Reinforcement Learning Actor for Blade Pitch Control in Wind Turbine Systems

Edouard Lacroix
Romain Birling
Thomas Zurecki

ML Project Report
In collaboration with UNFoLD LAB at EPFL
Lab supervisor : Daniel Fernex
GitHub Repository

20 décembre 2023

Group project 2 in Machine Learning

Edouard Lacroix, Romain Birling, Thomas Zurecki
Department of Computer Science, EPF Lausanne, Switzerland

I. Introduction

The global push for cleaner energy requires a focus on sustainable electricity generation. This project aims to use reinforcement learning to optimize wind turbine performance, enhancing wind energy capture and reducing environmental impact. The main goal is to create an intelligent agent capable of dynamically adjusting blade pitch based on real-time data, specifically forces on the blades. The project addresses the need for sustainable energy, recognizing traditional control methods may not fully exploit wind energy potential.

This project is made possible by Unfold Laboratory at EPFL, which specializes in experimental measurements and modeling of vortex-dominated flow. Our project uses Unfold Lab's experimental data, featuring a miniaturized turbine in water currents simulating real-world conditions. The wind turbine has three pivoting blades rotating around a vertical axis, with sensors for real-time data capture on Ct and Cr (see Figure 1 for details).

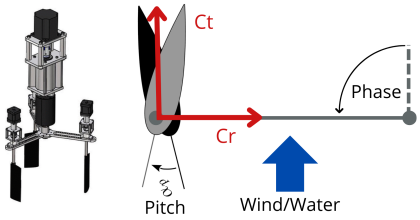

FIGURE 1 – Experience set-up [1].

Before delving into the specifics of our implementation, it is essential to provide a concise overview of our methodology and the process employed to establish and train our intelligent agent.

II. Methodoly

Understanding the intricacies of this project requires a foundational grasp of reinforcement learning (RL), a subset of machine learning. Situated within the broader domain of machine learning, RL involves an agent sequentially making decisions through interaction with its environment, aiming to maximize a cumulative reward signal. The interpretation of "reward" is intricately tied to the specific goals set for the agent, serving as a distinct metric for success. In our case, the reward should be associated with the level of energy captured by the turbine.

To train our agent, an interactive environment it can interact with and explore is crucial. Due to setup inaccessibility, such as the number of trials needed to train an agent, we created a virtual environment using a surrogate model, replicating turbine dynamics over time. This phase entailed replicating the physical characteristics and dynamics of the wind turbine over time.

With the surrogate model, agent training began using the Deep Deterministic Policy Gradient (DDPG) algorithm for continuous action spaces (details in section 4).

The methodological synthesis unfolded through two pivotal tasks : 1) the development of a surrogate model simulating the state-action-state relationship in a virtual environment, and 2) the training of the actor agent to propose actions based on the current state as it can be seen on figure 2.

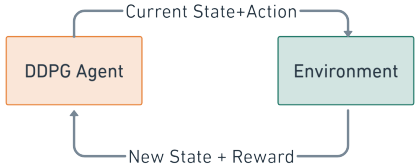

FIGURE 2 – Overview of the methodology applied

III. Surrogate model

In the pursuit of optimizing wind turbine performance through reinforcement learning, an accurate surrogate model is crucial. This section details the steps in creating and training the surrogate model, serving as the virtual environment for our intelligent agent.

A. Dataset analysis and preprocessing

To develop a robust surrogate model, it is essential to work with a sizable and diverse dataset that spans the entire environmental space. Consequently, we utilized three datasets derived from distinct experiments to ensure comprehensive coverage. These datasets, denoted as Case 1, Case 2, and Case 3, collectively contribute to capturing a wide range of environmental scenarios.

[Case 1] The experiment was a single objective Bayesian optimization, where the goal was to maximize the produced power C_p that is directly linearly proportional to the tangential

force C_t. The first 30 tests of the experiment were randomly defined pitch kinematics.

[Case 2] The experiment was a dual objective Bayesian optimization : 1) maximize the produced power C_p, 2) minimize the thrust force C_{thrust} because it is a loss of wind for other wind turbines that may be behind, and thus a loss of energy. The first 30 tests of the experiment were randomly defined pitch kinematics.

