensionless λ dependent coefficients maps, $C_f(\lambda)$ and $C_p(\lambda)$, used are shown in Figure 3.7 and are estimated from the work in [35]. The estimation was done by fitting a third degree polynomial to model test points found in the paper. Since λ depends on the turbines rotational speed, the dynamics of the turbine and the connected generator needs to be derived, this is done below in Section 3.1.5.2.

The turbine force $F_{turbine}$ acts in the negative $\hat{e}_1$ direction. Therefore, the following force vector is obtained in the path frame,

$$\vec{F}_{turbine}^{(p)} = \begin{bmatrix} -F_{turbine} \\ 0 \\ 0 \end{bmatrix}, \tag{3.29}$$

which is rotated to the inertial frame,

$$\vec{F}^{(i)}_{turbine} = R_{pi}\vec{F}^{(p)}_{turbine}. \tag{3.30}$$

For calculation of the aerodynamical forces acting on the kite, the current frame is the easiest frame to use. This is because lift and drag forces act parallel with, and orthogonal to, the direction of the relative flow, which is parallel with the $\hat{e}_{c,1}$ axis. Since the trajectory of the kite is locked to movements in the direction of the path tangent, the side flow acting on the kite is disregarded. In practice, this means that the relative water velocity in the current frame $\vec{v}^{(c)}_{rel}$ has a non-zero flow along the $\hat{e}_{c,1}$ axis only,

$$\vec{v}^{(c)}_{rel} = \begin{bmatrix} \sqrt{(v^{(p)}_{rel,1})^2 + (v^{(p)}_{rel,3})^2} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} v^{(c)}_{rel} \\ 0 \\ 0 \end{bmatrix}. \tag{3.31}$$

Using the absolute flow velocity relative to the kite $v^{(c)}_{rel}$ the aerodynamical forces lift and drag can be calculated,

$$F_L = \frac{1}{2}\rho(v^{(c)}_{rel})^2 A_{wing} C_L(\alpha), \tag{3.32}$$

$$F_D = \frac{1}{2}\rho(v^{(c)}_{rel})^2 A_{wing} C_D(\alpha). \tag{3.33}$$

Here, A_{wing} is the reference area of the wing, in other words, the area of the wing seen from above. $C_L(\alpha)$ and $C_D(\alpha)$ are the lift and drag coefficients, which depends on the angle between the kite and the relative current and are specific to the D4 kite. $C_L(\alpha)$ and $C_D(\alpha)$ are shown in Figure 3.8 and are retrieved from earlier sub-sea kite research [42]. The angle of attack α used in $C_L(\alpha)$ and $C_D(\alpha)$ is the total angle between the kite and the relative flow direction, in our case,

$$\alpha = \alpha_{pc} + \alpha_{pb}. \tag{3.34}$$

Where α_{pb} is the angle between the kite body and the path frames $\hat{e}_1$ axis.

Since the lift force F_L acts in negative $\hat{e}_{c,3}$ direction and the drag force F_D in negative $\hat{e}_{c,1}$, the resulting aerodynamic force acting on the kite seen from the current frame is,

$$\vec{F}^{(c)}_{aero} = \begin{bmatrix} -F_D \\ 0 \\ -F_L \end{bmatrix}. \tag{3.35}$$

Transformed to the inertial frame,

$$\vec{F}^{(i)}_{aero} = R_{pi}R^T_{pc}\vec{F}^{(c)}_{aero}. \tag{3.36}$$

The final force acting on the kite, the tether force, can not be calculated at the moment. The reason for this is that the tether force is dimensioned to counteract all forces in the $\hat{e}_3$ direction to ensure that the model only moves tangential to the

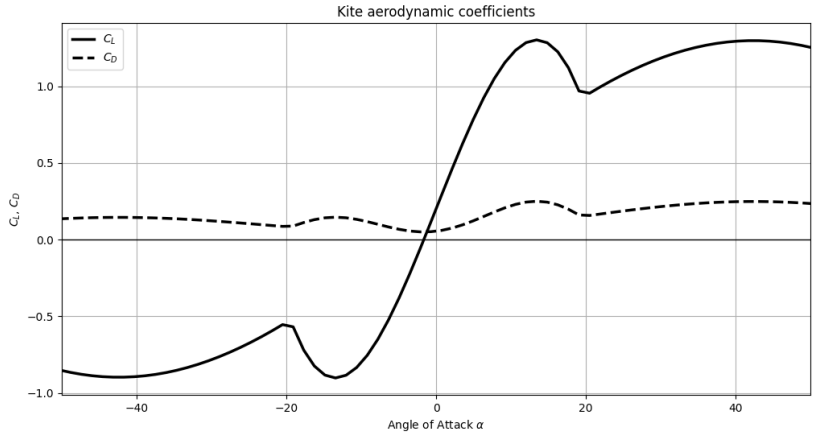


Figure 3.8: Kites non-dimensional lift and drag coefficient maps, mapping angle of attack α to C_L and C_D.

path and not perpendicular to it. Therefore, F_{tether} is found below through Newtons second law of motion as the force in $\hat{e}_3$ direction that keeps the kite on the path. With all force contributions transformed to the inertia frame the sum of forces acting on the kite can be calculated as,

$$\vec{F}^{(i)}_{sum} = \vec{F}^{(i)}_{aero} + \vec{F}^{(i)}_{turbine} + \vec{F}^{(i)}_{grav} + \vec{F}^{(i)}_{bouy} + \hat{e}_3 \cdot F_{tether}. \tag{3.37}$$

3.1.5 Dynamical model

To model the dynamics of the system, the position and velocity along the path $p(t)$ and $\dot{p}(t)$ are used as states. Furthermore, to keep track of the dynamics of the generator, its angular velocity $\omega_{gen}(t)$ is set as the third state and its electrical torque $T_{el,gen}(t)$ as the forth and final state,

$$\vec{x}(t) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} p(t) \\ \dot{p}(t) \\ \omega_{gen}(t) \\ T_{el,gen}(t) \end{bmatrix}. \tag{3.38}$$

To model the dynamics of the kite the ordinary differential equations (ODE) for each state variable needs to be found, resulting in the following system of coupled ODEs,

$$\dot{\vec{x}}(t) = \begin{bmatrix} \dot{p}(t) \\ \ddot{p}(t) \\ \dot{\omega}_{gen}(t) \\ \dot{T}_{el,gen}(t) \end{bmatrix} = \begin{bmatrix} x_2 \\ \ddot{p}(t) \\ \dot{\omega}_{gen}(t) \\ \dot{T}_{el,gen}(t) \end{bmatrix}, \tag{3.39}$$

where each state derivative needs to be found as functions of states and constants.

3.1.5.1 Kite dynamics

From Newtons second law of motion,

$$m\vec{a} = \vec{F}^{(i)}_{sum}, \tag{3.40}$$

an equation for $\ddot{p}(t)$ was found by inserting the linear acceleration, Equation (3.15), and the sum of forces acting on the kite, Equation (3.37), into Equation (3.40), and projecting onto $\hat{e}_1$. The projection onto $\hat{e}_1$ was done to find the contributions to the linear acceleration along the direction of motion.

$$m(\hat{e}_1 \cdot \vec{a}) = \hat{e}_1 \cdot \vec{F}^{(i)}_{sum}, \tag{3.41}$$

$$m[(\hat{e}_1 \cdot \vec{r}_{pp})\dot{p}^2 + ||\vec{r}_p||\ddot{p}] = \hat{e}_1 \cdot (\vec{F}^{(i)}_{aero} + \vec{F}^{(i)}_{turbine} + \vec{F}^{(i)}_{grav} + \vec{F}^{(i)}_{bouy}) + (\hat{e}_1 \cdot \hat{e}_3)F_{tether},$$

where $\hat{e}_1$ and $\hat{e}_3$ are orthogonal and the scalar product equals 0.

$$m[(\hat{e}_1 \cdot \vec{r}_{pp})\dot{p}^2 + ||\vec{r}_p||\ddot{p}] = \hat{e}_1 \cdot (\vec{F}^{(i)}_{aero} + \vec{F}^{(i)}_{turbine} + \vec{F}^{(i)}_{grav} + \vec{F}^{(i)}_{bouy}),$$

$$\ddot{p}(t) = \frac{\hat{e}_1 \cdot (\vec{F}^{(i)}_{aero} + \vec{F}^{(i)}_{turbine} + \vec{F}^{(i)}_{grav} + \vec{F}^{(i)}_{bouy}) - m(\hat{e}_1 \cdot \vec{r}_{pp})\dot{p}^2}{m||\vec{r}_p||}. \tag{3.42}$$

In a similar fashion, the tether force F_{tether} can be derived by projecting Newtons second equation (3.40) onto the $\hat{e}_3$ unit vector,

$$m(\hat{e}_3 \cdot \vec{a}) = \hat{e}_3 \cdot \vec{F}^{(i)}_{sum}, \tag{3.43}$$

$$m[(\hat{e}_3 \cdot \vec{r}_{pp})\dot{p}^2 + ||\vec{r}_p||\ddot{p}] = \hat{e}_3 \cdot (\vec{F}^{(i)}_{aero} + \vec{F}^{(i)}_{turbine} + \vec{F}^{(i)}_{grav} + \vec{F}^{(i)}_{bouy}) + (\hat{e}_3 \cdot \hat{e}_3)F_{tether},$$

where the scalar product $\hat{e}_3 \cdot \hat{e}_3$ equals unity. Solving for the tether force,

$$F_{tether} = m(\hat{e}_3 \cdot \vec{r}_{pp})\dot{p}^2 - \hat{e}_3 \cdot (\vec{F}^{(i)}_{aero} + \vec{F}^{(i)}_{turbine} + \vec{F}^{(i)}_{grav} + \vec{F}^{(i)}_{bouy}). \tag{3.44}$$

3.1.5.2 Generator dynamics

As stated above, the turbine dynamics is needed to obtain the rotational speed of the turbine $\omega_{turbine}$. The turbine is connected through a gearbox to the generator, where the gearbox increases the rotational speed of the generator ω_{gen} in relation to the turbine,

$$\omega_{gen} = N\omega_{turbine}, \tag{3.45}$$

where N is the gear ratio. The asynchronous induction machine used as generator is modeled with the swing equation,

$$J_{gen}\dot{\omega}_{gen} = T_{mech} + T_{el,gen}. \tag{3.46}$$

Here J_{gen} is the inertia of the generator and turbine system, T_{mech} is the mechanical torque put on the generator by the turbine and $T_{el,gen}$ is the electrical breaking torque realized by decreasing the slip of the induction machine. T_{mech} is obtained directly from the torque exerted on the turbine by the water flow $T_{turbine}$,

$$T_{mech} = \eta_{gearbox} \frac{T_{turbine}}{N}, \tag{3.47}$$

where $\eta_{gearbox}$ is the efficiency of the gearbox. $T_{turbine}$ is derived from its relation to turbine power and angular velocity,

$$T_{turbine} = \frac{P_{turbine}}{\omega_{turbine}}, \tag{3.48}$$

where $P_{turbine}$ was calculated from Equation (3.28). In contrast to T_{mech}, T_{el} can not be derived from known values. It instead is determined by the slip (difference between stator and rotor frequency) in the induction generator, which in turn depends on the voltages and currents in all of the phases. In this paper, the induction machine is simplified to its swing equation and a PI driven T_{el}. This is to avoid heavy simulations, stiffness in the simulation due to large time-scale differences in the subsystems and to be able to divert more work to the control algorithms. Furthermore, since the time-scale of the induction generators electrical processes is much smaller then that of the kite and turbine dynamics, this simplification should not reduce the validity of the model considerably.

The error for the PI torque controller is obtained as the difference between the reference generator speed and the real generator speed,

$$e = \omega_{gen,ref} - \omega_{gen}, \tag{3.49}$$

where $\omega_{gen,ref}$ is determined by the controller. The PI controller derives the electric torque reference $T_{el,ref}$ from the error and its integral,

$$T_{el,gen,ref} = K_p\, e + K_i \int e = K_p\, e + K_i I. \tag{3.50}$$

To avoid completely neglecting the electric dynamics of the induction generator, a first order generator model is implemented [3],

$$\dot{T}_{el,gen} = \frac{1}{\tau_{el,gen}} (T_{el,gen,ref} - T_{el,gen}). \tag{3.51}$$

Here, $\tau_{el,gen}$ is the time constant of the generator [3], which determines how quickly the electric torque follows the reference. From the electric torque ODE, Equation (3.51), the electric torque can be derived. Finally the torque is compared to the generators real torque-speed curve to keep the torque within generator specific torque boundaries. Furthermore, from the torque-speed curve, the efficiency of the generator at a specific point can be found, which is used when calculating the power produced by the generator,

$$P_{gen} = \eta_{gen}(T_{el,gen}, \omega_{gen})\, T_{el,gen}\, \omega_{gen}. \tag{3.52}$$

From the the generators swing equation (3.46) and the torques T_{mech} and $T_{el,gen}$, the dynamics of the generator can be solved from $\dot{\omega}_{gen}$,

$$\dot{\omega}_{gen} = \frac{T_{mech} + T_{el,gen}}{J_{gen}}. \tag{3.53}$$

The torque-speed-efficiency map was interpolated from test points obtained from a generator test previously executed by Minesto, as well as generator specification. Therefore, this is not included in this report and the clipping of $T_{el,gen}$ and the derivation of η_{gen} is not further explained.

3.1.6 Sensors

The kite sensors are modeled by adding gaussian noise to each of the simulated values. The sensors are modeled with a sampling rate of 0.02 seconds to match the data measurements from the real kite. While most sensor readings can be obtained directly from the system model, this is not the case for the used accelerometer and gyroscope IMU sensor readings which have to be derived.

3.1.6.1 Accelerometer

The body relative acceleration $\vec{a}^{(b)}$ is calculated from the inertial acceleration $\vec{a}^{(i)}$ of the kite, which can be obtained from the acceleration Equation (3.15). By rotating the inertial acceleration to the body relative frame, the acceleration felt by the kites IMU is obtained,

$$\vec{a}^{(b)} = R_{pb} R_{pi}^T \vec{a}^{(i)}. \tag{3.54}$$

3.1.6.2 Gyroscope

The gyroscope, or angular velocity measurements of the kite are obtained from the kite body frame. It is calculated by using the present and past rotation matrices. The rotation from the inertial to body frame is $R_{ib} = R_{pi}^T \cdot R_{pb}(\alpha_{pb})$ as given in Section 3.1.3. The kite velocity at time t is given as,

$$\vec{v_b}(t) = R_{ib}(t) v_i. \tag{3.55}$$

In order to obtain the gyroscopic measurements, we want to find a matrix ΔR such that $\vec{v_b}(t) = \Delta R \cdot \vec{v_b}(t-1)$. From equation (3.55), we get that,

$$R_{ib}(t) v_i = \Delta R R_{ib}(t-1) v_i \rightarrow \Delta R = R_{ib}(t) R_{ib}(t - \Delta t)^T. \tag{3.56}$$

ΔR tells us how the body frame rotated between two instances in the figure eight kite motion. The Euler angle rates $(\dot{\psi}, \dot{\theta}, \dot{\phi})$ can now be obtained from the rotation matrix and the timestep to give the angular velocity measurements $(\omega_x, \omega_y, \omega_z)$ by using the kinetic transformation matrix $T(\phi, \theta)$ given as,

$$\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} = \underbrace{\begin{bmatrix}1 & 0 & -\sin\theta\\ 0 & \cos\phi & \sin\phi\cos\theta\\ 0 & -\sin\phi & \cos\phi\cos\theta\end{bmatrix}}_{T(\phi,\theta)} \begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}. \tag{3.57}$$

3.1.7 Water current

The water current vector denoted as $\vec{v}^{(i)}_{current}$ is a three dimensional vector with positive contribution along the inertial x-axis and no contributions along the other dimension, see Figure 3.5. That is, the water current flows along the sea bed straight towards the center of the kites path. Since the water current is the main force giving rise to the acceleration of the kite, its size and behavior over time is important to model as it greatly effects the dynamics of the kite.

Two models of water current was implemented and used. Firstly, a deterministic, constant model where the water current was set to a specified value and remained at that value at all times. Secondly, a more dynamic and realistic model was implemented. This model is stochastic and contains randomized smoothed steps around a set mean current, sinusoidal variations and gaussian noise. Here a more detailed explanation follows:

1. The mean current v_{mean}, the current range $[v_{min},\ v_{max}]$ and switching time range $[t_{switch,min},\ t_{switch,max}]$ are set.
2. The water current v is initialized to the mean current v_{mean} and a switching time t_{switch} is randomized within a given range.
3. If the time t_{switch} has passed since the last switch, a random target current v_{target} is randomly set within the specified range.
4. Through the use of the hyperbolic tangent function, v is smoothly changed from the prior current to v_{target}, in a predetermined time. One step along the smooth change curve is taken each time-step.
5. For each time-step, a time dependent sinusoidal variation and a small value sampled from a gaussian distribution is added to v.
6. The process is repeated from step 2.

The time dependent sinusoidal variations added makes the resulting wave none flat at all time periods. Without it, the water current would be constant after the hyperbolic changing period and before the next randomized step. The gaussian sample added each time-step gives the current a low amplitude white noise. An example current generated by the stochastic method is presented in Figure 4.8b. The deterministic method instead simply generates a flat water current at a specified velocity for all time-steps.

The mean water current was set to be in the range $[2.0, 2.5]$ since that is approximately the maximal water current obtained at the Faroe islands [18]. The mean

water current was selected to represent plausible operating conditions for the studied system. The shape of the stochastic water current is argued to be realistic since the kite is rapidly moving over many water columns, which likely results in the kite observing a much more stochastic and changing water current compared to a stationary power plant.

3.2 Simulation

Using the ODEs, Equations (3.42), (3.53) and (3.51), the dynamics of the system, described by Equation (3.39), is fully described. To solve these, a forth order Runge-Kutta solver was implemented. A visual description of the system layout is presented in Figure 3.1.

The simulation process is defined as follows:

1. Define initial state values $p(0)$, $\dot{p}(0)$, $\omega_{gen}(0)$ and $T_{el,gen}(0)$.
2. Calculate path and kite dynamics according to Sections 3.1.1 and 3.1.2.
3. Get turbine speed reference ω_{ref} from the controller.
4. Use ω_{ref} to calculate the dynamics of the turbine and generator system, and in addition updating $\dot{\omega}_{gen}$ and $\dot{T}_{elec}$.
5. Using what is obtained from path, kite and turbine dynamics, kite dynamics is calculated and $\dot{p}$ and $\ddot{p}$ is derived.
6. Simulate noisy measurement using the modeled sensors.
7. Store states and variables derived from the states.
8. Return to Step 2 if not maximal simulation time is reached.

The measurements, with added noise, retrieved from the simulation are compared to the real data. The parameters in the model and simulation are then tuned to obtained simulated data faithful to the real kite data.

To obtain a realistic simulation, different time-steps was used for different systems. Three time-steps were defined, firstly a physics time-step dt_{sim} determining how often the kite dynamics are updated. Secondly, a measurement time-step $dt_{measure}$ determining how often new sensor values are retrieved. And finally, the controller time-step $dt_{controller}$ which determines how often the specified controller generates a new generator control signal. The two first time-steps were set to 0.01 and 0.02 respectively. $dt_{measure}$ was set to 0.02 due to limitations seen on the real kite, while dt_{sim} was set to 0.01 to allow a balanced relation between simulation speed and accuracy. $dt_{controller}$ differs from controller to controller. For the original and RNN-based controller it was set to 0.02, while for SAC-based controllers it was set to 0.05. The fact that $dt_{measure}$ and $dt_{controller}$ is larger then dt_{sim} is in practice handled by skipping Steps 6 and 3 respectively when simulating the kite dynamics, and using the last sensor or controller values until the next update occurs.

4

Control Methodology

This chapter aims to describe the methodology workflow in creating the control strategies introduced in Chapter 2. It starts with explaining and motivate the steps involved in creating a reliable RNN based predictive controller, to then move on to the SAC controller, its environment and implementations.

4.1 Supervised predictive controller

The supervised predictive controller was proposed as a solution to the time delayed signal generated by the baseline controller. The predictive part was introduced under the assumption that the controller could benefit from knowing future system dynamics, either to account for system inertia, or to take advantage of the periodic system behavior.

The methodology of obtaining the new controller consist of an offline training phase and an online deployment phase. By offline, we mean the supervised training phase, where historical simulation data is used to train a RNN to forecast future relative velocity. During this stage, the controller itself is not active, we are only learning the system dynamics from data. By online, we refer to the real-time deployment phase, where the trained RNN continuously predicts future inflow and feeds the predictions into a generator speed reference controller. This section lays out the offline and online strategy that results in a new novel control method. An overview is shown in Figure 4.1.

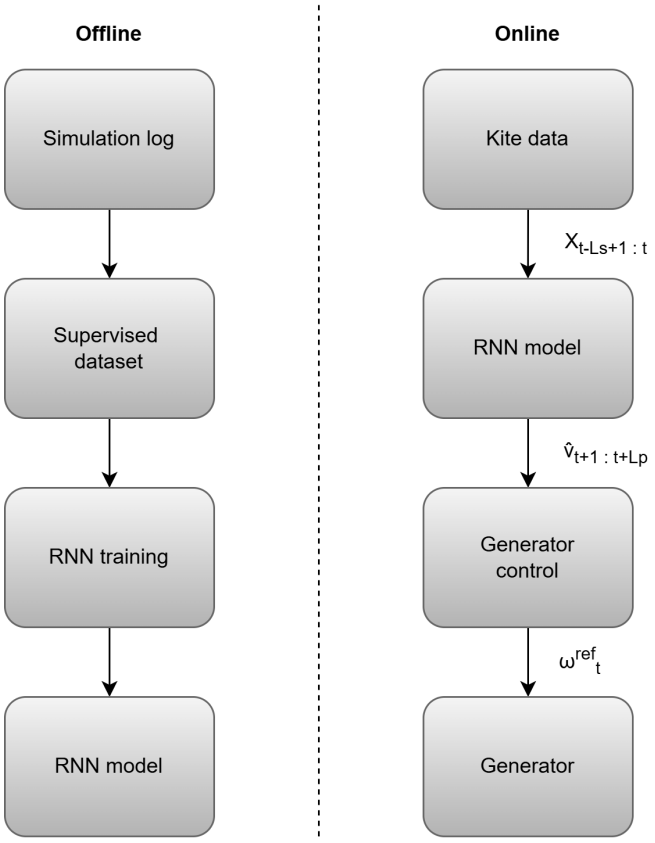


Figure 4.1: Block diagram of predictive controller offline and online system overview

4.1.1 Supervised dataset

The sensor readings derived in Section 3.1.6 form the feature space of the supervised dataset and were used to calculate the target inflow velocity outputs. Sensor data were gathered for a four hour simulated kite run with time varying water current conditions, designed to mimic realistic environmental variability. The data preprocessing focused on obtaining a delay compensated estimate of the inflow velocity to create a dataset with minimal time lag. This way, the network could learn from almost instantaneous inputs and targets to predict future ones.

It has to be acknowledged that using the power turbine equation 1.1 to make sense of the inflow velocity from the power and rotation sensor signals is a gross assumption due to the kite design and sensor setup. However, it is argued that these signals can be manipulated in such a way that the calculated inflow velocity is accurate enough to capture meaningful inflow behavior. The inflow velocity was calculated by combining the TSR and turbine equations and solve for the real positive root to,

$$\frac{1}{2}(\eta\rho A v_{rel}^3 Cp(\lambda)) - P = 0, \tag{4.1}$$

where

$$\lambda = \frac{\omega r_{\text{turb}}}{v_{\text{rel}}}.$$

4.1.1.1 Inflow velocity time lag

The time lag that arise when calculating inflow velocity from the turbine power equation is a result of where sensors are physically located in the system and the systems inertia. The kite sensors sees a rotational velocity and power change from a increased inflow velocity only when the change has resulted in a faster shaft rotation or higher applied torque by the generator. As such, the delay in ω_{sensor} and P_{sensor} has to be taken into account to make the power and rotation signal in sync with the mechanical power and rotation from the turbine rotor. Since the turbine is directly connected to the generator through a gearbox, the ω_{sensor} is expected to be very small compared to the power sensor which incorporates the generator system dynamics.

The ω_{sensor} and P_{sensor} delays were accounted for by comparing the signals to the signal that was expected to be the most responsive to inflow velocity changes, namely the tether force. By looking at a time-frame where the system showed stable periodic behavior, the cross correlation between these signals could estimate the time delay to create the time-aligned ω_{sensor} and P_{sensor} signals.

4.1.1.2 Inflow velocity magnitude

Even though the time delayed compensated sensor signals are shifted accurately to be in near perfect phase with tether force and as a result relative inflow, their respective magnitudes are still inaccurate seen from the equation. Sensor noise, turbine, gear and generator losses and efficiencies all play a significant part in manipulating the output power we see to make it deviate from the equations turbine power. As such, the calculated velocity is only considered to be a estimate for the flow behavior rather than its precise value. However, one could argue that a near precise value in magnitude could be obtained by a constant multiple of the calculated power that accounts for all the losses. As described later, this is used in one of the control designs.

4.1.1.3 Preprocessing validation

Figure 4.2 demonstrates the significance of the signals delay in comparison to the tether force. As expected, the delay in output power was more severe showing a 0.74 second delay, compared to the 0.04 seconds for the generator speed, barely visible in the plot. Note that the delay in generator speed is a direct result from the generator swing Equation (3.46) and the moment of inertia from the generator and rotor mass. Additionally, the power delay comes from the PI control modeling of the generator torque and the added torque state dynamics. The final calculated v_{rel} seen in plot (c) (Figure 4.2) shows promising validation of the calculated inflow velocity. The signal appears to capture the periodic behavior and phase of the real velocity inflow. As described earlier, the magnitude difference was likely a result of the output power estimation used in the equation. A constant multiple of the calculated v_{rel} would likely decrease the magnitude difference between the signals notably. While the validation of v_{rel} only exists because of the simulation environment, it serves as a

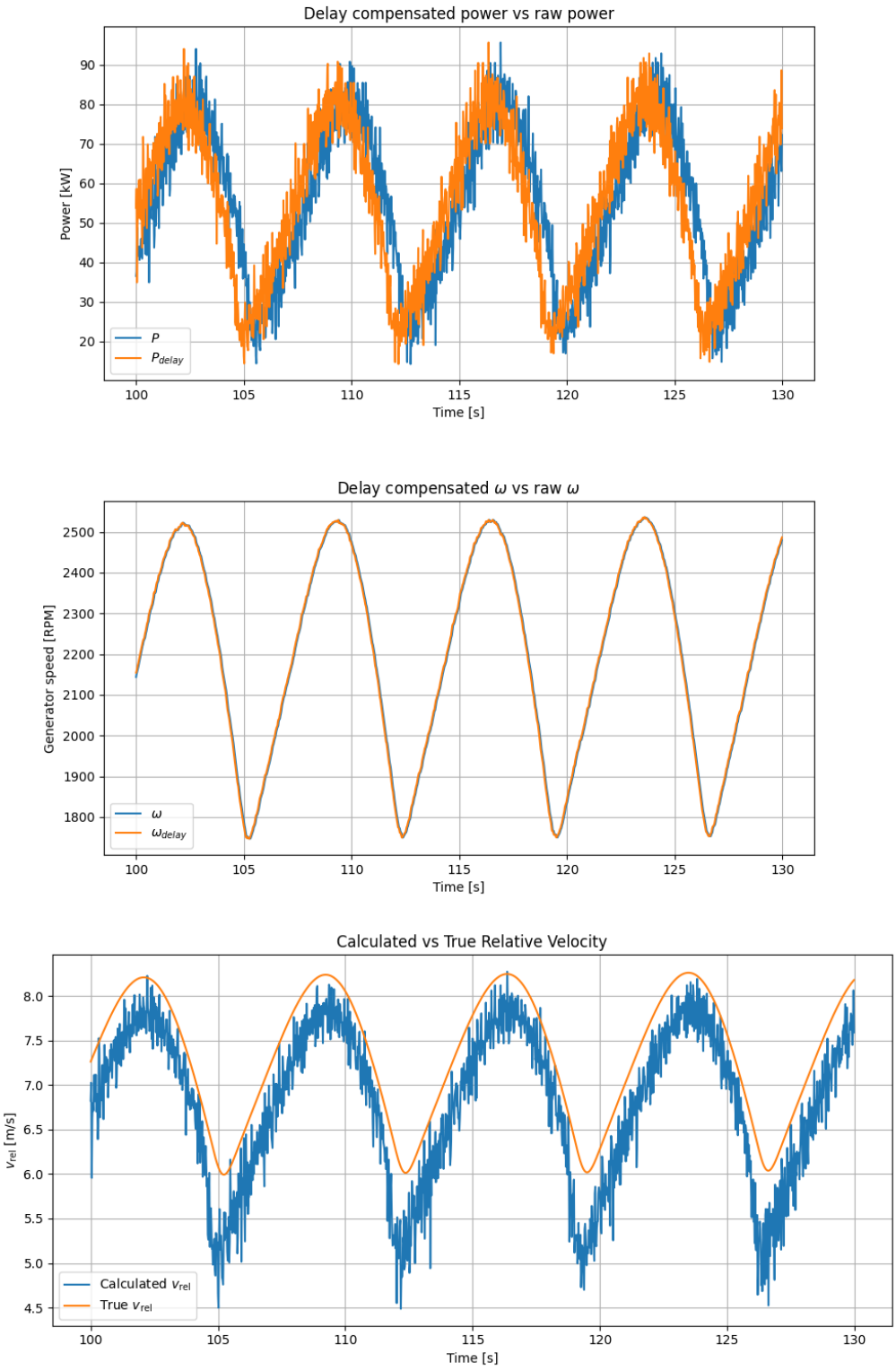


Figure 4.2: Validation of signal preprocessing and inflow reconstruction. **(a)** Raw and delay-compensated power signal. **(b)** Raw and delay-compensated generator speed. **(c)** Calculated versus true relative inflow velocity.

confirmation that the turbine and TSR equations can be used to get meaningful data of the inflow.

4.1.1.4 Output data leakage handling

Using the strategy laid out above, every sensor reading data point from the kite run could be mapped to a relative velocity output. To ensure that v_{rel} is treated as an independent physical state and that no data leakage occurs, signals used to derive it was excluded from the feature space. Along with the generator speed and power output, the torque was excluded as it is mathematically tied to both signals and hence would give a indirect data leakage. The data was split in a 80/20 ratio training and test set with features normalized using the training set mean and standard deviation.

4.1.2 Velocity forecasting

With the supervised dataset in place, a forecasting many-many RNN was developed based on the modern recurrent network fundamentals explained in Section 2.1.1. The idea was to take a input sequence window, compress the past information in a hidden state, and let it recursively evolve to multi-step forecast future inflow velocities. By doing so, the hidden state could learn the system state and forecasts based on the learned dynamics. The architecture used is explained as well as results of model training on the holdout test set.

4.1.2.1 RNN design

While many designs exists for many-many forecasting, the open-loop design was a choice made as a result from data analysis and the targeted output results. The sensor signals showed clear periodic behavior, and as such, it was expected that the non-linear dynamics could be captured easily. Additionally, as all sensor measures and hence the calculated velocity experienced noise, the model risked overfitting to it if teacher forcing or a closed-loop approach were used. Furthermore, a model that could generalize the behavior of future velocity inflow was considered sufficient for developing the new control method.

The design of the RNN forecasting is shown in Figure 4.3 and can be described as consisting of two phases. The term *phase* here is used to separate the compressing of past information from the forecasting and should not be confused with encoder-decoder RNNs. While encoder–decoder RNN architectures use separate networks for encoding-decoding, and handles input and output sequences of variable length, the model used in this work relies on a single RNN and uses fixed length sequences. The hidden units h are GRUs as explained in Section 2.1.1 and was a design choice made for maximizing computational speed.

The first phase consisted of compressing a input sequence $(X_{t-L+1}, ..., X_{t-1}, X_t)$ into a hidden state. For every input sequence window, the hidden state h_0 was initialized to zero and propagated through the network to give the hidden state at h_t. Here,

L_s is the sequence length and denotes how many time-steps are compressed into the hidden state.

The second phase uses the hidden state at h_t to unfold the network and generate the output sequence O_t. The output is a fully connected linear layer that maps the hidden state vector to a predicted relative inflow velocity as such,

$$\hat{v}_t = W_o h_t + b_o, \tag{4.2}$$

where W_o is the connected output weight matrix and b_o is the bias.

The inputs x_0 were set to zero for every forecasting time-step as the network was expected to have compressed all the information needed in h_t to evolve on its own. The prediction length is denoted L_p and determines how many future steps the network predicts.