[Case 3] The experiment was a dual objectives genetic optimization : 1) maximize the produced power C_p, 2) minimize the standard deviation of the torque C_m applied to the blade. It doesn't try to minimize directly C_m because high torque is not a problem, but high standard deviation means vibrations and may damage the turbine in the long term.

All dataset cases were recorded on the same wind turbine, maintaining a constant rotation speed and water flux. However, sampling rates varied from 250Hz for $case$ 3, to 1200 Hz for (res), and 300Hz for $case\ 1 + case\ 2$. To ensure consistency between data, we resampled every case to 50 data points per revolution (around 1 second).

A meticulous dataset analysis and preprocessing were crucial for eliminating irrelevant features and refining data selections, ensuring the surrogate model's effectiveness.

In the data preparation process, we took several key steps. Initially, we converted Matlab data to a Pandas DataFrame for smoother manipulation in Python scripts.

The subsequent step involved attribute selection, a collaborative effort with our Unfold Lab supervisor to identify key attributes defining observations, actions, and rewards for the RL agent. Refer to figure 3 for a detailed explanation of how the agent interacts with its environment.

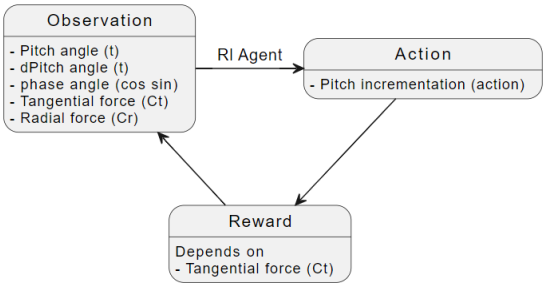

FIGURE 3 – Observations, actions, and rewards

To simulate the environment for training RL agents, replicating the next observation vector from the current one and a given action was essential. Many features of the observation space could be computed at simulation time :

1) $phase_{cos}$ and $phase_{sin}$: As the wind turbine rotation speed is constant, we can compute the next angle by constantly incrementing it.

2) $pitch$: as the action is the pitch increment, we can sum up the action with the current pitch to determine the next pitch angle.

3) $dpitch$: the feature $dpitch$ is equals to $pitch_t - pitch_{t-1}$. It approximately represents the $pitch$ angular speed in the observation vector. As the action is simply the pitch increment to the next step, we simply put the action at time t into the column $dpitch$ of the observation at time $t + 1$.

However, it was practically not feasible with our resources to compute mathematically the evolution of the tangential force C_t and of the radial force C_r because of the complex physical behaviors behind this system. The purpose of the surrogate model was to determine the evolution of those two forces given the current state and action. Figure 4 illustrates the surrogate model's intended behavior.

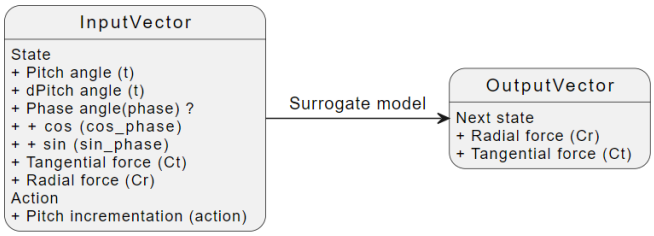

FIGURE 4 – Surrogate model inputs and outputs

Finally, we figured out that the dataset $case$ 3 was much bigger than the two others. There were also other sources of unbalancing in the dataset cases, as when the Bayesian optimization started after the random phase for $case$ 1 and $case$ 2, every case experiment looked closely to each other. To address this and ensure a balanced dataset for an accurate surrogate model in any situation, we implemented random case selection using a slightly adjusted probability distribution.

Data selection constituted another integral aspect. Indeed, as said before, we had to make a dataset that represents the entirety of the environment, thus we decided to conduct a precise selection of experiment rounds through dataset cases, favoring disparity. As a result, we decided to choose the first 30 rounds of $case$ 1 and $case$ 2 for their randomness plus 30 and 40 rounds respectively. For $case$ 3, we applied a decreasing linear probability from round 1 to the end.

Finally, we decided to shuffle normalize, and then partition the dataset into a 90% training set and a 10% validation set. This deliberate division aimed at selecting a good regularization term and then avoiding overfitting while maintaining very good results.

B. Model Architecture

The surrogate model's architecture is a multi-layer perceptron designed to capture complex relationships between input state-action pairs and resulting output states.