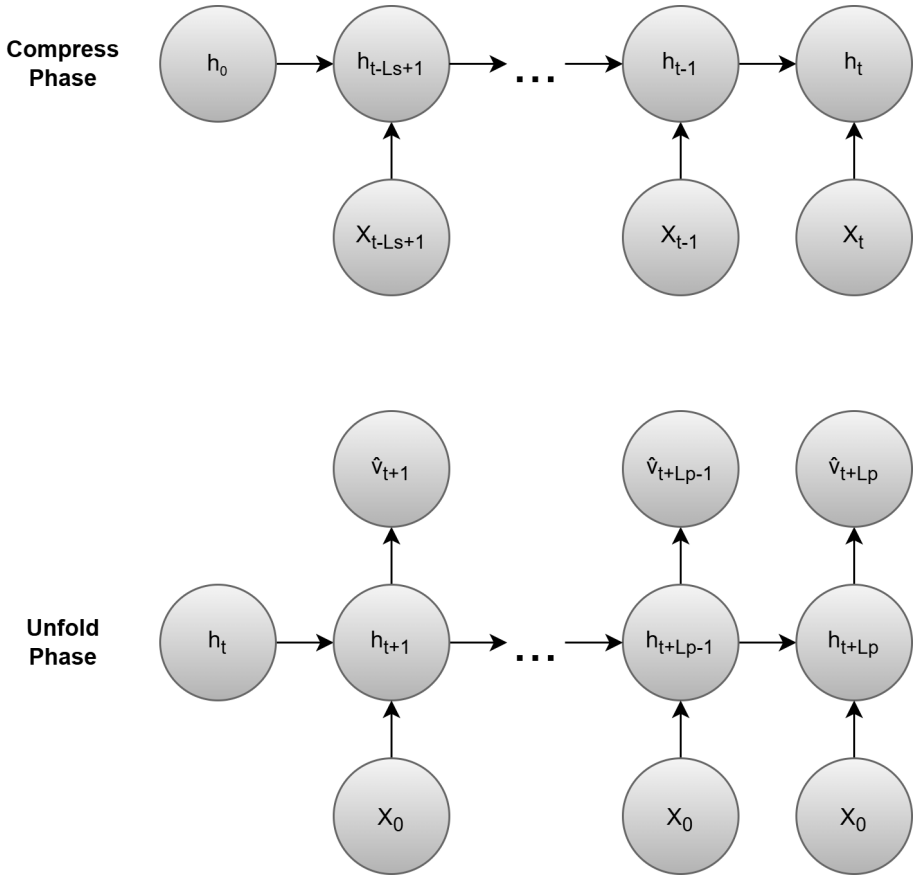


Figure 4.3: RNN compressing and unfolding architecture

Although the time-series is continuous and all of its history in principle are reflected in the current system state of the kite, only a few past states are considered using the independent input sequence encoding. The design choice was made as it was assumed that the most relevant information needed to predict future values lied in the recent past, given the relatively short period time of the kite. Furthermore, it enabled

manually controlling the memory capacity of the network which enabled a bounded Backpropagation Through Time (BPTT) cost and tuning of model complexity versus performance.

4.1.2.2 Model training and tuning

The design used, seen in Figure 4.3, allowed for flexible tuning of the RNN. Since the optimal prediction length horizon window L_p for the new controller was unknown, it was together with the sequence length L_s treated as hyper parameters. Although a longer prediction horizon in general was considered better since it allowed for more flexible control designs, it also came at a heavier training cost. Furthermore, the sequence length allowed for a bias-variance tradeoff. A too short sequence may result in the model not capturing and generalizing well to the expected periodicity of the signal, whereas a too long sequence risked overfitting and slows down computational speed.

The training consisted of a nested hyperparameter tuning procedure with an outer and inner loop. The outer loop iterated over sequence and horizon length pairs and created a supervised dataset using sliding windows. The RNN was tuned on 25 L_s and L_p pairs ranging from 10 to 100 steps, corresponding to 0.2 and 2 seconds respectively. For each L_p and L_s pair, the model was tuned on batch size, number of hidden neurons and learning rate. The configuration with the the lowest validation error was saved as the winning inner-tuning candidate.

4.1.2.3 Holdout test evaluation

The tuned models for each L_s, L_p pair were evaluated on the holdout test set using a common fixed evaluation batch size. Performance is calculated as the mean squared error (MSE) between predicted and true values over all samples and batches, shown in Figure 4.4.

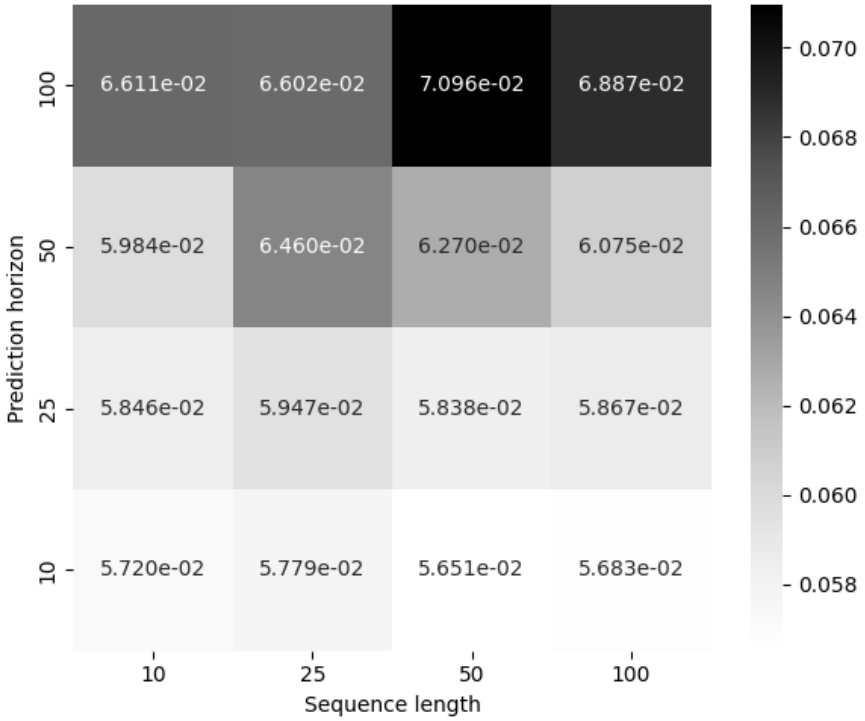


Figure 4.4: MSE test error for varying sequence and prediction length

As expected, the test error shows a dependence on the prediction horizon and increases with it for all sequence lengths, suggesting farther predictions are more difficult to predict. However, the test error is stable for varying sequence lengths. Increasing the sequence length does not indicate an improved predictive accuracy. Instead, the dynamics from recent past values are sufficient for capturing the periodic behavior and forecast future dynamics.

Since longer history sequences increase computational cost without a large difference in test accuracy, $L_s = 10$ was chosen for further control designs. Furthermore, the small test error difference between $L_p = 10$ and $L_p = 100$ was considered negligible for control performance. A prediction horizon of $L_p = 100$ was selected, as it provided increased flexibility in designing the generator controls. As seen in Figure 4.5, the RNN effectively generalizes to the high frequency noise in the calculated v_{rel} and captures the underlying periodic behavior of the signal.

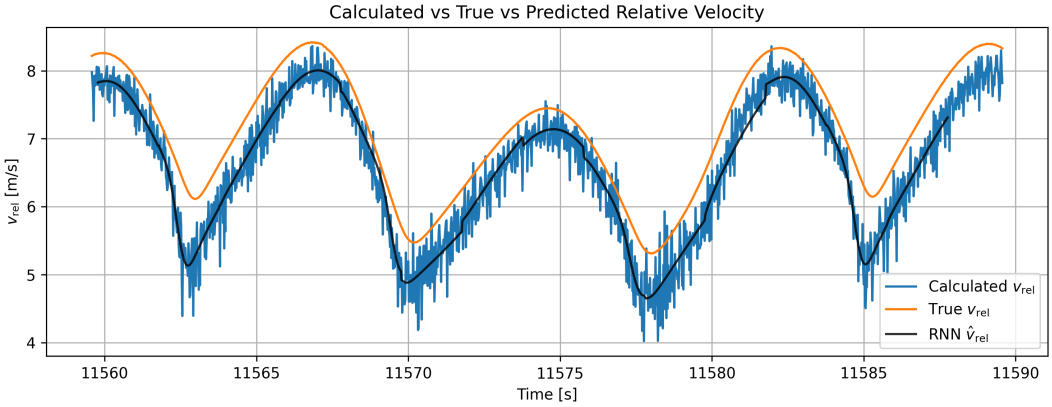


Figure 4.5: Tuned $L_s = 10$, $L_P = 100$ RNN model prediction versus calculated and true relative inflow velocity on holdout test set.

4.1.3 Generator control

Two generator control approaches were investigated to optimize power output while keeping the kite dynamics stable. The first control method was an extension of the baseline controller and added a predictive velocity derivative part. The second was a standalone controller that directly utilized the calculated inflow velocity and TSR-equation, see Equation 1.2, to get the optimal w_{ref} given a desired TSR.

4.1.3.1 Inflow gradient predictive controller

Extending the baseline controller with a predictive part was a natural method as the baseline controller was reactive to environmental changes and lacked anticipation. The extended controller included a variable that was proportional to the predicted change in relative inflow. It was expected that the new feature could reduce over- and undershoot in the generator control, either as a consequence of large inflow velocity changes or kite position in the figure-eight motion. As a result, extrema caused by noise or kite position were expected to get flattened out, giving a more stable and potentially increased power output.

The control consisted of three input readings, the reactive $S_r[t]$, stabilizing $S_s[t]$ and predictive $S_p[t]$ reading, each amplified by a constant gain k_1, k_2, k_3 respectively:

$$w_{ref}[t] = k_1 S_s[t] + k_2 S_r[t] + k_3 S_p[t], \tag{4.3}$$

where the predictive reading is,

$$S_p[t] = \frac{\hat{v}_{rel}[t + 1 + h_{idx}] - \hat{v}_{rel}[t + 1]}{h_{idx}\, dt}. \tag{4.4}$$

The controller was optimized by tuning the amplification constants k_1, k_2, k_3 along with the horizon length h_{idx}.

4.1.3.2 TSR predictive controller

The TSR-based predictive controller was introduced with the intention of finding a near constant optimal TSR for varying inflow. If hypothetically, a future inflow prediction was correct and the w_{ref} controller signal timed such that the generator got a speed at the instant we anticipated, then the controller would stay at a constant chosen optimal TSR_{opt} according to Equation (1.2).

$$w_{ref}[t] = \frac{TSR_{opt}}{r_{turb}} \hat{v}_{rel}[t + h_{idx}] N_{gear} k_4 \tag{4.5}$$

The controller was optimized by tuning the amplification constant k_4 for a given optimal TSR constant TSR_{opt}. The amplification constant was used to get the magnitude of $\hat{v}_{rel}$ correct. As noted during the preprocessing, see Section 4.1.1.3, the calculated inflow consistently underestimated the true v_{rel}.

4.1.3.3 Tuning control parameters

In order to find a control configuration that maximizes power while keeping the kite-turbine dynamics stable, a random search was made to explore the configuration landscape. This was followed by narrowing down the search space by detecting and excluding candidate configurations that could lack robustness and stability relative to the baseline controller. Several structurally distinct high-performing configurations was then selected for further investigation to avoid overfitting to a single parameter combination. Lastly, the top configurations were compared on different water conditions to finally pick the configuration that was stable and performed the best.

- **Initial random parameter search**. Both control methods were tuned by an initial random search of the control parameters, that is k_1, k_2, k_3, h_{idx} for the gradient and k_4, h_{idx} for the TSR predictive controller. A trial was performed for 100 seconds each parameter configuration under identical inflow conditions to identify well performing configurations. As seen in Figure 4.6, the random search quickly finds high performing configurations areas. It is assumed that the broad search space was explored after 1000 random parameter configurations for both controllers.

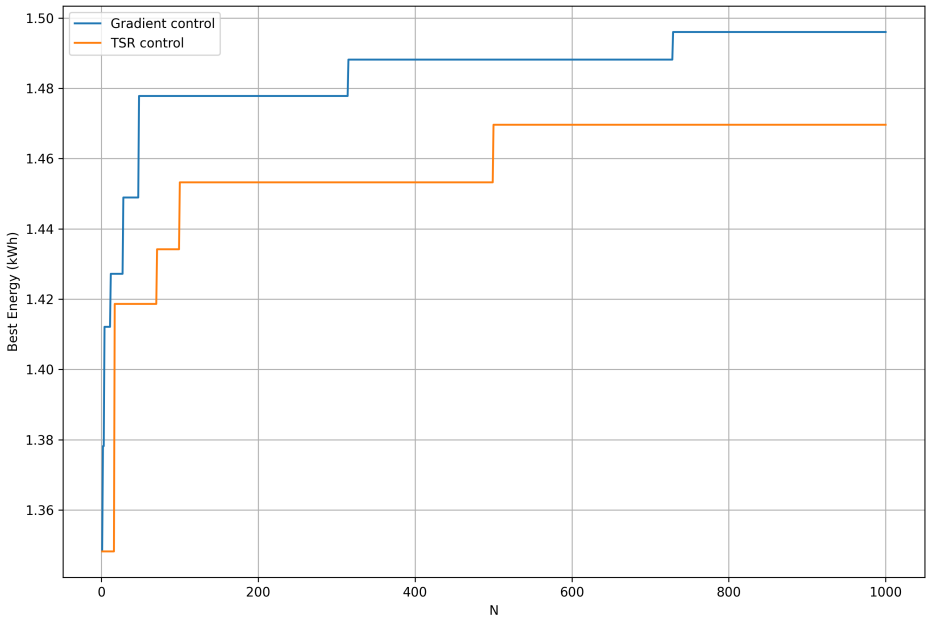


Figure 4.6: Initial random search best total power vs trials for different gradient and TSR predictive controller parameter configurations

- **Robustness constraints**. From the initial random parameter search, the top performing parameter configurations were selected for further investiga-

tion. Candidates were required to satisfy robustness constraints relative to the baseline controller, as the new ω_{ref} could cause a too aggressive or instable controller. Relevant metrics connected to the robustness of the new control signal were logged. Along with the ω_{ref} and ω, the TSR was logged to give insight into how the new control signal affects turbine-inflow behavior. Standard deviation was used as a measure to quantify the deviation from the baseline controller and what could potentially cause an instable controller, where a parameter configuration with a significantly higher standard deviation was flagged as potentially instable. The resulting top configurations after stability filtering are seen in Tables 4.1 and 4.2.

	$P_{tot}(kWh)$	k_3	$h_{idx,grad}$
1	1.4960	141.9853	59
2	1.4882	166.0038	86
3	1.4778	24.3665	76
4	1.4709	77.5683	67
5	1.4623	23.1798	95
6	1.4620	58.6075	82
7	1.4572	85.4858	35
8	1.4560	11.7697	56
9	1.4503	145.1539	40
10	1.4492	78.8852	76
11	1.4489	86.4186	10
12	1.4489	143.6125	71

Table 4.1: 100 second kite-run comparison of top performing parameter configurations for gradient predictive controller after stability filtering.

	$P_{tot}(kWh)$	k_4	$h_{idx,TSR}$
1	1.4696	1.1245	19
2	1.4532	0.9978	10
3	1.4446	1.1447	54
4	1.4342	1.1030	48
5	1.4311	1.0008	18
6	1.4307	1.1064	34
7	1.4300	1.1150	2
8	1.4282	1.1080	35
9	1.4275	1.0038	34

Table 4.2: 100 second kite-run comparison of top performing parameter configurations for TSR predictive controller after stability filtering.

- **Distinct unique configurations** Along with the parameter configuration with the highest total power P_{tot}, others that were high performing and showed clear deviation in parameter values were investigated further to avoid overfitting to a single configuration. Special attention was paid to deviations in the

predictive related constants, that is k_3, k_4, $h_{idx,grad}$ and $h_{idx,TSR}$. Three distinct configurations for each controller, highlighted green in Tables 4.1, 4.2 were selected for further tuning.

- **Final tuning** The best configurations were re-evaluated under multiple inflow conditions to reduce sensitivity to the tuning seed and an additional parameter search was performed. Final selection was based on mean extracted energy across seeds, giving the following parameter values: $k_3 = 89.21, k_4 = 1.128, h_{idx,grad} = 33, h_{idx,TSR} = 3$.

4.2 Soft Actor-Critic controller

The basics of SAC is described in Section 2.4 above. In this work the SAC implementation from the library Stable Baselines 3 (SB3)[1] is used as a starting point. Apart from the SAC algorithm, and equally important, is the environment in which the kite, water current, generator dynamics and reward function exists. The SAC algorithm uses this environment to interact with its surrounding and learn an optimal policy, as described in Section 2.4. Below follows a deeper description of the environment and SAC agent implementations.

4.2.1 Environment

The environment defines the world that the SAC agent uses to learn and is used for training and evaluation of the agents. It can be seen as a Markov Decision Process, where the agent takes a step, base on an action, from one state to another and receives a reward. Where the state and action space as well as the reward function needs to be defined within the environment. The environment uses the dynamical model of the system derived in Section 3.1 together with the Runge-Kutta solver to allow physics steps to be taken in the environment. The environment is created as a fully custom gymnasium environment[2] and follows the structure necessary structure described in Section 4.2.1.6.

4.2.1.1 State

The full state s_t gives a complete description of the current environment at a specified time. In this case the knowledge of the full state is not available and the system is therefore partially observable. Instead an observation o_t, that is a subset of the state, is available and was used in place of the state. That is, the system is a Partially Observable Markov Decision Process (POMDP).

The observation includes most of the features that the Minesto D4 kite observes during normal operation. Since some features, out of the full feature list, does not add any new information these could be removed to reduce the dimensionality of the observation space. One such feature is the three dimensional magnetometer measurements, which where removed since these describe the orientation of the kite, which is already well described with the heading feature. Therefore, a 14-dimensional observation space o_t was used. The individual measurements contained in o_t is not explicit written out in this thesis.