At its core, the neural network comprises an input layer, seven hidden layers, and an output layer. The input layer, with

six nodes, encapsulates the selected state variables—$Pitch$, $Dpitch$, $Phase$, C_t, C_thrust, C_r and $Action$ (pitch increment)—forming a comprehensive representation of the wind turbine's state and action pair. The inclusion of seven hidden layers allows the neural network to discern complex patterns and dependencies within the data. The dimensions of these hidden layers (refer to Figure 5) have been fine-tuned through experimentation to allow the model to capture subtle features from the dataset, incorporating batch normalization and ReLU activation functions between each hidden layer. The output layer, consisting of two nodes, predicts the new state variables, specifically C_t and C_r, based on the provided input state-action pairs. The accurate prediction of these variables is paramount for the intelligent agent's decision-making process.

The output layer, with two nodes, predicts new state variables, specifically C_t and C_r, based on the provided input state-action pairs. Accurate prediction of these variables is crucial for the intelligent agent's decision-making process.

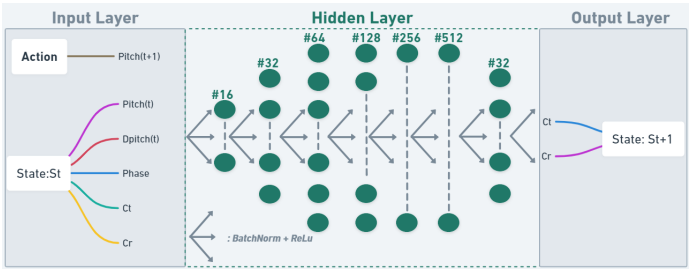

FIGURE 5 – Surrogate model architecture

In terms of optimization, the surrogate model employs the Adam optimizer due to its efficacy in handling non-stationary objectives and noisy gradients. The learning rate is set to 1e-3 to strike a balance between rapid convergence and stability during training. Additionally, a regularization term of 1e-3 is introduced to mitigate overfitting, promoting the model's generalization across diverse environmental scenarios. To adaptively adjust the learning rate during training, the model utilizes the StepLR scheduler. With a step size of 10 epochs and a decay factor (gamma) of 0.95, this scheduler enhances the model's adaptability to evolving patterns in the dataset, contributing to improved convergence and performance. The surrogate model undergoes 1000 epochs of training to balance achieving convergence and ensuring robust learning. A batch size of 2048 is chosen to optimize computational efficiency and effective gradient descent. These parameters are fine-tuned through iterative experimentation to enhance the model's overall performance.

C. Evaluation

In this section, we thoroughly evaluate the performance of the surrogate model, employing diverse quantitative metrics and visualization techniques to assess its efficacy.

Quantitative evaluation of the surrogate model focuses primarily on MSE loss values, crucial for computing gradients during training. Our best-performing model achieved a train loss of 0.0038 and a test loss of 0.0055. The low train loss indicates successful capturing and learning of underlying patterns within the training data, an essential measure of the model's ability to understand complex relationships between input state-action pairs and their corresponding output states. The test loss of 0.0055 demonstrates the surrogate model's robustness and generalization capability. The minimal difference between train and test loss suggests that the model avoids overfitting to the training data and can effectively generalize to unseen instances. To gain insights into the surrogate model's understanding of the environment, Principal Component Analysis (PCA) is employed to reduce dataset dimensionality. This technique helps visualize the environment distribution in reduced dimensions, assessing how well the surrogate model captures its complexities. Gaps in visualization highlight areas needing further refinement, especially in cases of suboptimal data selection (see Discussion for more details). Furthermore, the Shapley Additive Explanations (SHAP) method is employed to provide valuable insights into the importance of each feature influencing the model's output. Analyzing SHAP values identifies features with significant impact on predictions, crucial for understanding the relative importance of different input variables on the output and ensuring alignment with physical constraints.

D. Final Environment

After training the surrogate model, we utilized it to implement the environment class for training our RL agents. To ensure compatibility with popular reinforcement learning libraries, we chose to create a class inherited from the Env class in the gymnasium package. The $step$ and $reset$ functions were implemented to simulate a wind turbine episode in its environment and facilitate agent training. We assessed the accuracy of our surrogate model through this environment using the procedure implemented in the $simulate\_open\_loop\_episode$ function within the $environment.py$ file :

1) Start from an initial state and a list of actions.

2) Compute the next state by calling the environment step function with the initial state and the first action.