To note in the observation space is that it includes the reference speed for the generator $w_{gen,ref}$. This is the last action a_{t-1} that was taken in the environment and can ease the training procedure.

[1] `https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/sac/sac.py`

[2] `https://github.com/openai/gym`

4.2.1.2 Action

To interact with the environment and influence its trajectory the agent is allowed to determine its action a_t each controller time-step. Since the control variable of the kite system is the generator reference speed this is the action variable of the environment. Instead of allowing the agent to directly determine $\omega_{gen,ref,t}$ each time-step, where the action would directly be transformed into a reference speed within the allowed speed range, the action was instead modeled as a speed rate. That is, the action is the desired change in reference speed $\omega_{gen,ref,t}$ compared to the prior time step. Furthermore, to stabilize SAC training, the action a_t is in the range $[-1, 1]$ and scaled into the desired magnitude when used as the control signal. Resulting in the $\omega_{gen,ref,t}$ control signal given to the generator,

$$\omega_{gen,ref,t} = \omega_{gen,ref,(t-1)} + a_t \cdot \delta\omega_{\text{scale factor}}, \quad a_t \in [-1, 1]. \tag{4.6}$$

The resulting reference signal is then clipped to not contravene the generators speed limits. The action space is continuous in the given range, which is allowed due to SAC's stochastic policy. A positive a_t represents a command for increased generator speed while a negative represents a decrease. Actions close to zero results in a steady reference speed.

The reason for modeling the environment action as a rate change instead of an absolute value is to encourage exploration. When a rate change action was used the exploration of the $\omega_{gen,ref}$ space resembled random walk, while for an absolute action a white noise exploration pattern was observed.

4.2.1.3 Reward

The reward signal r_t indicates the performance of the action a_t in state s_t. It is used by the agent to improve its state-value functions and policy. The reward is determined within the environment based on current and prior observations and actions. It is based only on these since observations and actions is the only values available to the agent during operation. The shape of the reward function as well as the sparsity of rewards are important for the performance of the agents. It is usually preferred that the shape of the reward promotes both the optimal behavior as well as behaviors similar or in the direction of the optimal, which gives the agent a non-flat reward landscape. Furthermore, how often the reward is given is of importance. For example, is it given every time step, every other or just for the final time step [5].

To derive a reward function that rewards optimal behavior and results in a non-flat reward landscape it was constructed to consist of a reward part and a penalty part. The reward part promotes favorable behavior by returning the scaled and modified (see power smoothing and power baseline below) power generated by the generator P_{gen}. While the penalty component discourage unwanted behavior by returning a negative reward based on a couple of factors.

Positive reward The positive part of the reward function was modeled to directly reflect the generated power. Meaning that if more power is generated a higher re-

ward is achieved, which the SAC agent then strives for. This reward part was implemented to allow three setting, each adapted for use in different environment and results.

- **Base reward.** For the simplest setting the measured generated power P_{gen} was extracted from the sensor model and multiplied with the controllers time-step (the controller time-step is larger then the simulation time-step, resulting in intermediate values between the controller time-steps). This resulting reward the average power generated since the last time-step.

- **Power Smoothing.** In the base setting the power generation since the last time-step is based upon the power generated during one single time-step. This could allow the SAC agent to utilize this to generate high power at very short time duration with the purpose to maximize the reward. Therefore, a power smoothing function was implemented. It was constructed as a low-pass filter where a smooth power variable is updated with the current generated power at each simulation time-step,

$$P_{smooth} = \alpha P_{gen,t} + (1 - \alpha) P_{smooth}. \tag{4.7}$$

  Where $\alpha = \frac{dt_{sim}}{dt_{SAC}}$ so that P_{smooth} resembles an approximation of the actual mean power since the last controller time-step.

- **Power baseline.** In the case of a partially observable system where the surrounding water flow is stochastic and unknown to the agent a problem occurs. For the same observation and action pair the agent receives widely different rewards. This happens since the generated power, and consequently the reward, is highly dependent on the surrounding water flow, see Section 3.1.5.2, which is not included in the observation space o_t. This could possibly be harmful to the agent training and therefore a power baseline was implemented.

  The function of the power baseline was inspired by the method used in average-reward SAC. For which an averaged-reward value is subtracted from the reward at each time-step. Improve training stability and performance of the algorithm [10, 41, 2]. Our power baseline instead uses the relation between generate power P_{gen} and tether force F_{tether} to map F_{tether} to P_{gen},

$$P_{baseline} = \hat{P}_{gen} = f(F_{tether}) = a_0 F_{tether}^2 + a_1 F_{tether} + a_2. \tag{4.8}$$

  Where the coefficients a_0, a_1, a_2 were approximated by fitting a second degree polynomial to the F_{tether}, P_{gen} data points, see Figure 4.7. The data point were retrieved from simulations where the current controller was used. Figure 4.8a shows the baseline power obtained from the tether force, through Equation (4.8), compared to the actual power obtained. Comparing the power baseline in Figure 4.8b with the water current in Figure 4.8b. It is clear that the baseline follows the water current, high baseline when the water current is high and vice versa.

The baseline power $P_{baseline}$ is then subtracted to the measured power P_{gen} at each controller time-step resulting in a reward signal that is normalized to this baseline. Which in theory should mitigate the correlation between surrounding water flow and reward, and instead base rewards on the performance relative to the baseline.

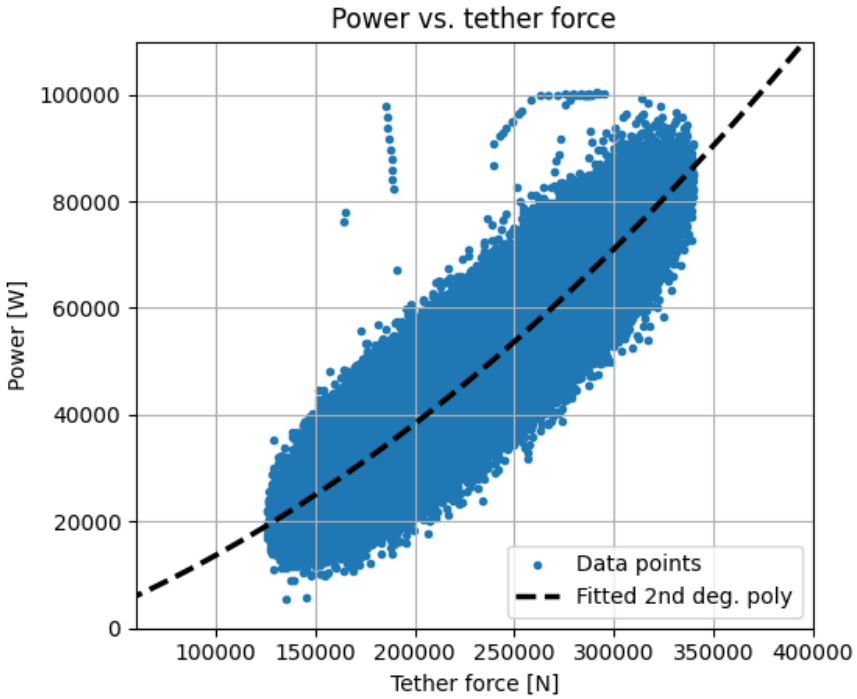
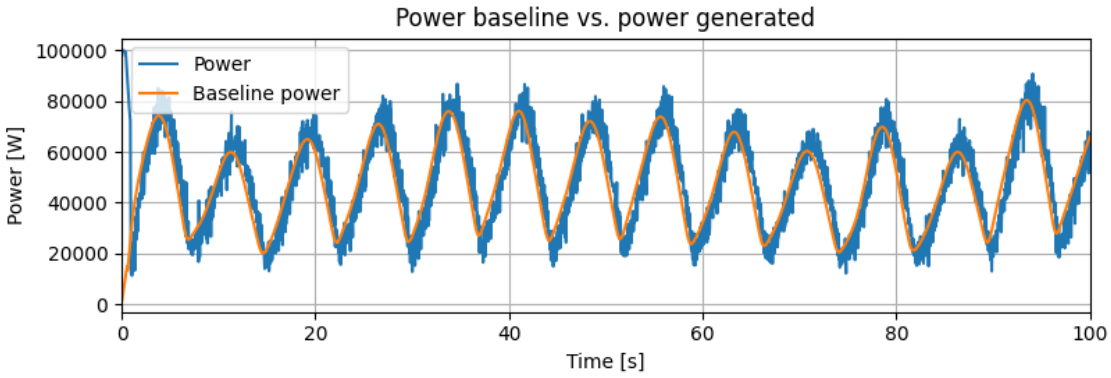


Figure 4.7: Fitment of the second degree polynomial mapping tether force to power. The large covariance of the main cluster is obtained due to the time shift between tether force and power. While the smaller clusters are classified as outliers and is generated at the initial transient state of the simulations.

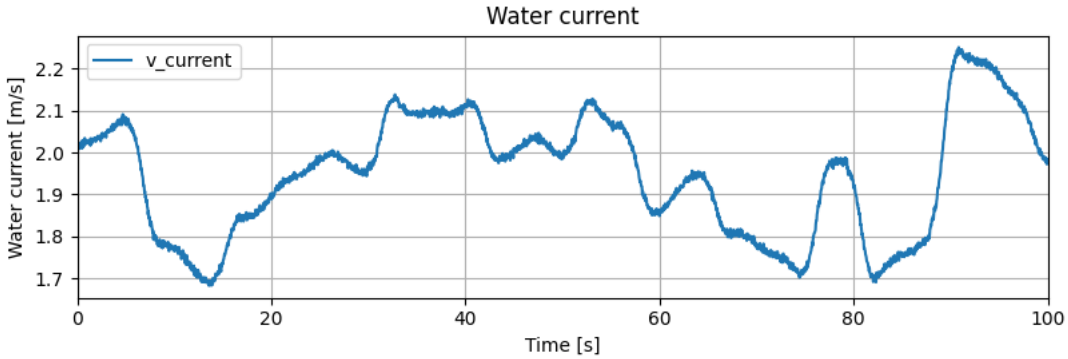
Negative reward (Penalties) The negative component of the reward function was designed to punish unwanted behaviors. Five penalty terms were implemented to target different behaviors,

- **Negative power.** The positive reward component is design in a way that allows it to be both negative and positive since it is directly dependent on the generated power. In the case of negative generated power, that is power is consumed from the grid, the generator is used as an electric motor and acts as a source of propulsion for the kite. This is not a sought after behavior during normal operation and was therefore penalized even more then just the negative reward received from the negative power. In addition a constant negative term was added to the reward for each time-step where $P_{gen} < 0\ kW$.

- **Generator speed violations.** Since the generator has specified limits in rotation speed the agent is not allowed to reach these speeds. If it does the



(a) Power baseline versus actual power



(b) Water current

Figure 4.8: In subplot **(a)** the fitted power baseline is compared to the actual power through time. Subplot **(b)** illustrates the water current during the same simulation. Comparing them, the power baselines relation to the unknown water current is clear.

run is terminated, see more below. To shape the reward landscape away from these limits two squared penalty terms were added, one for high speed and one for low speed. Both terms are similar and uses soft speed limits which are displaced by 500 RPM compared to the absolute limits. The difference between the actual generator speed ω_{gen} and these limits was squared and normalized to the range $[0, 1]$. A penalty value of 1 corresponding to the highest possible speed violation.

$$penalty_{overspeed} = max\left(0, \frac{\omega_{gen} - \omega_{softlim,upper}}{\omega_{hardlim,upper} - \omega_{softlim,upper}}\right)^2, \tag{4.9}$$

$$penalty_{underspeed} = max\left(0, \frac{\omega_{softlim,lower} - \omega_{gen}}{\omega_{softlim,lower} - \omega_{hardlim,lower}}\right)^2. \tag{4.10}$$

- **Absolute action size.** Large actions results in large and rapid change of the reference speed $\omega_{gen,ref}$ and consequently the actual generator speed ω_{gen}. This behavior does not necessarily result in inferior power generation, but can

result in an unstable, rapid and possibly hard controlled system. Therefore, a penalty term was added to directly punish large actions. The term is squared to punish larger actions more then smaller,

$$penalty_{action\ size} = a_t^2. \tag{4.11}$$

- **Action change.** To further disincentive the behavior presented above an additional action penalty term was designed. This term aims to discourages large changes in the actions between time-steps. Since the action a_t is a rate (the velocity of the control variable) the change in action can be seen as a kind of acceleration of the control variable $\omega_{gen,ref}$. With the same reasoning as above large changes in the action (acceleration) results could result in, similar to above, an unstable, rapid and possibly hard controlled system. Analogues to the above penalty term the action change term is squared,

$$penalty_{action\ change} = (a_t - a_{t-1})^2. \tag{4.12}$$

- **Reference divergence.** The final penalty term aims to reduce the divergence between the reference speed $\omega_{gen,ref}$ and the actual speed ω_{gen}. The reason for this is that large differences results in maximal generator torques T_{elec}, which, similar to above, can result in a rapid, unstable and hard to control system. Furthermore, if large divergences occurs while the reference signal is speeding up the generator, huge power losses could occur. Therefore, this term additionally aims to discourage such behavior.

$$penalty_{divergence} = \left(\frac{\omega_{gen,ref} - \omega_{gen}}{\omega_{hardlim,upper} - \omega_{hardlim,lower}} \right)^2, \tag{4.13}$$

  where the denominator normalizes the penalty term to range $[0, 1]$.

Each of the penalties above was scaled with independent scale factors to allow optimizing of reward function shape. All five scaled penalties were then subtracted from the positive component of the reward, resulting in the final reward r_t.

Reward scales To avoid large gradients during training the range of the per step reward was put to approximately $[-3, 3]$. This was not enforced through clipping, but instead through scaling of the positive reward part and all the penalties. Where each term, as mentioned, has a individual scale parameter. The scaling of each term allows for precise shaping of the reward function.

The value of each scale parameter was found by weighting the importance of each penalty as well as the focus on power generation versus mitigation of unwanted

behavior. Through trial-and-error of different distributions, all adding up to a maximum possible step-reward in the range $[-3, 3]$, different scale parameter combinations where established with widely different performance and behavior. The final combinations is covered in Section 4.2.3.1.

4.2.1.4 Environment wrappers

To keep the base environment clean environment wrappers were used to modify different aspect of the base environment [40]. In total four environment wrappers were used. Two were created specifically for the kite control system while the other two were predefined gymnasium wrappers[3].

- **Normalization wrapper.** The first one was implemented as an observation normalization wrapper. This wrapper was implemented as max-min clipping normalizer. It initially clip the observation based on the maximum and minimum value and then normalize each observation independently into the range $[-1, 1]$ based on the observations current, maximum and minimum value.

$$o_t^i = \frac{2 \cdot (clip(o_t^i, o_{min}^i, o_{max}^i) - o_{min}^i)}{o_{max}^i - o_{min}^i} - 1, \quad \text{for observations } o^i,\ i \in [0, 14], \tag{4.14}$$

  where the observation specific limits was set up wide enough to mostly avoid clipping and tight enough to enforce normalized observation widely spread in the range $[-1, 1]$.

- **Physics-logging wrapper.** The other implemented wrapper is a read-only wrapper used to log observation and other kite system value during operation. The wrapper temporary stores the data every pre-determined step and at the end of the episode the time-series data is written to a local file. Which allows debugging and visualization of all values from the environment, SAC agent and kite system during training and evaluation.

- **Gymnasium monitor wrapper.** The first pre-implemented wrapper used is the Monitor wrapper. It is used to monitor RL training specific values during the training procedure. For example, episode reward, length and total elapsed time [40].

- **Gymnasium frame stacking wrapper.** The final environment wrapper used is the gymnasium frame stacking wrapper. The wrapper concatenates the observation o_t with the observations obtained at k prior time steps, that is $o_t = [o_{t-k}, \cdots, o_{t-1}, o_t]$. Appending earlier observations in this manner is done to insert information about feature velocities and accelerations into the observation [20].

[3] `https://github.com/openai/gym/tree/master/gym/wrappers`

4.2.1.5 Termination conditions

In the environment there exists two types of termination conditions, terminations and truncations. Both types are endpoints of the MDP and have the same effect in that if any is fulfilled the episode is ended and a new is initiated. Terminations are usually activated if a system limit is breached and, in the real system, continued operation could result in damage. Three termination condition were used in this work, each indicating that a unsafe system state is reached. The first two checks the generator speed ω_{gen} for over and under speed. If ω_{gen} is greater than $\omega_{hardlim,upper}$ or lower then $\omega_{hardlim,lower}$ this termination condition is satisfied. The last termination condition checks for low kite speed, that is if $\dot{p}$ is less then or equal to zero. While this condition does not have a comparable real system risk, it do result in an unstable simulation point and is therefore to be avoided.

The second type of termination condition is called truncation. Truncations does not indicate any system violation but instead that the maximum episode time is reached and that a new episode is to begin. The reason for two different types of termination condition is to allow the RL agent to distinguish between undesirable system violations and natural time limits.

4.2.1.6 Environment structure

The usage of the custom environment follows the steps of a RL algorithm. That is, the initial environment state is given, the agent chooses an action. The environment reacts to the action and returns the corresponding reward as well as the next state, and the pattern is repeated until a termination condition is fulfilled. In the gymnasium framework this algorithm is represented in three functions, initialize, reset and step.

- **Initialize** is only executed once and simply initialize all environment and kite system variables and parameters. One example is the shape and limits of the action and state space representations. Here the Random Number Generator (RNG), consistently used in all stochastic parts of the environment, is created. The reason for a consistent RNG is to allow reproducibility in the environment.

- **Reset** corresponds to the first step in a basic RL algorithm. Here the initial position is randomly set. This is done by initializing a stochastic water current, see Section 3.1.7, with the environment specific RNG. Then randomizing each state of the dynamical model including position p, velocity $\dot{p}$, generator angular velocity ω_{gen} and the generator torque $T_{gen,el}$. Where each is uniformly randomized within its specific operation range. However, to avoid initializing the position at the path singularity (see Section 3.1.1.4), the initial position is randomized until a value outside the singularity region is received. Randomizing the kite environment ensures that the learned RL agent generalizes and this improves performance on unseen states and when porting the agent to the real world [29]. Furthermore, the reset function steps the kite dynamics two simulation time-steps using the RK4 solver. This is done to prime the

environment by simulating, in a controlled manner, the first transient time where rapid fluctuations in states occur due to random initial conditions. But also to receive a first batch of observation to be used by the SAC agent to determine the first action.

- **Step** is the dominant function of the environment. Given an action and the prior state the step function rescales the action, simulates the kite and water dynamics, determines termination conditions and calculates the reward for the given step-action pair. Similarly to the reset function, the step function simulates the kite dynamics with the RK4 solver. To reduce computation time and allow a longer reward horizon, frame-skipping is implemented [16, 27]. Where for each environment step (controller time-step) d physics time-steps are executed with the same action signal, where $d > 1$. In our work d is set to four meaning that for every control/environment step four physics steps are taken with the RK4 solver. This is further described in Section 4.2.2.5 below. For each function call the reward is calculated and the termination conditions checked following the description above. If no termination condition are fulfilled the step function will prompt the execution of a new step function call, continuing until a termination condition is satisfied.

4.2.2 SAC implementation

The SAC algorithm used in this work was adopted from the Stable Baselines 3 SAC implementation[4]. The relevant theory regarding the algorithm is presented in Section 2.4 above. Our expanded SAC implementation concerns adaptations to reduce negative impact of partial observability, enhance training efficiency and fitting SAC to the custom kite environment. Below is a description of the SAC design, based upon the math derived in Section 2.4, and its training process. A simplified diagram of the interaction between the major SAC subsystems is shown in Figure 4.9.

4.2.2.1 Network structures

The SAC implementation contains five neural networks, one in the actor and four in the critic. These are continuously updated through stochastic gradient descent and constitutes the core of the SAC algorithm. Each network was created with the same hidden neuron structure. While the output layer differs between policy and action-value networks.

The actor's policy network $\pi_\phi(a_t|o_t)$ takes as input the current observation vector o_t. The input is connected with three fully connected 256 neuron layers with leakyReLU activation functions. The final layer consists of two output neurons with hyperbolical tangent activation functions. Where the hyperbolical tangent activation functions ensures that the output of the policy network, the actions mean and

[4] `https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/sac/sac.py`

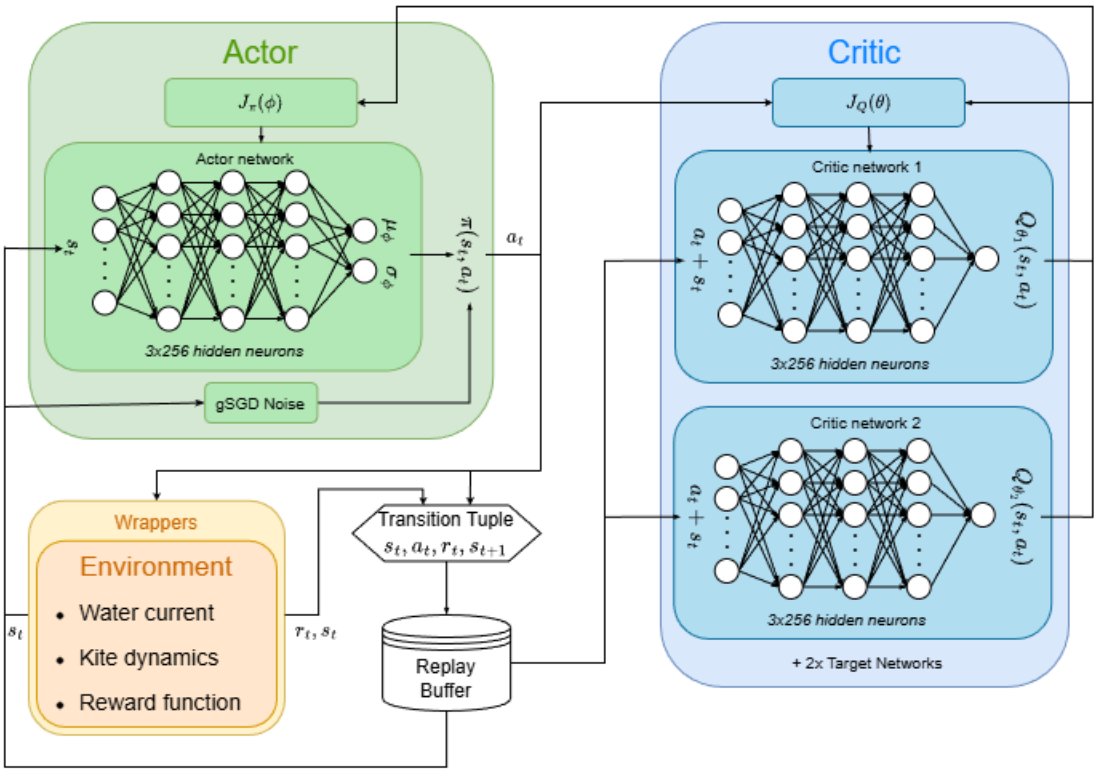


Figure 4.9: Simplified visualization of SAC's subsystems, Actor, Critic, Environment and Replay buffer, and their cooperation. The diagram is not a complete representation and disregards the critic's target networks, their connections and update rules as well as some loss function dependencies.

standard deviation, is within range $[-1, 1]$. Resulting in the stochastic policy found in Equation (2.23).

The critic's two action-value networks $Q_\theta(s_t, a_t)$ converts a state-action pair to a scalar value. The critic's networks has the same hidden structure as the policy network. The input layer of the critic's networks consist of the concatenated observation vector o_t and action vector a_t. The output layer is a single neuron without activation function, allowing the networks to output a scalar value, positive or negative, for the given state-action pair. The two action-value target networks has identical structure as they follow the exact weights from the primary action-value networks.

The incorporation of leaky ReLU in place of the commonly used ReLU activation functions was to mitigate deterioration of performance during longer training periods, likely caused by inactive neurons [44].

4.2.2.2 Action noise

As described in Section 2.4, the entropy term ensures sufficient exploration through incorporation of the policy entropy term. The entropy coefficient α included in the objective function, Equation (2.17), was initialized to 0.7 and automatic tuning through the entropy term objective function. The target entropy is the entropy the

objective function aims to reach and in practice determines how small the action noise is allowed to be. The target entropy was set to -0.5.

To allow the SAC agent to explore, a stochastic policy is used, see Equation (2.23). Where the stochasticity originates from the gaussian sample $\mathcal{N}(0,1)$ and the magnitude of it is determined by the policy network through $\sigma_\phi(s_t)$. Combined with the environment's rate action, a random walk exploration pattern is obtained. However, the rapid fluctuations in action, and consequently in $\omega_{gen,ref}$, results in weak exploration since the gaussian exploration noise of $\omega_{gen,ref}$ is reduced due to the slower dynamics of the generator-turbine shaft and the electrical torque dynamics. That is, the exploration noise is low-pass filtered by the system dynamics. To counteract this a more sophisticated noise model was adopted, the so called generalized State Dependent Exploration (gSDE). Where the gaussian noise $\mathcal{N}(0,1)$ is replaced by a state dependent noise model $\epsilon(o_t, \theta_\epsilon)$, resulting in policy,

$$\pi_\phi(a_t|o_t) = \mu_\phi(o_t) + \sigma_\phi(o_t)\epsilon(o_t, \theta_\epsilon). \tag{4.15}$$

This noise model is deterministic, meaning that a (o_t, θ_ϵ)-pair will always result in the same value. The policy's stochasticity instead stems from that $\theta_\epsilon \sim \mathcal{N}(0,1)$, which is only sampled once every episode to obtain an action noise that explores less sporadically [32].

4.2.2.3 Training settings

Training the policy and action-value networks is done through stochastic gradient decent using the objective functions described in section 2.4. Where each network is trained using ADAM with a scheduled learning rate. The learning rate was initialized at $5e^{-5}$ and linearly decreased to zero at the end of training. A decreasing linear schedule was incorporated to stabilize training at later training stages by reducing large fluctuations of policy and action-value networks. This is also the reason for the relatively low initial learning rate. The maximum allowed training steps was set to $7e^5$. The number of training steps indicating how many steps are executed with the SAC algorithm. As mentioned above four times more physics steps were taken due to frame-skipping.

The entropy term α follows a similar training procedure. Where α is updated through stochastic gradient accent with a specific entropy objective. The stochastic gradient accent was also conducted with ADAM and with identical learning rate schedule.

The two target networks of the action-value networks were updated with polyake averaging. Meaning that the old target network weights and the new action-value network weights are average with a fraction factor τ. Which results in updated target networks,

$$\theta_{target} = \theta \cdot \tau + \theta_{target} \cdot (1 - \tau). \tag{4.16}$$

Where τ was set to 0.005 to get slow moving targets, stabilizing the training process.

4.2.2.4 Buffer settings

The replay buffer, responsible for storage of transition tuples, was given a max length of $1e^6$. Since the buffer size is greater then the number of total SAC steps all transition tuples can be stored and randomly sampled during training.

Instead of the standard replay buffer type, which only stores the tuples, we opted to use the NStepReplayBuffer[5]. Similarly to the standard buffer it stores each transition in the order they were obtained. However, during sampling of transitions from the buffer the NStepReplayBuffer uses n_{step} transitions following the sampled one to calculate the cumulative discounted reward for the sampled transition. That is, the reward in the transition tuple is based on the reward of the following transitions,

$$r_t = \sum_{i=0}^{n_{step}} \gamma^i r_{t+i}. \tag{4.17}$$

Where n_{step} was set to 10. This effects the bootstrapping used to calculate the action-value function in Equation (2.20), in that the current and n_{step} following rewards are used before the recursive relation of the action-value function is utilized,

$$Q^\pi(s_t, a_t) = \underset{\substack{(a_t)\sim\pi \\ (s_t, r_t)\sim D}}{E} \left[\sum_{i=0}^{n_{step}-1} \gamma^i r_{t+i} + \gamma^{n_{step}} [Q^\pi(s_{t+n_{step}}, a_{t+n_{step}}) - \alpha \, log \, \pi(a_{t+n_{step}}, s_{t+n_{step}})] \right]. \tag{4.18}$$

That is, n-step bootstrapping is used instead of single-step bootstrapping. Resulting in improved SAC performance on POMDPs [25].

Prior to training the replay buffer was filled with $5e^4$ transition tuples obtained from the current controller. These transitions were obtained by using the original generator controller to determine actions within the custom gymnasium environment. For each step the transitions were stored and inserted into the empty replay buffer. These prefilled transition tuples greatly dominates the stochastic gradient decent during early SAC training and should result in a policy with similarities to the current controller. Hence, acting as a sort of basic offline RL solution.

4.2.2.5 Partially observability

It has been established that the simulated kite environment given the features in the observation o_t, is not fully observable. That is, the environment is a POMDP. Since SAC, and most other DRL algorithms, are designed for MDP environments, suboptimal performance is obtained in POMDP environments [25]. Meaning that

[5] `https://github.com/DLR-RM/stable-baselines3/blob/master/stable_baselines3/common/buffers.py`

inferior performance was expected training the standard SAC algorithm on the custom kite environment. To mitigate the performance loss due to partial observability three main remedying improvements were implemented, n-step bootstrapping, frame-stacking and frame-skipping.

N-step bootstrapping was implemented through the NstepReplayBuffer, see Section 4.2.2.4. This is shown to improve SAC performance on POMDP environments by including temporal information into the action-value estimation, effectively reducing the SAC algorithms sensitivity to POMDP environments [38]. Inclusion of temporal information makes the environment more observable since the current state-action pair better describes the following state. That is, the transition is more accurately described as $p(o_{t+1}|a_{t+1})$, see Section 2.2.

The second improvement implemented was frame-stacking, introduced in Section 4.2.1.4. By stacking the current observation with the previous 9 observations, temporal information in the form of feature changes over time is introduced to the agent, improving its performance [20, 25]. For example, stacked observations gives the agent information about the change in F_{tether} which can increase its predictive capabilities. Furthermore, appending more frames gives the agent access to more distant data, possibly allowing for better performance.

Lastly, frame-skipping is used, see Section 4.2.1.6. While not introducing any remedy to the POMDP issue one its own, it synergizes with the above mentioned methods to further improve SAC's performance on the POMDP kite environment. By increasing the time between adjacent observations the system is given adequate time to reach a new state with enough difference compared to the old state [16]. If features of the two adjacent states are too similar, no information can be drawn from the transition. Furthermore, an increased frame-skip, similarly to frame-stacking, makes more distant observations available to the agent further extending the time horizon available to the agent.

4.2.3 Training

The training procedure for the SAC agent involves stepping through the environment and executing the SAC algorithm once every step. Meaning that for each environment step, the SAC network parameters are updated through one step of stochastic gradient decent. The gradient decent is based on the different objective functions presented in Section 2.4. They use the step dependent learning rate schedule and a batch of 256 transitions sampled from the NStepReplayBuffer. Where each sampled transition is connected to the following n_{step} transitions as described above.

Since the SAC training process does not guarantee increased performance with increased number of training steps, a callback for saving the best seen SAC agents was implemented. The callback continuously monitors the per episode cumulative return. If the average episode return obtained during the last three episodes is greater then the all-time best, the SAC agent is stored and the all-time best is replaced.

The callback utilize the data stored by the gymnasium Monitor wrapper introduced above. Moreover, the final agent obtained during training was also stored.

To store a SAC agent all networks, settings and buffer transitions were stored to file. This ensures that training can be resumed without any interruptions directly from the stored agent. However, to use the agent for evaluation only the policy/actor network is required.

As mentioned above, the custom environment is partially observable, a POMDP, meaning that performance is reduced. The observability problem could be even worse with the introduction of stochasticity in sensors and water. Therefore, independent SAC agents were trained on different environment cases to visualize this phenomenon.

4.2.3.1 Models

To show the potential of SAC three different primary SAC agents were developed. The first two models were trained on the stochastic environment. The stochastic environment being the environment most similar to the real kite system. It includes the stochastic water function implemented in Section 3.1.7 and gaussian noise added to each feature in the observation, see Section 3.1.6. While the last agent was trained on a deterministic environment, which excludes the mentioned sources of stochasticity.

- **Agent 1 (Smooth control):** This agent was implemented with a well balanced reward function, aiming to generate maximal power while avoiding rapid or large changes in the reference signal. Since this agent was used in the stochastic environment it utilizes the power baseline reward to mitigate agent unknown reward fluctuations.

- **Agent 2 (Power maximization):** Was created to heavily focus on power generation. Meaning that the reward function was weighted toward power generation and away from smooth control signals. Similar to Agent 1, Agent 2 utilize the power baseline reward. However, it does not use power smoothing since maximizing the instantaneous reward better aims the agent toward power maximization behaviors.

- **Agent 3 (Deterministic):** The final agent was created to show SAC's performance on a more deterministic environment, that is, more observable and closer to a true MDP. This agent has similar reward distribution between power reward and penalties as Agent 1. It does not use the power baseline reward since the water current is constant in the deterministic environment. The usage of absolute power generation instead of the baseline difference, results in a reduced power reward scale to obtain a similar reward-penalty balance.

4.3 Evaluation

Evaluation of controller performance was conducted by running each controller in the environment. Since the environment is randomly initialized the same controller will perform differently from environment to environment. Therefore, each controller was evaluated on the same 15 evaluation environments, where each evaluation environment generates a random initial position, water current and sensor noise depending on the seed of the environment's RNG. Evaluating the controllers on multiple predetermined environment ensures that an average performance is obtained. Furthermore, since each controller is evaluated on the same evaluation environment it is ensured that all controllers are tested with the same conditions, allowing for a fair comparison. The performance graphs obtained evaluating each controller on the same environments was plotted. Furthermore, numerical statistics regarding the performance was also presented. This allowed deep analysis and comparison of the different controllers in a fair environment.