3) Feed this new state to the environment with the second action to generate the third state, and so on...

By providing this function with a true initial state recorded in the dataset and the following list of actions, we can compare the simulated states evolution to the states that occurred when playing those actions from the same initial states. This procedure is tested in the notebook "test environment.ipynb". Despite initial concerns about potential divergence in simulation by feeding predicted states to the surrogate model, it demonstrated remarkable accuracy even in simulating long episodes, such as 3 turns. The predicted evolution of forces C_t and C_r, computed by the surrogate model, is depicted in figure 6 below.

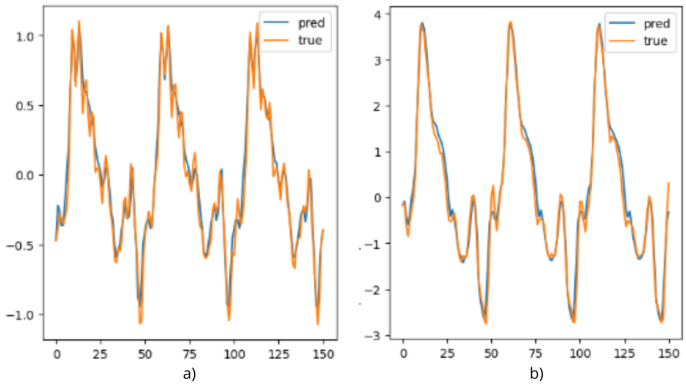

FIGURE 6 – Simulation by the environment during 3 rounds for a) C_t and b) C_p.

IV. DDPG Agent

We initially employed the Deep Deterministic Policy Gradient (DDPG) algorithm, being familiar with its concepts, as it seemed to be an appropriate algorithm for this task.

Utilizing the DDPG class from the StableBaselines3 library, we trained it on our custom gymnasium environment. The algorithm quickly identified effective strategies for maximizing the tangential force C_t, and consequently, C_p. However, our attempts to find the "best strategy" proved challenging, as we noticed a decline in the agent's performance after a certain number of training steps, causing the mean reward per episode to plummet. Despite efforts, we couldn't identify reasons within the DDPG algorithm for this abrupt learning stoppage. Consequently, we decided to explore an alternative learning strategy : the Soft Actor Critic (SAC).

V. SAC Agent

When discussing with our supervisor from UNFoLD Daniel Fernex, we learned that a student successfully trained a Soft Actor-Critic agent by allowing it to interact with a real wind turbine during his master's thesis. Considering our limited success with DDPG, we decided to explore Soft Actor-Critic. Utilizing the StableBaselines3 implementation, the disturbing behavior observed before seemed to be repeated. Monitoring the training mean reward curve with TensorBoard revealed that, after a certain number of steps, the mean reward also crashes down.

VI. Discussion

In this section, we'll discuss past failures encountered during the project and experiments that, unfortunately, we didn't have the time to explore.

A. Failed experiments

We kept a trace of our failed experiments in the directory "outdated tests" at the root of our project. Respectively :

1) Custom implementation of the DDPG algorithm, where we aimed to refine parameters such as exploration noise, learning rates, target network update frequency, etc. Due to time constraints and the complexity of implementing such an intricate algorithm, we abandoned this idea and opted for StableBaselines3's DDPG implementation.

2) A full offline DDPG implementation, exploring a strategy to use the algorithm without implementing an environment and surrogate model. We attempted to populate the DDPG agent's replay buffer with reformatted data.This idea was abandoned when we realized our surrogate model performed exceptionally well after resampling the data to a higher timestep (50 measures per revolution).

It is important to note that given the abandonment of these ideas and subsequent refactoring, the notebooks may not work. However, they remain valuable for the plots they contain.

B. Unrealized Experiments

There were a lot of things we would have wanted to try but couldn't because we ran out of time. The following list represents only a small part of them :

1) Removing the phase from the agent's observation space to make it solely consider wind force, thus adapting to wind direction variations.

2) Clipping the action to a narrower range to address unrealistic pitch policies and account for motor constraints, as slight noise led to unrealistic jumps.

3) Use pitch angular acceleration $ddpitch$ as an action instead of pitch angular speed or pitch increment $dpitch$. While this could address the noise issue, it might slow down agent learning as they would need to understand the impact of the action on $dpitch$ before grasping its influence on $pitch$ and, consequently, the reward.