For the SAC controllers, the policy is executed without any action noise. That is, the policy network is used but the deterministic mean μ_ϕ controls the action directly,

$$a_t = \mu_\phi(o_t). \tag{4.19}$$

The reason for the removal of the randomness is because the action noise is only needed during training to explore different policies. The optimal policy is usually the mean policy. To examine the SAC algorithm's performance on different environment structures the above evaluation method was repeated for a deterministic environment as described in Section 4.2.3.1.

5
Results

This chapter shows the main results of the thesis connected to the aim of the project. To find a generator speed control signal that increases the power generated by the kite. Results seek to show how the developed controllers performed and their unique characteristics. This is done by first showing an overview of all the controllers, to then dive deeper into each controller type and their specific behaviors.

5.1 Comparison to baseline controller

To compare the developed controllers to the original baseline controller, three main result types are presented. A power-per-episode plot is shown to compare the controller's power generative abilities under different circumstances, see Figure 5.1. A table containing important statistics for each controller to expose their behavior, see Table 5.1. Finally, multiple time plots from a specific episode are shown to visualize how the different controllers act, see Figure 5.2.

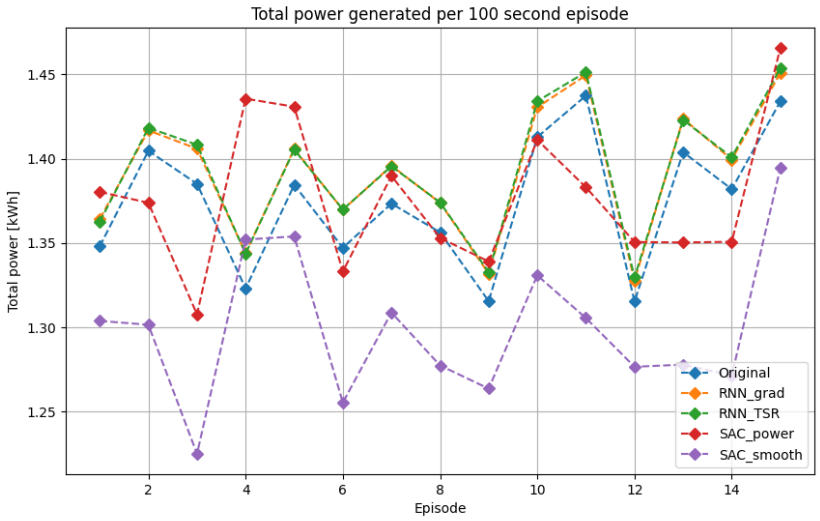


Figure 5.1: Total power generated by each control for 15 distinguished episodes, each with 100 second simulation time. For each episode on the x-axis, each controller was given the same environment with identical circumstances. None of the 15 environment were previously used during training.

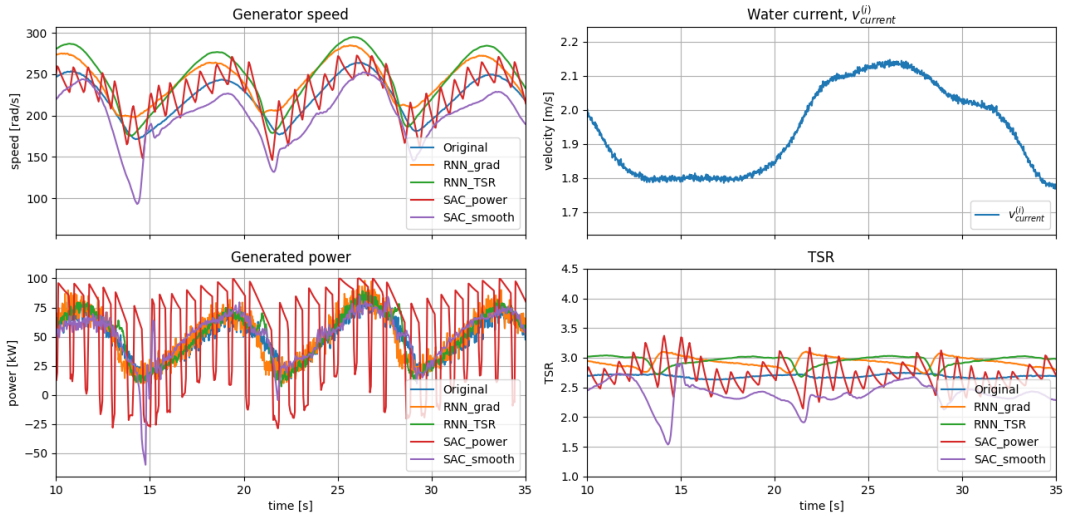


Figure 5.2: Time series plots of the generator speed ω_{gen}, generated power P_{gen} and tip-speed ratio for each controller type. The corresponding water current $v_{current}^{(i)}$, identical for all five controllers are also visualized. The time series plots shows the features for each controller for a 25 second time-span, for one of the episodes presented in Figure 5.1.

5.1.1 Performance and behavior overview

From Figure 5.1, we see that both RNN-based controllers consistently shows very similar performance and outperforms the baseline controller for all unique environment conditions, showing on average a 1.3% increase in power generation. It is also seen that the total power generated per episode is similar for similar environments for the RNN-based and the original controller. The SAC controllers on the other hand, does not follow the same pattern. They have similar performance to each other on similar environments, but with a clear performance increase for the power maximizing SAC agent. While the power maximizing controller often outperforms the baseline controller, and occasionally outperforms all other controllers, the smooth SAC agent seems to lack behind in power generation performance for most inflow conditions. These finding are confirmed by the statistics seen in the power column of Table 5.1.

A short sequence of the time-dependent behavior of the controllers for generator speed, generated power and TSR are shown in Figure 5.2. The generator speed plot, reveals that the RNN-based controllers follow a similar shape to the original, while the SAC controllers have found unique trajectories. All controllers does however, interestingly, share a similar mean, amplitude and period. This behavior is further explained and discussed in the following sections.

The TSR plot reveals that most controllers operate in the same TSR-region. The predictive controllers shows different TSR trajectories but seems to generally operate at a higher TSR than the baseline. The power maximizing SAC agent shows some notable rapid changes in TSR while the smooth agent seems to operate at a TSR

below the other controllers. All controllers except the power maximizing SAC agent seems to generally follow the baselines generated power curve.

Table 5.1: Statistics of controllers across the 15 episodes shown in Figure 5.1. For each controller, the table shows the mean and standard deviation of the gausian distributed generated power P_{gen}, tip-speed ratio (TSR), and generator speed ω_{gen}. For the generator efficiency η_{gen}, reference speed variation $\Delta\omega_{gen,ref}$, and tracking error $(\omega_{gen,ref} - \omega_{gen})$, percentile-based statistics are used due to their non-Gaussian distributions.

Controller		P_{gen} [kW]	TSR	η_{gen}	ω_{gen} [rad/s]	$\Delta\omega_{gen,ref}$ [rad/s]	$\omega_{gen,ref} - \omega_{gen}$ [rad/s]
Original	Mean	49.495	2.701	–	222.032	–	–
	Std	19.663	0.19	–	27.456	–	–
	Median	–	–	0.926	–	0.357	1.469
	25 %	–	–	0.918	–	0.168	0.737
	75 %	–	–	0.93	–	0.61	2.295
	5 %	–	–	0.907	–	0.033	0.146
	95 %	–	–	0.932	–	1.038	3.959
RNN grad	Mean	50.135	2.944	–	243.377	–	–
	Std	22.872	0.199	–	26.935	–	–
	Median	–	–	0.925	–	0.366	1.787
	25 %	–	–	0.913	–	0.173	0.983
	75 %	–	–	0.93	–	0.623	2.586
	5 %	–	–	0.897	–	0.033	0.212
	95 %	–	–	0.932	–	1.06	4.09
RNN tsr	Mean	50.166	2.955	–	245.439	–	–
	Std	23.342	0.203	–	34.972	–	–
	Median	–	–	0.926	–	0.26	1.737
	25 %	–	–	0.913	–	0.135	1.018
	75 %	–	–	0.93	–	0.407	2.477
	5 %	–	–	0.897	–	0.028	0.223
	95 %	–	–	0.936	–	0.761	4.178
SAC power	Mean	49.561	2.727	–	222.906	–	–
	Std	50.843	0.278	–	28.706	–	–
	Median	–	–	0.933	–	1.423	20.642
	25 %	–	–	0.842	–	0.854	4.855
	75 %	–	–	0.934	–	1.966	26.89
	5 %	–	–	0.842	–	0.19	0.74
	95 %	–	–	0.934	–	2.992	34.71
SAC smooth	Mean	46.787	2.385	–	194.151	–	–
	Std	22.528	0.297	–	33.761	–	–
	Median	–	–	0.927	–	0.241	1.754
	25 %	–	–	0.921	–	0.108	0.968
	75 %	–	–	0.93	–	0.482	2.971
	5 %	–	–	0.842	–	0.022	0.218
	95 %	–	–	0.932	–	1.003	6.602

X% indicating the Xth percentile. It is shown in place of standard deviation when the distribution is not bell shaped.

5.2 Predictive models comparison

This section focuses on comparing the predictive controllers with each other and the baseline controller. An interpretation of the tuned configuration for each controller is given, followed by a results focused analysis of the controllers. The tuned parameter configurations for the controllers, obtained in Section 4.1.3.3, was used for all result metrics.

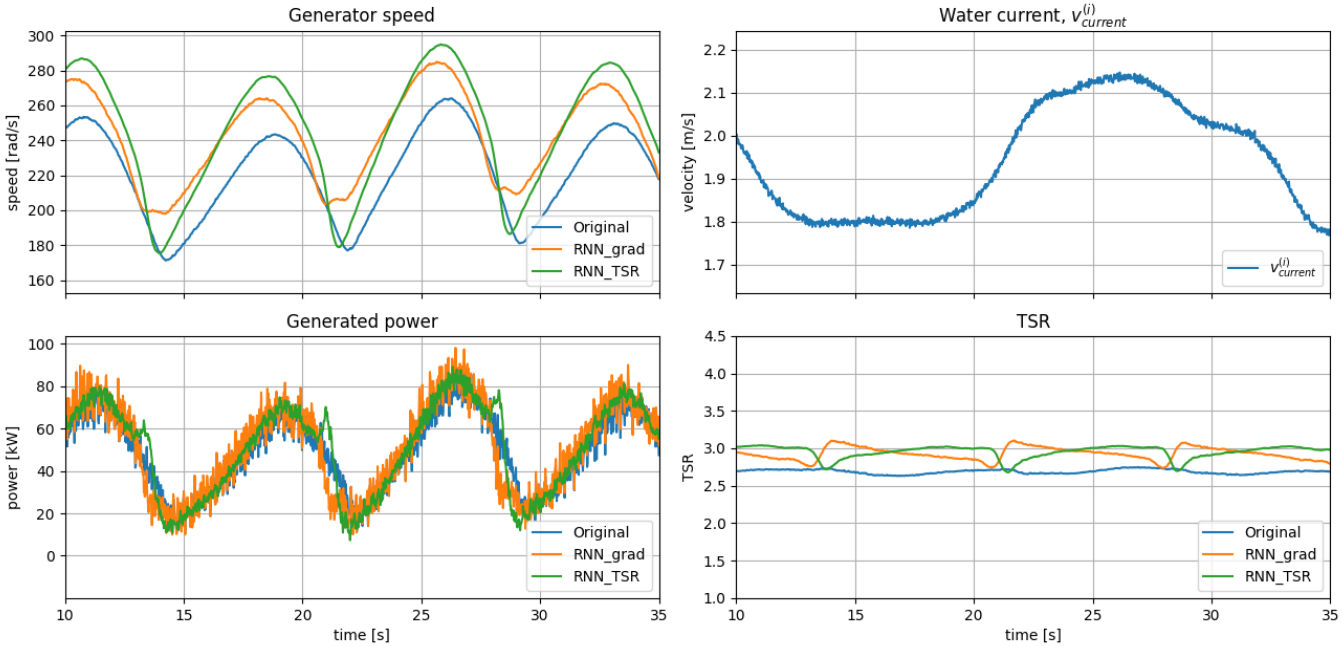


Figure 5.3: Time series plots of the generator speed ω_{gen}, generated power P_{gen} and tip-speed ratio for both predictive- and the baseline controller. A short 25 second time-span of a simulation is shown.

5.2.1 Interpreting the tuned predictive controllers

The control parameter tuning in Section 4.1.3.3 focused on maximizing the power output of the predictive controllers. The gradient predictive controller was expected to be tuned to balance its stabilizing, reactive and predictive parts for power optimization and use the horizon index as a look-ahead. The TSR predictive controller on the other hand, was expected to choose a horizon index to mitigate inertial effects, and a amplification constant to match the real relative velocity. As a result of this tuning, we expected to see a stable TSR trajectory very close to the optimal TSR value.

5.2.1.1 Gradient predictive controller

The tuned gradient predictive controller used a horizon index of 33 steps which corresponds to 0.66 seconds and a gain of 89.2. The generator speed plot in Figure 5.3 reveals the reference signals effects. As expected, the added predictive term

causes the controller to smooth out the extrema in ω_{gen} when comparing it to the baseline controller. This is most evident in the local minima of the generator speed, and is a result of the predictive parts contribution increasing when the reactive part, being dependent on tether force, decreases. It is also evident from the generator speed plot, that the predictive part makes the RNN_grad signal lead in phase compared to the original baseline which is a result of reduced lag time between inflow changes and turbine response. From this behavior, the results show that the tuned controllers predictive gain of 89.2 made it have a significant predictive contribution on the final reference signal.

5.2.1.2 TSR predictive controller

The TSR predictive controller used a horizon index of 3 and a gain of 1.128 which corresponds to a 0.06 second velocity look-ahead and a 12.8% added gain of the calculated v_{rel} to match the actual relative velocity. The horizon index was expected to be larger than 3 and a 12.8% difference between the calculated and true v_{rel} seemed large. It could suggest that the system did not have a lot of inertia in regards to obtaining a desired w_{ref} and that a 0.06 second future anticipation was perfect. Furthermore, the power losses that caused the underestimating calculated relative velocity could be more severe than anticipated and perhaps a 12.8% gain was reasonable.

On the other hand, it could hint that the tuned controller was not behaving as anticipated. The TSR plot in Figure 5.3 reveals that the controller did not make the TSR stay near the optimal $TSR_{opt} = 2.7$. Instead, the controller made the turbine operate at a much higher TSR, averaging 2.955, see Table 5.1, which explains the high gain. The 12.8% gain could be a result of a too low expected optimal power TSR value, taking the full coupled system into account. Furthermore, the horizon index was not tuned to mitigate inertial effects, even though this was expected to be the result of the tuning. Given the unexpected TSR behavior, it cannot be concluded from the results that the 0.06 second look-ahead corresponds to the inertial effects. The controller is a result of not tuning to optimal TSR and finding the configuration that maximized power in the simulation setting.

5.2.2 TSR Operating point

As seen in Figure 5.1, the gradient and TSR based predictive controllers are consistently very similar in total energy harvested from the water across different stochastic inflow conditions. This could suggests that both control methods find a similar operating point where power is optimized. However, the generator speed and TSR plot in Figure 5.3 tells a different story. The generator speed of RNN_TSR has a larger maxima and a lower minima compared to the RNN_grad controller. This results in a TSR plot were the two controllers cross each other at a point were the RNN_grad controller TSR increases due to the flattened generator speed, and the RNN_TSR keeps decreasing. Hence, the controllers both maximize the power to be in the same regime, but in very different ways. This suggest that the energy

harvesting optimum is not unique in regards to a TSR trajectory.

It is also noted that both predictive controllers operate at a mean TSR higher than the baselines 2.701, which could explain the power increase. While the baseline controller operates at a near optimal point according to the power coefficient C_p curve, seen in Figure 3.7, the small TSR increase might not be enough to punish the predictive controllers enough. Similarly, a increased TSR indicates a increased ω operating point which could decrease the generator efficient η_{gen} for high RPMs, but it does not seem to be the case. The generator efficiencies η_{gen} are almost identical across controllers and percentiles in Table 5.1. As such, a TSR value higher than the anticipated local turbine power optima might generate more power in the coupled system. These findings raises an important question on the TSR optima operating point for the coupled system. While the controllers operate in the same TSR regime, isolating the turbine to itself and staying close to the turbine power coefficient C_p curve optima is not the most effective for power maximization in the simulation setting.

5.3 SAC models

By altering the environment and SAC's parameters, different behaviors and performance was obtained from the SAC algorithm. Furthermore, by decreasing the stochasticity of the environment, better performance was seen. Below the results from three different SAC settings are shown.

5.3.1 Stochastic environment

The stochastic environment, introduced above, is the central environment used to evaluate different turbine controllers in this thesis. Two SAC agents were trained in this environment with different functionality. The first was modeled to smoothly control $\omega_{gen,ref}$ through a reward function with power smoothing and a power baseline, Agent 1. The parameters was set to prioritize a smooth reference signal. The second agent, Agent 2, was instead aimed to maximize instant power by removing power smoothing and through parameters set to prioritize high power output. As described in Section 4.2.3.1.

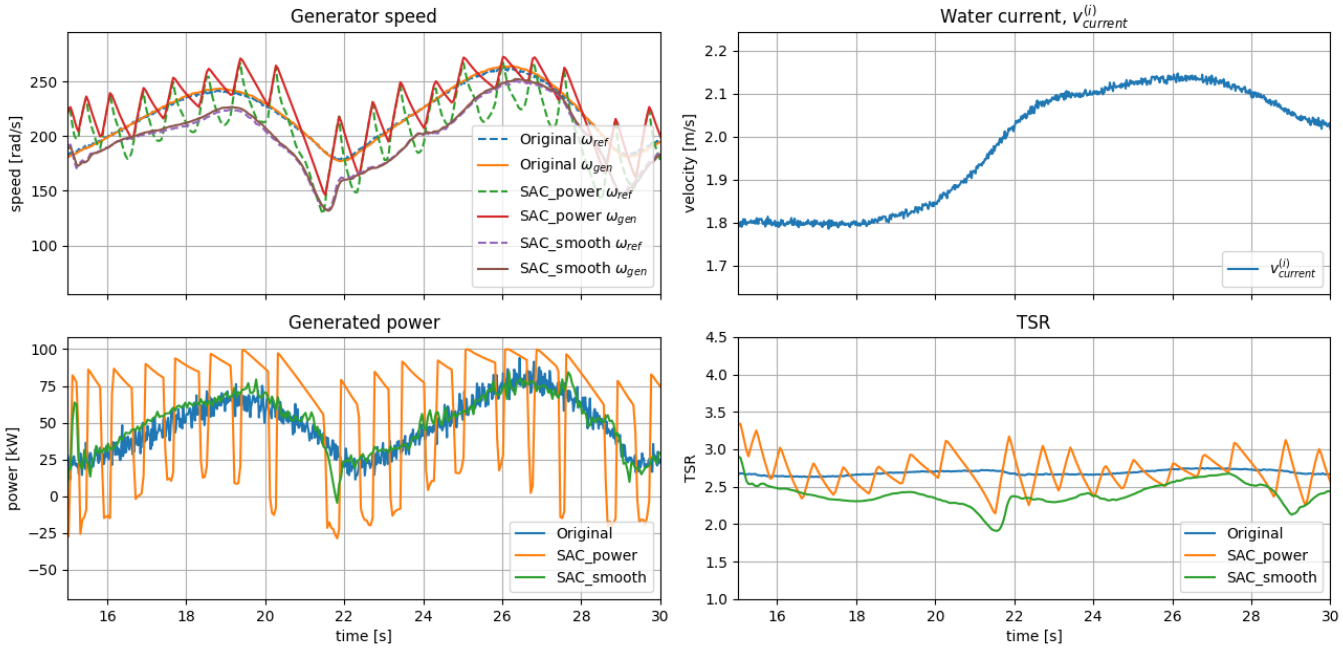


Figure 5.4: Time series plots of the generator speed ω_{gen}, speed reference $\omega_{gen,ref}$ generated power P_{gen} and tip-speed ratio for the baseline and both SAC controllers. A short time-span of an simulation is shown to allow for a clearer visual presentation.

Figure 5.1 together with Table 5.1, indicates that both stochastic SAC agents generally generates less power then the other controllers. The power maximizing SAC controller does however generate more power then its counterparts, but at the cost of large fluctuations in $\omega_{gen,ref}$, and consequently larger fluctuations in all shown features.

The performance can be further dissected by considering the time-series graph in Figure 5.4. Which presents the most important features through time for both SAC controllers and the original controller in the stochastic environment. Unlike Figure 5.2, Figure 5.4 shows both the reference and the real generator speed. Comparing the reference signal generated by the smooth and power maximizing SAC controller it can be seen that the smooth controller generate a reference that is close to the last reference signal, resulting in a controller with a smooth reference through time. Furthermore, the smooth reference results in a small divergence between $\omega_{gen,ref}$ and ω_{gen}, confirmed by Table 5.1, meaning that a smaller and smoother electric torque is generated. This behavior also results in reduced fluctuations in both the TSR and the generator efficiency.

The power maximizing SAC controller has the opposite behavior, and rapidly fluctuates in reference speed and consequently all shown features. It does however generate a higher average power then both the smooth SAC agent, and the original controller, see Table 5.1. This controller obtain high power generation by quickly increasing the reference signal and then slowly decreasing it. This behavior results in a short period where low or negative electric torque, and consequently, power is obtained. But a much longer period where a maximal positive power is generated. Resulting in a large, net positive, power production. However, the high power comes at a cost of large and rapid changes in the reference signal and consequently, system actuators.

5.3.2 Deterministic environment

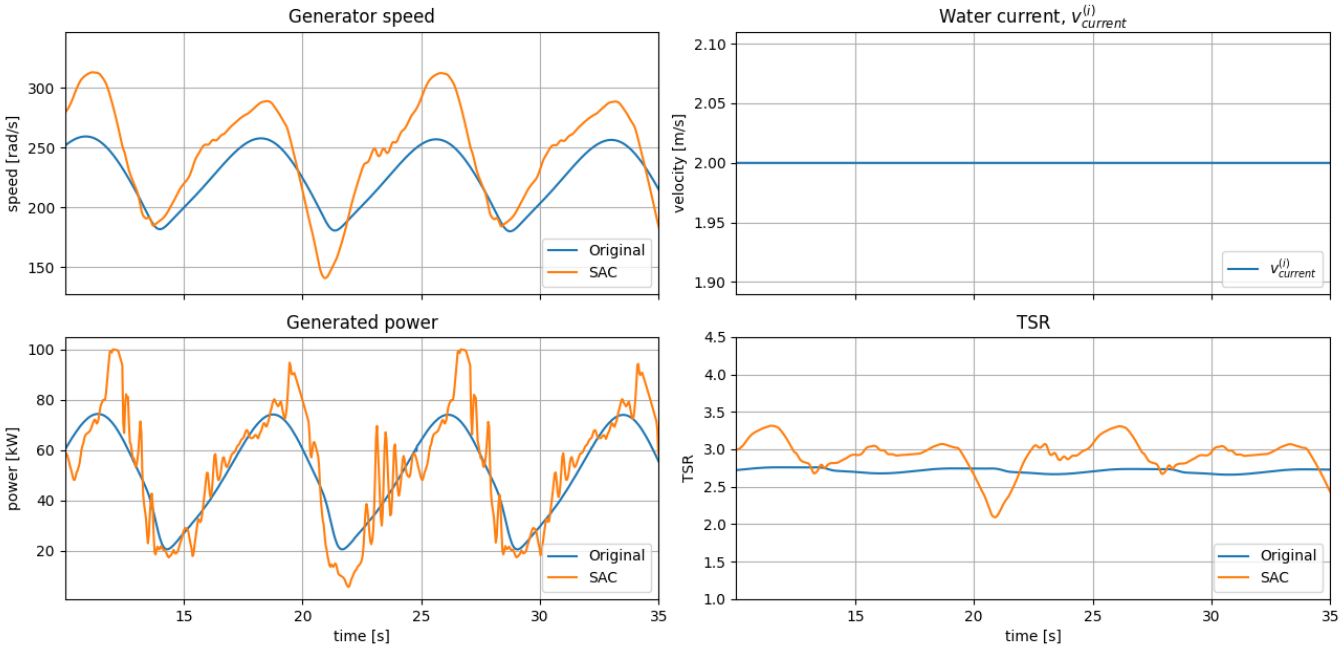


Figure 5.5: Time-series plot of major features for a SAC and the original controller. The time-series was obtained from the deterministic environment, were for example the water current is constant, see top right.

Table 5.2: Statistics of controllers across the 15 episodes shown in Figure 5.1. For each controller, the table shows the mean and standard deviation of the gausian distributed generated power P_{gen}, tip-speed ratio (TSR), and generator speed ω_{gen}. For the generator efficiency η_{gen}, reference speed variation $\Delta\omega_{gen,ref}$, and tracking error $(\omega_{gen,ref} - \omega_{gen})$, percentile-based statistics are used due to their non-Gaussian distributions.

Controller		P_{gen} [kW]	TSR	η_{gen}	ω_{gen} [rad/s]	$\Delta\omega_{gen,ref}$ [rad/s^2]	$\omega_{gen,ref} - \omega_{gen}$ [rad/s]
Original	Mean	49.612	2.708	–	222.499	–	–
	Std	18.311	0.283	–	26.306	–	–
	Median	–	–	0.926	–	0.199	1.495
	25 %	–	–	0.918	–	0.151	0.751
	75 %	–	–	0.93	–	0.231	2.018
	5 %	–	–	0.908	–	0.033	0.147
	95 %	–	–	0.93	–	0.368	3.69
SAC	Mean	49.963	2.909	–	242.385	–	–
	Std	27.195	0.373	–	44.288	–	–
	Median	–	–	0.926	–	0.332	1.976
	25 %	–	–	0.908	–	0.178	1.124
	75 %	–	–	0.931	–	0.564	3.416
	5 %	–	–	0.842	–	0.042	0.259
	95 %	–	–	0.936	–	1.0	9.192

X% indicating the Xth percentile. It is shown in place of standard deviation when the distribution is not bell shaped.

In Section 5.1, it is seen that SAC is out performed by the other controller types in most scenarios. To display a situation where SAC performance better, the results from the deterministic environment is shown, see Section 4.2.3.1. In the deterministic environment the stochastic water function is for example exchanged with a steady water velocity. In Figure 5.5, a segment of a simulated episode is shown using the deterministic environment. From the graphs it can be seen that the SAC agent generates more power then the original controller by increasing the mean TSR and the amplitude of the generator speed. The increase in power is confirmed in Table 5.2. The increased power generation also seems to cohere with the rapid decline of generator speed after the peek is reached, when comparing SAC to the original. Both the table and the time-series plots indicates a similar pattern as in the stochastic case, SAC can increase power but at the cost of increased fluctuations.

6
Discussion

The discussion focuses on elaborating on and interpreting the findings of the project. By firstly presenting a comparison between all controllers, to then analyze each control method separately, the discussion seeks to give results evidence based claims about the developed control methods and how they relate to the research questions of the project.

6.1 Main takeaway

The main takeaway from the results is not merely that two data-driven methods were tested against a baseline, but that predictive use of system dynamics appears more effective than purely reactive control in this setting. The RNN-based controllers consistently outperformed the baseline across unseen evaluation environments, which indicates that short-horizon prediction of relative inflow provides control-relevant information before the disturbance is fully visible in the measured output. In practice, this means that the controller can adjust the generator reference in anticipation of the coming operating condition rather than only react to it afterwards.

The results also highlight the role of periodicity. Because the kite motion is repetitive and the inflow-related signals exhibit structured temporal patterns, forecasting becomes useful for control rather than only for analysis. This helps explain why the predictive controllers improve performance even though the numerical gain is modest: the improvement is consistent, and it follows from exploiting recurring dynamics that a reactive controller does not explicitly use.

6.2 Comparison between controllers

From the results shown in Section 5.1 it was seen that the RNN-based controllers in general outperformed the baseline and SAC controllers. It was however seen that all controllers shared some general power, generator speed and TSR characteristics in regards to their means, amplitudes and periods. This finding came as no surprise for the predictive controllers, as the gradient controller was an extension of, and the TSR controller shared the same objective as the baseline controller. But, this was not true for the SAC agents, which did not have any penalties or rewards directly influencing them to follow or stay close to a generator speed or power trajectory. Thus, this indicates that the control signals generated by the controllers shares behaviors that aligns with, and are close to, an optimal control strategy. While this

does not necessarily imply that a similar generator control would be close to the optimal on the real system, it is a strong indication that it is for the given system and kite model developed in this project.

The results indicates, five out of the six controllers operate at a similar mean TSR in the range $[2.7, 2.9]$, see Table 5.1, which coincide with the power coefficient optimal TSR seen in Figure 3.7b. The smooth SAC agent has a notable lower mean TSR of 2.39 which could be because it is tuned for some other relations at the cost of a sub optimal TSR value. Since the smooth SAC controller fell behind in power generation, this serves as a strong indicator that optimizing ω_{ref} for turbine optimal TSR is a reliable method for power optimization. However, the results shows that the power optima is not unique to a TSR trajectory, see Figure 5.2. As such, a coupled systems approach that considers both turbine, generator and kite control dynamics may be required to find the true power optima.

The difference between the controller families is not only a matter of performance, but of how the control problem is posed. The predictive controllers solve a narrower problem: first estimate a physically meaningful future inflow signal, then use that estimate inside a structured control law. This makes the method easier to interpret and allows it to benefit directly from the periodic character of the kite motion. SAC instead solves a broader end-to-end learning problem, where the policy must infer useful state information, balance exploration and exploitation, and learn from a reward that only indirectly reflects good control behavior. In the present setting, that added flexibility appears to come at the cost of lower robustness and less consistent performance.

6.3 Supervised predictive controllers

This section discusses the RNN-based supervised predictive controllers by examining the supervised dataset and the RNN model used. This is followed by a discussion on the developed predictive controllers, their control behavior and TSR operating point.

6.3.1 The supervised dataset and RNN

One central question when developing the RNN predictive controllers was whether a limited feature space was sufficient for a RNN to learn meaningful relationships from. The reliance on a limited set of data features and assumption that the turbine and TSR equations could be used to calculate the relative velocity and find the ground truth were key uncertainties in the development of the predictive controllers. Despite this, the RNN model developed was able to find and learn the inflow velocity dynamics and accurately predict it. While the question about the reliability in using real world data remains highly uncertain, the results indicate that the limited feature space captures the dynamics needed to create a supervised dataset that an RNN can learn from.

Furthermore, it was concluded that the RNN did not require long term memory to accurately predict distant future inflows. Even though a GRU based RNN was used to handle eventual long-term dependencies, this was not needed for the task. A simpler RNN structure may be sufficient for velocity prediction and is likely a result of the highly periodic behavior in most of the features used.

6.3.2 Predictive control methods

The predictive controllers support a clearer interpretation of why performance improves. The gradient-based controller extends a reactive baseline with a forecast-dependent term that compensates part of the lag between inflow change and generator response. The TSR-based controller uses predicted inflow to move the system toward a favorable operating region before the disturbance is fully reflected in the measured output. In both cases, the benefit comes less from model complexity in itself and more from converting temporal structure into anticipatory control action.

This also clarifies the role of periodicity in the system. Since the kite motion and several measured signals exhibit a recurring pattern, short-horizon forecasts become informative enough to influence the present control decision. The result is therefore best understood as evidence that prediction is useful when the plant contains structured, partly repetitive dynamics.

In regards to the TSR operating point, seen from the predictive controllers, the results show that a higher average TSR value is preferred to maximize power and that power optimization is not unique to a TSR trajectory. However, it remains unclear if it is a result of the anticipatory capabilities of the controllers, or simply the baseline controller being tuned to stay at a lower TSR. Investigating whether the baseline controller can achieve the same amount of power output, simply by increasing its TSR operating point would provide further insight.

6.4 SAC performance issues

As depicted in Section 5, SAC often suffers from reduced power generative performance in comparison to the other controller architectures. With the power maximizing SAC agent being the exception. The reason for the performance of the different SAC agents is discussed in this section.

In contrast to the predictive controllers, the SAC results should be interpreted primarily as a demonstration of potential rather than as evidence that reinforcement learning is currently the stronger solution for this problem. This is best understood as a consequence of the formulation used in the present study. In the stochastic environment, the agent must learn under partial observability and from a reward signal that only approximates the true control objective, which can lead to unstable or overly aggressive policies.

Three limitations appear especially important. First, performance is sensitive to reward design, since small changes in how power, smoothness, and penalties are weighted can greatly affect the learned behavior. Second, the problem is partially observable, because important drivers such as the full water-current state are not directly available to the agent. Third, SAC remains highly dependent on tuning and training setup, which makes it difficult to separate limitations of the algorithm itself from limitations introduced by the chosen hyperparameters. Together, these factors help explain why SAC performs less consistently than the predictive controllers in the stochastic environment, while improving in the more deterministic case. The stronger SAC performance in the deterministic environment supports this interpretation: when the environment becomes easier to infer, the learned policy also improves. This suggests that SAC is not necessarily unsuitable for the problem, but that its performance in the present setting is more dependent on observability, reward shaping, and tuning than that of the predictive controllers.

6.4.1 Insufficient parameter optimization

Even though SAC is less sensitive to poor parameter selection then other DRL algorithms [31]. The performance of a trained SAC agent is still dependent on the choice of training parameters, and especially the shape of the reward function. Meaning that there exist optimal combinations thereof and that a suboptimal combination leads to reduced performance. Since there exits no predetermined set of parameters suitable for any custom environment, all parameters and the reward shape was manually set based on similar works and optimized through trail and error. Which is highly unlikely to result in an optimal agent. Therefore, a common optimization path is to randomly test a huge amount of combinations, using tools like Optuna[1] and use the best performing one. In this work, this kind of optimization was not utilized due to limited computer power and high fidelity simulation with an extended simulation time. The problem is further obstructed due to the stochastic nature of SAC training, were two identical training instances may lead to fundamental different control policies. Which introduced the requirement of running multiple training sessions for each parameter combination while tuning.