4) Other RL algorithms and better parameter adjustments. If we had more time we would have also liked to try other policies and to play with more hyperparameters of each policy to try to get better and better policies. We were advised to use SimpleRL because it provides a modifiable independent implementation of many RL algorithms. We didn't have time to do so.

5) Integrating dropouts into the surrogate model architecture to enhance robustness.

VII. Conclusion

In overview, this project aimed to enhance wind turbine performance by employing reinforcement learning techniques. Drawing on the expertise of Unfold Laboratory, we developed and trained a precise surrogate model, using it to train an intelligent agent capable of dynamically adjusting wind turbine's blades pitch. Unfortunately, our final agent fell short of meeting the performance expectations necessary for practical deployment.

VIII. Ethic & Risks

The main concern in terms of risk considering wind turbines would be their effect on wildlife, more precisely on birds. Wanting to quantify this risk we can look at figure 7 presenting the proportion of birds killed in a year by different hazards.

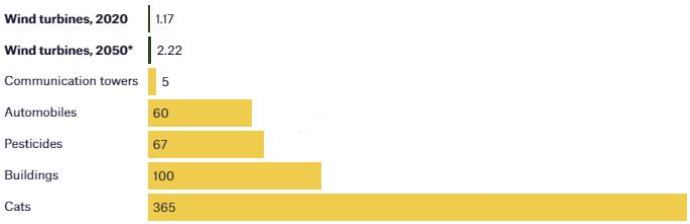

FIGURE 7 – Estimated number of birds killed by hazards in the US each year (millions) [2].

As it can be seen, the number of birds killed by wind turbines is rather small when put into comparison with other hazards. Even though this number is small, it still represents one of the most important ethical challenges in the implementation of wind turbines. An interesting advantage proposed by a vertical wind turbine is that it is considered safer for birds as it is easier to distinguish and dodge for them [3]. Thus the implementation and optimization of vertical wind turbines could represent a serious step up, both from an energy gathering and ethical point of view. Addressing furthermore this ethical concern would require the integration of mitigation strategies into turbine design. However, our project focused on a specific aspect of control, and we lacked the ability to influence the turbine's design. During the training of our agents, the wind turbine we used was entirely simulated in our gymnasium environment, ensuring no harm to animals throughout the project.

Références

A. Sources

[1] Optimization and control of vertical axis wind turbines, Unfold Lab EPFL https://epfl-unfold.notion.site/Optimisation-and-control-of-vertical-axis-wind-turbines-3d9c2c945da5419a805479d3822961a4

[2] Weekly data : How many birds are really killed by wind turbines ? Energy Monitor, Nick Ferris, January 31, 2022 https://www.energymonitor.ai/renewables/weekly-data-how-many-birds-are-really-killed-by-wind-turbines/?cf-view

[3] The impact of wind turbines development on birds, OWELL, April 06 2023 https://www.owellindustries.com/impact-of-wind-turbines-development-to-birds.html

B. Libraries

[4] NumPy : The fundamental package for scientific computing with Python https://numpy.org/

[5] SciPy : Scientific Library for Python https://www.scipy.org/

[6] pandas : Powerful data structures for data manipulation and analysis https://pandas.pydata.org/

[7] iPlantUML : A Python library for generating PlantUML diagrams https://pypi.org/project/iplantuml/

[8] PyTorch : An open-source machine learning library https://pytorch.org/

[9] Matplotlib : Comprehensive library for creating static, animated, and interactive visualizations in Python https://matplotlib.org/

[10] tqdm : A fast, extensible progress bar for loops and pipelines https://tqdm.github.io/

[11] scikit-learn : Simple and efficient tools for predictive data analysis https://scikit-learn.org/

[12] ipywidgets : Interactive HTML widgets for Jupyter notebooks and the IPython kernel https://ipywidgets.readthedocs.io/

[13] mat73 : A Python library for reading MATLAB v7.3 format files https://pypi.org/project/mat73/

[14] Stable-Baselines3 : High-quality implementations of reinforcement learning algorithms https://github.com/DLR-RM/stable-baselines3

[15] gymnasium : An API standard for reinforcement learning with a diverse collection of reference environments https://gymnasium.farama.org/index.html

[16] SHAP (SHapley Additive exPlanations) : A unified measure of feature importance https://github.com/slundberg/shap