Therefore, this work did not seek to find the optimal combination of parameters leading to optimal performance. Hence, the trained SAC models are suboptimal agents but shows the approximate potential of SAC in the kite turbine control domain.

6.4.2 Simulation exploitation

The power maximizing SAC agent does produce more power then the other controller, see Section 5.1. This is however accomplished through excessive changes in the reference signal and consequently the electrical torque of the generator. While this is possible in the simulated environment, it could lead to problems when applied to the real system. Firstly, in the real system the behavior is likely to result in high

[1] `https://github.com/optuna/optuna`

wear and tear of mechanical components [32]. Secondly, the dynamics of the real system is not fully captured in the developed model and therefore the behavior is likely to have reduced performance due to unmodeled dynamics. Following the argument, it is also likely that the control behavior observed while simulating the power maximizing SAC agent does not translate to reality at all. Meaning that the agent has learned to use an unrealistic property of the simulation to obtain unrealistically high power output. Meaning that the power maximizing SAC agent is likely to have substantially worse performance in the real system compared to the other controllers.

Another simulation related performance issue is that the action noise used during training of an SAC agent results in increased power generation, and consequently reward. That is, the larger the action noise the larger the reward. The reason for this behavior is identical to that of the power maximizing agent, spikes in the reference signal leads to increased power generation. Since the action noise is usually only present during training of a SAC agent, and the deterministic action is used alone during evaluation, this often leads to worse performance at evaluation time. As concluded above, this is likely not a realistic behavior but instead an oversight in the model that taints the performance of the SAC algorithm.

The strongest SAC behavior should therefore be treated with caution. When a learned policy improves power mainly through aggressive oscillations in the reference signal, this may reflect exploitation of simplifications in the simulated environment rather than a robust operating strategy. The SAC results are thus informative about what learning-based control can discover in the model, but not yet sufficient to claim that the same policy structure would be desirable on the physical system.

6.4.3 Partial observability

As mentioned in Section 4.2.1.1 the kite system is a POMDP, thus inferior performance is expected. The reason for this is likely the measurement noise added to the features of the observation and because the observation is a reduced set of the state. That is, the observation is missing information necessary to describe the complete state of the kite, turbine and surrounding environment. Some example of missing information is the water current $v_{current}$, position (x and y, z is already present in the observation) and linear velocity of the kite.

To mitigate the performance loss in the stochastic POMDP environment the measures presented in Section 4.2.2.5 was implemented. Nonetheless, the SAC algorithm performs worse than the other controllers on the stochastic environment. SAC does however perform better on the deterministic case. Which is not fully observable, due to missing features, but much more observable then the stochastic environment. Since the introduction of random change in an unobserved feature, the water current, lead to strongly reduced SAC performance this indicates that a factor in the performance issues with SAC is the partial observability of the stochastic environment. Meaning that the observation, even extended with frame-stacking and enhanced with frame-skipping does not fully describe the current kite state.

To mitigate the performance loss the three measures presented in Section 4.2.2.5 usually helps [25]. In addition, to these measures the POMDP environment can be made more observable by including more features, for example the water current and the missing positions and velocities. Which would require fitting additional sensors on the kite or calculating these features through for example sensor fusion. Furthermore, the introduction of recurrent neural networks [28] or transformer networks [39] into the actor and/or critic structure would improve SAC's performance in POMDP environments. Effectively improving the temporal information available to the agent and the usage of it. This can be seen as an improvement on the frame-stacking implemented in this work.

6.4.4 Reward shape

Even though the environment normalizes the power reward using the power baseline the reward received by the agent is not solely connected to the observation and the action. This occurs due to imperfections in the baseline power fitment as well as the POMDP nature of the environment. The power baseline is imperfect since it maps tether force to power, where power is delayed in comparison to the force. Which is the reason for the large covariance in data points seen in Figure 4.7. Furthermore, the power baseline is generated from power measures obtained from the original controller. Meaning that the power baseline indicates the power that the original controller could produce at a specific tether force, not the total available power given the current tether force.

The suboptimal reward shape could be influential in the performance degradation of the SAC algorithm in the current environment. Since the SAC training is negatively influenced if there is a discontinuity between the reward and observation-action pair.

6.5 Real system potential

It is concluded that both algorithms, RNN-based and SAC, can be trained and used to control the speed of the generator with sufficient performance in the simulated environment. While indicating the potential of adaptations to the real system, good performance in simulation is no guarantee for good performance in the real system. Below, the potential and possible solutions for this purpose are discussed.

These results should therefore be interpreted as comparative findings within the adopted simulation model, not as direct performance guarantees for the physical system. The simulation is valuable because it enables repeatable controller comparison under identical conditions, but its path locked dynamics, rigid tether assumption, simplified turbine–kite coupling, and simplified generator model limit external validity. Confidence is therefore highest in the qualitative conclusions, namely that predictive use of periodic inflow structure is beneficial and that RL performance is strongly affected by observability and reward design, while confidence is lower in the

exact numerical improvement and in how directly the learned policies would transfer to the real kite.

6.5.1 RNN

One of the main challenges in incorporating a RNN based predictive controller on the real system, lies in trusting the data collected. While this project demonstrates that the data is sufficient, it has to be acknowledged that this is in a simulation setting. Real world noise and signal processing may affect the data quality substantially. Hence, filtering out noise or processing the data in such a way that the signals can be trusted might have to be done before implementing the RNN to the real system.

Another important consideration that has to be taken into account when implementing the RNN-based predictive controller to a real system, is the computational cost and hardware requirements. While the baseline controller relies on sensor readings, the predictive controller requires a neural network inference step each time the controller reference is updated. However, it was shown in the project that the RNN needed was small and operates on a small feature space, which suggests that the computational cost likely is manageable for a modern embedded control hardware.

Lastly, a challenge when applying the RNN based predictive controllers to the real system is the model's ability to generalize to operating conditions. This challenge is not only present when creating the RNN training dataset, but also when tuning the predictive controllers. In a real world setting, the kite may experience a wider range of variations than what the simulation environment provides, such as turbulence or changing tidal patterns. If the kite encounters conditions outside of the training dataset, or that used to tune the controllers, the prediction accuracy and power output may worsen. To address this, the RNN might have to be retrained and controllers retuned for specific environmental conditions.

6.5.2 SAC

SAC and other RL algorithms are designed to find the optimal policy given an environment. This means that the algorithm is likely to find oversights in the environment to increase the obtained reward. This can be seen in Figure 5.4, were the power maximizing agent acts unrealistic, which was discussed in Section 6.4.2. This can be mitigated by introducing a shaped reward, as done with penalties for action change and divergence in Section 4.2.1.3. But in the end it puts immense weight on the importance of creating a realistic, high fidelity, model, not containing any unrealistic oversights or loopholes. While simultaneously being light enough to be feasible for training.

This leads to the consideration of development time spent on developing an accurate and light environment model and optimizing an agent for the environment, versus development time of an agent optimized directly for the real world. That is, the

perfect environment. SAC, and other model-free RL algorithms, has the advantage that no modeling of the system, nor the environment is strictly needed. Instead the algorithm can learn directly from the real world physics, reducing the need to develop high fidelity, ultra realistic and fast system models. This does however give raise to questions about safety, mechanical wear and ethics [32].

Training SAC directly on the real system does however come with difficulties. Firstly, SAC and most DRL algorithms are designed to train on episodic tasks, where a random initial position is given each episode and the algorithms tries to maximize the cumulative reward during one episode. However, whilst training on a real system, it is not plausible to divide the training into episodes with independent random initial positions since no state is independent of past states [10]. Therefore, the training is to be done in a continuous setting instead of episodic. This also effects the reward that the agent tries to maximize since an indefinitely growing cumulative reward (infinite horizon) is not feasible. Implementation of continuous version of SAC solves this through reward centering, reset-less training and agent-controlled resets [10, 41].

Secondly, training directly on the real kite system comes with risks concerning mechanical wear and damage due to SAC actions, as well as ethical considerations. Wear and damage risks can be mitigated through a rigid fail-safe controller that assumes control when the SAC agents acts maliciously. Furthermore, offline training procedures can be adopted to pretrain the SAC algorithm with collected kite data, allowing the controller to have a stable, proficient and safe policy when the online training is commenced. This was to some extent done through the prefilled buffer, implemented in Section 4.2.2.4. More advanced algorithms that pretrain the actor and/or critic network based on collected data and has shown promising results is IQL[17] and CQL[19]. These method were not considered in this thesis, but should be implemented if SAC is to be utilized on the real system.

7
Conclusion

7.1 Summary of findings

The main contribution of this thesis is the demonstration that predictive, data-driven turbine control can outperform a reactive baseline in a simulated underwater-kite setting. The strongest result is not simply that multiple methods were evaluated, but that short-horizon prediction of relative inflow, combined with the periodic structure of the kite motion, provides actionable information for control. The RNN-based controllers consistently improved generated power relative to the baseline across unseen evaluation environments, with an average improvement of about 1.3%.

The results also show that the relevant operating point should be understood at system level. Operating near a favorable TSR region remains important, but the best-performing controllers do not follow one identical TSR trajectory, which indicates that the relevant optimum belongs to the coupled kite-generator system rather than to the turbine in isolation.

For SAC, the thesis shows promise but also clear limitations. The method can learn meaningful control behavior, especially in the more deterministic environment, but its performance in the stochastic setting is limited by reward design, partial observability, and tuning sensitivity. In the present formulation, this makes SAC less reliable than the predictive controllers.

Finally, the conclusions must be interpreted within the scope of the simplified simulation environment. The work supports the qualitative claim that predictive use of periodic system behavior is beneficial for turbine control, but the exact quantitative improvements should be treated as simulation-based indications until validated in a higher-fidelity model or on the real system.

7.2 Future work

The project serves as a proof-of-concept for the potential of data-driven control solutions for cross-flow underwater kites. To further investigate the potential of data-driven turbine control systems, the kite simulation environment needs to improve. A 6 DOF, high fidelity, kite model, accurate generator modeling, more realistic sensor noise and inclusion of the kite steering system are improvements that would further

strengthen the claims made in the project.

Future work also includes investigating the optimal TSR operating point, as there were mixed signals in the results in regards to an optimal TSR trajectory for power optimization. This could be due to the simplified simulation and has to be further investigated.

In regards to the predictive controllers, the accuracy of the calculated relative velocity would need to be verified for a eventual future implementation, as a reliance on noisy signals can cause problems. Furthermore, it would be of interest to see how the developed control methods behave in a more realistic environment and if they can be further optimized for power while maintaining system stability.

To increase the viability of DRL-based control solutions, further remedies for the partial observability is needed. This might be to add features to the kite measurements, either through the addition of sensor or through derivation of positional features from the available measurements. Or by incorporating new functionality to the SAC algorithm, for example RNN-based actor and critic networks. It is also likely that more advanced offline algorithms are needed to allows for more efficient usage of historical kite data, prior to online training.

Bibliography

[1] Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018. [Accessed 02-03-2026].

[2] Jacob Adamczyk, Volodymyr Makarenko, Stas Tiomkin, and Rahul V. Kulkarni. Average-reward soft actor-critic, 2025.

[3] Ehsanolah Assareh and Mojtaba Biglari. A novel approach to capture the maximum power from variable speed wind turbines using pi controller, rbf neural network and gsa evolutionary algorithm. *Renewable and Sustainable Energy Reviews*, 51:1023–1037, 2015.

[4] Mehmet Bakırcı and Sezayi Yılmaz. Theoretical and computational investigations of the optimal tip-speed ratio of horizontal-axis wind turbines. *Engineering Science and Technology, an International Journal*, 21(6):1128–1142, 2018.

[5] Hansenclever F. Bassani, Renie A. Delgado, José Nilton de O. Lima Junior, Heitor R. Medeiros, Pedro H. M. Braga, Mateus G. Machado, Lucas H. C. Santos, and Alain Tapp. A framework for studying reinforcement learning and sim-to-real in robot soccer, 2020.

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[7] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2), 1990.

[8] European Environment Agency. Renewables, electrification and flexibility: For a competitive eu energy system transformation by 2030, 2025.

[9] Alireza Falatoori, Alireza Riasi, Gretar Tryggvason, and Alireza Mahdavi Nejad. Tethered undersea kite turbine for tidal energy harvesting: A numerical design study. *Renewable Energy*, 256:124215, 2026.

[10] Homayoon Farrahi and A. Rupam Mahmood. Learning without time-based embodiment resets in soft-actor critic, 2025.

[11] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies, 2017.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[13] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[15] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8), 1982.

[16] Shivaram Kalyanakrishnan, Siddharth Aravindan, Vishwajeet Bagdawat, Varun Bhatt, Harshith Goka, Archit Gupta, Kalpesh Krishna, and Vihari Piratla. An analysis of frame-skipping in reinforcement learning, 2021.

[17] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning, 2021.

[18] Tróndur Kragesteen, Knud Simonsen, Andy Visser, and Ken Andersen. Identifying salmon lice transmission characteristics between faroese salmon farms. *Aquaculture Environment Interactions*, 10:49–60, 02 2018.

[19] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning, 2020.

[20] M. Lapan. *Deep Reinforcement Learning Hands-On*. Expert insight. Packt Publishing Ltd, 2024.

[21] Haocheng Li, David J. Olinger, and Michael A. Demetriou. Modeling and control of tethered undersea kites. *Ocean Engineering*, 190:106390, 2019.

[22] Zhe Liu, Yi Zhao, Yuerong Zhou, and Faming Guan. Modeling, simulation and test results analysis of tethered undersea kite based on bead model. *Renewable Energy*, 154:1314–1326, 2020.

[23] Georgios Mademlis, Yujing Liu, Peiyuan Chen, and Eric Singhroy. Design of maximum power point tracking for dynamic power response of tidal undersea kite systems. *IEEE Transactions on Industry Applications*, 56(2):2048–2060, 2020.

[24] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4), 1943.

[25] Lingheng Meng, Rob Gorbet, Michael Burke, and Dana Kulić. Experimental study on the effect of multi-step deep reinforcement learning in pomdps, 2025.

[26] Lingheng Meng, Rob Gorbet, Michael Burke, and Dana Kulić. Multi-step first: A lightweight deep reinforcement learning strategy for robust continuous control with partial observability. *Neural Networks*, 199:108521, 2026.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[28] Tianwei Ni, Benjamin Eysenbach, and Ruslan Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps, 2022.

[29] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, page 3803–3810. IEEE, May 2018.

[30] Aske Plaat. *Deep Reinforcement Learning*. Springer Nature Singapore, 2022.

[31] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[32] Antonin Raffin and Freek Stulp. Generalized state-dependent exploration for deep reinforcement learning in robotics. 05 2020.

[33] James Reed, Andrew Abney, Kirti D. Mishra, Kartik Naik, Edmon Perkins, and Chris Vermillion. Stability and performance of an undersea kite operating in a turbulent flow field. *IEEE Transactions on Control Systems Technology*, 31(4):1663–1678, 2023.

[34] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 1958.

[35] Abolfazl Shiri, Jan Hallander, and Björn Bergqvist. Design of low drag-to-power ratio hydrokinetic turbine. In *MARINE VIII*, International Conference on Computational Methods in Marine Engineering, pages 441–452, Gothenburg, Sweden, 2019. CIMNE.

[36] Eric Singhroy. Taming dragons - improving power performance of tethered turbine tidal farms. Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2017.

[37] Daniel Soler, Oscar Mariño, David Huergo, Martín de Frutos, and Esteban Ferrer. Reinforcement learning to maximize wind turbine energy generation. *Expert Systems with Applications*, 249:123502, 2024.

[38] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 2018.

[39] Dong Tian, Onur Celik, and Gerhard Neumann. Chunking the critic: A transformer-based soft actor-critic with n-step returns, 2025.

[40] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2025.

[41] Yi Wan, Dmytro Korenkevych, and Zheqing Zhu. An empirical study of deep reinforcement learning in continuing tasks, 2025.

[42] Yao Wang and David Olinger. Modeling and simulation of tethered undersea kites. 06 2016.

[43] Kristin C. Wu and Jonathan S Litt. Reinforcement learning approach to flight control allocation with distributed electric propulsion. Technical Report E-20165, NASA, Washington, DC, November 2023.

[44] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network, 2015.

[45] Chen Zeng and C. Guedes Soares. Simulation and control of a tethered undersea kite with an onboard counter-rotating turbine. *Ocean Engineering*, 320:120289, 2025.

[46] Jincheng Zhang, Xiaowei Zhao, and Xing Wei. Data-driven structural control of monopile wind turbine towers based on machine learning∗∗this project has received funding from the european union's horizon 2020 research and innovation programme under the marie sklodowska-curie grant agreement no 765579. *IFAC-PapersOnLine*, 53(2):7466–7471, 2020. 21st IFAC World Congress.

DEPARTMENT OF MECHANICS AND MARITIME SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY