Towards fine tuning wake steering policies in the field: an imitation-based approach

Claire Bizon Monroc, A. Bušić, D. Dubuc, J. Zhu

Towards fine tuning wake steering policies in the field: an imitation-based approach

Journal of Physics: Conference Series **2767** (2024) 032017 doi:10.1088/1742-6596/2767/3/032017

Towards fine tuning wake steering policies in the field: an imitation-based approach

C Bizon Monroc [1,2], A Bušić [2], D Dubuc [1] and J Zhu [1]

[1] IFP Energies nouvelles, 1 et 4 avenue de Bois-Préau, 92852 Rueil-Malmaison, France and Rond-point de l'échangeur de Solaize, BP 3, 69360 Solaize, France
[2] Inria and DI ENS, École Normale Supérieure, CNRS, PSL Research University, Paris, France

E-mail: claire.bizon-monroc@inria.fr

Abstract. Yaw misalignment strategies can increase the power output of wind farms by mitigating wake effects, but finding optimal yaws requires overcoming both modeling errors and the growing complexity of the problem as the size of the farm grows. Recent works have therefore proposed decentralized multi-agent reinforcement learning (MARL) as a model-free, data-based alternative to learn online. These solutions have led to significant increases in total power production on experiments with both static and dynamic wind farms simulators. Yet experiments in dynamic simulations suggest that convergence time remains too long for online learning on real wind farms. As an improvement, baseline policies obtained by optimizing offline through steady-state models can be fed as inputs to an online reinforcement learning algorithm. This method however does not guarantee a smooth transfer of the policies to the real wind farm. This is aggravated when using function approximation approaches such as multi-layer neural networks to estimate policies and value functions. We propose an imitation approach, where learning a policy is first considered a supervised learning problem by deriving references from steady-state wind farm models, and then as an online reinforcement learning task for adaptation in the field. This approach leads to significant increases in the amount of energy produced over a lookup table (LUT) baseline on experiments done with the mid-fidelity dynamic simulator FAST.Farm under both static and varying wind conditions.

1. Introduction

Wind farms are subject to the so-called "wake effect": when a wind turbine extracts energy from the wind, the wind speed downstream decreases and its turbulence increases. This leads to sub-optimal conditions for the energy production of the turbines located downstream. This can be mitigated by wake steering strategies, that use controllable actuators on upstream turbines to deflect the wake away from downstream turbines [1]. In this article, we focus on strategies relying on yaw misalignment, that increase the angle between the rotor and the wind direction (or "yaw") to redirect the wake.

Formally, we consider a wind farm with M turbines. The spatial organization of the turbines in the farm is called a layout. We assume that we observe the freestream wind conditions at the entrance of the wind farm $\mathbf{w} = (u_\infty, \phi_\infty)$ with u_∞ and ϕ_∞ respectively the wind speed and wind direction. For a space of admissible yaw angles $\mathcal{Y}$ defined to respect physical constraints on the wind turbine, we consider the vector $\gamma \in \mathcal{Y}^M$ of turbine yaws in the wind farm. The individual power outputs of the turbines $\{P_1, \dots, P_M\}$ are then a function of $\mathbf{w}$ and γ. We want to find the optimal yaws $\gamma^* : \gamma^* = \arg\max_\gamma P(\gamma, \mathbf{w})$ with $P(\gamma, \mathbf{w}) := \sum_{i=1}^M P_i(\gamma, \mathbf{w})$. Designing

efficient methods of cooperative control to find optimal yaw angles is however a challenging task. Classical optimization algorithms can be fed a model of the wind farm and output the yaws that maximize its power output estimation. This method has been used to extract a lookup table (LUT) of optimal yaws as a function of wind conditions [1], which can then be deployed in the field. Yet its performance is limited by the fidelity of the model: without feedback, it cannot recover from model inaccuracies, and the yaws deployed have indeed turned out to be sub-optimal in certain field campaigns [2].

To overcome these limitations, several multi-agent Reinforcement Learning (RL) methods have been proposed as model-free and data-based alternatives for wind farm control via wake steering [3]. In RL, learning agents infer the best actions solely by observing the system's responses to input changes. In the last decade significant progress has been achieved by RL methods for many sequential decision-making problems with successful applications in various fields including games, robotics or biology. By formulating farm power output maximization as a distributed optimization problem, decentralized multi-agent RL solutions have led to significant increases in total power production on experiments with both static [1, 4, 5] and dynamic wind farms simulators [6, 7]. In particular, decentralized learning approaches bypass the exponential dependency of the search space on the number of turbines, and promise to be more tractable by modeling every turbine as an agent following a local learning algorithm [5, 7–9]. Yet new evaluations on dynamic simulations suggest that convergence time remains too long for online learning on real wind farms [7, 10].

As an improvement, baseline policies obtained by optimizing through steady-state models can be fed as inputs to an online reinforcement learning algorithm: this can both improve the speed of adaptation to changing wind conditions and serve as a safe reference for an algorithm as it searches better policies.

We propose an imitation approach, where learning a policy is first considered a supervised learning problem by deriving references from steady-state wind farm models, and then as an online reinforcement learning task for adaptation in the field. This approach allows us to significantly increase the amount of energy produced over a LUT baseline on experiments done with a 3 turbines wind farm in the dynamic simulator FAST.Farm [11] under varying wind conditions,

2. Background: Multi-Agent Reinforcement Learning for wind farm control

In reinforcement learning (RL), agents try to directly learn the best mapping from states to (probabilities on) actions by interacting with an environment. Formally, we define a Markov Decision Process (MDP) $\{S, A, r, P\}$, with S the state space, A a discrete action space, P the matrix of transition probabilities of the environment and $r : S \times A \to \mathbb{R}$ a reward function. We write $A(s)$ the subset of actions $a \in A$ available in state s. An agent interacts with the environment by following a stochastic policy $a \sim \pi(s), s \in S, a \in A(s)$, where $\pi(a|s)$ is the probability of choosing action a when in state s. If the policy is deterministic, there exists an a' for which $\pi(a'|s) = 1$ and we directly write $a' = \pi(s)$. The agent's goal is then to find a policy π^* that maximizes the expectation of its infinite-horizon discounted reward, or discounted return:

$$\max_{\pi} \mathbb{E}[J|d_0] \quad J := \sum_{k=0}^{\infty} \beta^k r(s_k, a_k)$$

with d_0 a distribution over initial states, $0 < \beta < 1$ the discount factor, s_0 the initial state, and $\{s_k, a_k\}_{k=0...\infty}$ the trajectory of the agent in the environment under policy π. For a policy π, we define the state value function V_π as:

$$V_\pi(s) := \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \beta^k r(s_k, a_k) | s_0 = s \right]$$

with $a_k \sim \pi(s_k)$. For any state s, $V_\pi(s)$ is therefore the expected value of following policy π starting from state s. The value function can be conditioned on taking an action a in s, in which case it becomes the state-action value function $Q_\pi(s,a) = \mathbb{E}_{\pi,s'}\left[r(s,a) + \gamma V_\pi(s') \mid s,a\right]$. It is a solution of the fixed point Bellman Equation:

$$V_\pi(s) = \mathbb{E}_\pi\left[r(s,a) + \beta V_\pi(s') \mid s\right] \tag{1}$$

with $a \sim \pi(s)$ and $s' \sim P_{s,a}$. The optimal state value function V^* can then be introduced as : $\forall s, \quad V^*(s) = V_{\pi^*}(s) = \max_\pi V_\pi(s)$.

Several RL algorithms can be applied to wind farm control problems, but recent work has focused on actor-critic approaches [12]: Deep Deterministic Policy Gradient (DDPG) is used in [13–16], Proximal Policy Gradient (PPO) in [8], and a custom actor-critic method is developed by [17]. In the following, we briefly introduce this family of algorithms. Among the different strategies to learn an approximation of the optimal policy π^*, actor-critic methods directly update a policy parameter $\theta \in \mathbb{R}^d$ defining a policy $\pi_\theta : S \to \pi_\theta(a|s)$, with the goal to maximize the expectation of the total return $EJ(\theta) = \mathbb{E}_{\pi_\theta}[J|d_0]$. To avoid the explosion of variance that would be caused by a direct estimation of the objective in the field - recall that the policy is stochastic, actor-critic methods rather maintain an estimate of V_{π_θ} or Q_{π_θ}, the value functions for the current policy π_θ. The value function V_π can then be estimated by a function V_ν, parameterized by $\nu \in \mathbb{R}^b$, for a $b \in \mathbb{N}$. Recall that V_π satisfies (1), so that the learning objective for V_ν can be defined by the minimization of a squared error called the Temporal Difference (TD) δ (2) for any state transition (s, r, s'):

$$\delta(s,r,s') = r(s,a) + \beta V_\pi(s') - V_\pi(s), \qquad \mathcal{L}_\nu = \delta(s,r,s')^2 \tag{2}$$

with $a \sim \pi(s)$ and $s' \sim P_{s,a}$. Much of the recent progress in applying reinforcement learning to various problems has relied on using deep neural networks for parameters θ and ν.

When several agents interact in the same environment to maximize a shared reward however, we say that this is a task of cooperative multi-Agent RL (MARL) [18]. The agents can equally be thought of as solving a distributed optimization problem [19]. Problems of this type are commonly formalized with an extension of the MDP model called the decentralized partially observable MDP (Dec-POMDP): instead of observing the global state $s \in S$, every agent $i \in \{1, \ldots, M\}$ only has access to a partial observation of the global state defined by a function f_i. The $s_i = f_i(s) \in S_i$ are known as the local states, and every agent learns a local policy $\pi_i(a_i|s_i)$. For one agent, all others can be considered part of the environment, and existing standard RL algorithms can therefore be used to learn an optimal local policy. If all agents learn at the same time however, each of them is faced with an ever evolving environment, and the convergence guarantees of RL algorithms developed for single-agent problems no longer hold. Yet letting agents run these algorithms on their local observations with no centralized control and no other communication than the shared reward, an approach referred as *Independent Learning*, has recurrently produced good empirical results on various experiments [20] from simple grid tasks [21] to complex games [22].

3. Application to wind farm control

The wake steering problem for wind farm control can be seen as a cooperative multi-agent RL problem. At each time-step, every agent i observes a local state made of freestream wind measures $\mathbf{w} = (u^\infty, \phi)$ and its local yaw γ_i, and takes a local action commanding an increase or decrease of this yaw. All agents then receive a reward from the system, signaling whether their collective action has improved the total power output.

Under the *Independent Learning* approach introduced above, learning can be done in a decentralized manner with M RL agents separately learning local optimal policies to maximize

a shared reward. And indeed, this approach has been successfully applied to the wind farm control problem: significant increases were obtained by [5, 9, 23] on experiments with steady-state models like FLORIS [24] and PyWake [25], and by [7, 10] on mid-fidelity dynamic models like and FAST.Farm [11] and the control-oriented WFSim [26], all using independent learners.

Few experiments have so far evaluated RL methods on simulators with both dynamic wake propagation and turbulent wind inflow, but preliminary results also point towards decentralized learners being faster than centralized alternatives. In [27], an RL approach for wake steering where a centralized learner controls 3 turbines in a row of 4 was tested on HAWC2Farm [28] with a turbulence intensity of 7.5%. When the wind direction was aligned with the turbine row, which corresponds to the case with the largest wake effect, no significant increase in total power production was observed within the first 72h ($\phi_\infty = 2°$) and 96h ($\phi_\infty = 0°$) of the simulation. In [10], a similar experiment was done in FAST.Farm with 3 aligned turbines under $\phi_\infty = 0°$ and a turbulence intensity of 8%. Decentralized learners achieve an increase of 15% over the baseline in the first 96h, finally converging towards the optimal yaws in 125h. In [7], we introduced another decentralized approach called Delay-Aware Fourier Actor Critics (DFAC) and evaluated it on WFSim. We now test it our 3-turbine FAST.Farm experiment. The case considered has 3 NREL 5 MW turbines with a rotor diameter of $D = 126$m aligned in a row, and separated by a distance of $4D$. In order to replicate realistic wind conditions, we use the turbulent-wind simulator TurbSim [29] to simulate a time series of 3D wind velocity vectors in the flow field. Our wind inflow has an average freestream velocity $u_k^\infty = 8m/s$ with a turbulence of 8%. The wind is directed orthogonally to the axis of the turbine rows, i.e $\phi_k = 0°$. The sampling period in FAST.Farm is set to 3s. DFAC takes into account wake propagation time by using a delayed reward collection module. At each iteration the actor-critic parameters of agent i are only updated after a delay $rd_i(\mathbf{w})$ dependent on the position of the corresponding turbine in the farm, the distance to its neighboring turbines, the current freestream wind speed and direction $\mathbf{w}$, and a safeguard multiplier $m > 0$ which is an hyperparameter of the method. Further details about the algorithm are found in [7]: in the subsequent work, we use $m = 7$, continuous action spaces $[-1, 1]$ and for all other hyperparameters we take the default values introduced in the paper. Because DFAC has a stochastic component directing the yaw exploration in the system, we run it 4 times, each time initializing the controller with a different random seed. At the turbine level, the yaw command computed by the DFAC algorithm at every iteration is directly sent to a DISCON wind turbine controller interface. We plot the one-hour averaged power production outputs for the 4 different runs in Fig. 1. Starting from a greedy strategy where all turbines are facing the wind, DFAC is able to find the optimal yaws in 35h simulated hours, for a final increase of 21% over a greedy baseline, and a 3% increase over a LUT baseline derived with FLORIS. For comparison purpose, we will then use the DFAC algorithm as our starting point

Although these experiments point towards a method that can learn and adapt faster, the convergence time remains too slow for a deployment under real wind conditions. The experiments are made under stationary wind, with the mean of the wind velocity staying constant across the simulation, and the algorithm goes through a relatively long period of exploration. This is in part because it learns from scratch in the environment, meaning that it does not exploit any prior knowledge of the system. Several pre-training methods can be used to speed up learning and prepare a transfer towards more realistic wind condition. Because the algorithm is model-free and only interacts with the inputs and outputs of the system as a black box, any wind farm simulator can be used for pre-training RL algorithms, and the learned policies then used as initialization in the real farm. We tested this approach on our 3-wind turbine layout: DFAC was first trained on a FLORIS simulation of the farm, and continued to learn in the FAST.Farm environment. These experiments failed, with DFAC instead exhibiting a decrease in power production at the beginning of the FAST.Farm simulation. The question of adaptation

of RL algorithms from simulation to the real case is indeed not a trivial one. This problem is known in the RL literature as "knowledge forgetting"[30], but to the best of our knowledge it has not been discussed in the literature of RL algorithms for wind farm control, where most algorithms are both trained and evaluated on the same simulator [3].

To solve this issue we want a method that can:

(A) From any model, efficiently learn "good-enough" values on the full state-action space offline
(B) Allow for straightforward weight transfer to the field without significant performance loss.

We will see in the next section that for any set of optimal yaws, it is easy to derive analytic expressions for the policies and their respective value functions. Finding correct weights that approximate these functions can then be done by any supervised learning procedure.

4. Imitation-based DFAC

Let us consider a LUT mapping wind conditions to yaws in a specific farm $\mathcal{M}_{\texttt{Farm}} : (\mathbf{w}) \to (\gamma_1^{\mathcal{M}}, \dots, \gamma_M^{\mathcal{M}})$. This can be obtained through an optimization routine with a steady-state model. Then, the optimal local policies with respect to the LUT simply return the largest allowed step towards the LUT yaw in every state. If the policies of all agents are following this simple rule, every step will lead towards an increase in the total power output. Let us assume that this step is always rewarded by the maximal reward r_{ub}, the corresponding value functions can then also be derived any agent i and any local state s_i: for all $s_i = (\gamma_i, \mathbf{w}) \in S_i$,

$$\pi^{i\mathcal{M}}(s_i) = \begin{cases} \min(\Delta, \mathcal{M}_{\texttt{Farm}}(\mathbf{w})_i - \gamma_i) & \text{if } \gamma_i < \mathcal{M}_{\texttt{Farm}}(\mathbf{w})_i \\ \max(-\Delta, \gamma_i - \mathcal{M}_{\texttt{Farm}}(\mathbf{w})_i) & \text{if } \gamma_i > \mathcal{M}_{\texttt{Farm}}(\mathbf{w})_i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$V^{i\mathcal{M}}(s_i) = r_{ub} \sum_{k=0}^{T-1} \beta^k = r_{ub} \frac{1-\beta^T}{1-\beta} \quad \text{where } T = \left\lceil \frac{|\mathcal{M}_{\texttt{Farm}}(\mathbf{w})_i - \gamma_i|}{\Delta} \right\rceil \tag{4}$$

where with Δ the upper bound on absolute change in the yaw command at every iteration. The learned policy is therefore deterministic.

Note however that the policy deployed in the field will not be deterministic: to improve upon the initial baseline, agents must explore different actions for every state. DFAC therefore samples from a Gaussian distribution centered around the current estimate of optimal actions, with an adaptive standard deviation parameter σ. The assumption that all policies keep increasing the production until they reach the LUT yaws will therefore not hold even when the yaws are optimal. In particular, eq. (4) is a lower bound on the value function of the Gaussian policy whenever $\gamma_i = \mathcal{M}_{\texttt{Farm}}(\mathbf{w})_i$, and an upper bound elsewhere: it is overestimating the difference between states with the highest and lowest values. Formally, it will not satisfy the fixed point equation eq. (1) for a stochastic policy. To adjust the estimate, we can train the learned value functions to minimize their Temporal Difference losses eq. (2) on a small additional sample of rewards estimates collected under the Gaussian policies. These reward estimates can be easily computed with a steady-state model.

We can therefore adopt the following methodology:

(i) Generate a training dataset from the model-derived optimal policy and value functions $\pi^{i\mathcal{M}}$ and $V^{i\mathcal{M}}$
(ii) Offline, learn parameterized functions V_ν and π_θ approximating the model-derived optimal policies and value functions by minimizing the Mean Squared Error loss
(iii) Offline, adjust estimate by minimizing the TD error with rewards estimated under stochastic policies

(iv) Online, start learning the RL functions in the dynamic environment from the policies learned offline

We detail our methodology in the following paragraphs. A pseudo-code is also provided in algorithm 1.

4.1. Generate a training dataset

Algorithm 1 Imitation learning for DFAC (IDFAC)

Require: Number of agents M, $T \geq 0$, Wind distribution $\mathcal{W}$, RL parameters $\beta \in (0,1)$, $C \in \mathbb{N}^*$, $E < C$ Batch size B, Steady state model `model`, Standard deviation σ
Init Policy and value function parameters $\{\theta_i\}_{i=1\dots M}$, $\{\nu_i\}_{i=1\dots M}$, $\mathcal{D} = \emptyset, \mathcal{D}^r = \emptyset$, Neural networks π_θ, V_ν
(i) Generate the training dataset
for c = 1 ... C **do**
  $\mathbf{w} \leftarrow sample(\mathcal{W})$
  $\mathcal{M}_{\texttt{Farm}}(\mathbf{w}) \leftarrow \texttt{model}.optimize\_yaws(\mathbf{w})$
  for $\gamma \in |\mathcal{Y} \cap \mathbb{Z}|$ **do**
    for i = 1 ... M **do**
      $x_i \leftarrow (\gamma_i, \mathbf{w})$
      $y_i \leftarrow \left(\pi^{\mathcal{M}}(x_i), V^{\mathcal{M}}(x_i)\right)$ eqs. (3) and (4)
      $\mathcal{D} \leftarrow \mathcal{D} \cup (i, x_i, y_i)$
    end for
  end for
end for
(ii) Learn V_ν and π_θ offline
for i = 1 ... M **do**
  $\theta_i \leftarrow \texttt{minibatchSGD}(\theta_i, \mathcal{D}, \mathcal{L}_\theta, B)$ eq. (5)
  $\nu_i \leftarrow \texttt{minibatchSGD}(\nu_i, \mathcal{D}, \mathcal{L}_\nu, B)$ eq. (5)
end for
(iii) Adjust estimate V_ν to minimize the TD error
for c = 1 ... E **do**
  $\mathbf{w} \leftarrow sample(\mathcal{W})$
  $\gamma \leftarrow sample(\mathcal{Y}^M)$
  for i = 1 ... M **do**
    $x_i = (\gamma_i, \mathbf{w})$
    $a_i \leftarrow \pi_{\theta_i}(x_i, \sigma)$
  end for
  $r \leftarrow get\_reward(\texttt{model}, \mathbf{w}, \gamma, a)$
  for i = 1 ... M **do**
    $\nu_i \leftarrow \nabla\delta(x_i, r, (\gamma_i + a_i, \mathbf{w}))$ eq. (2)
  end for
end for
Return θ, ν

We first need to extract a representative range of wind conditions. We assume that we either have a wind distribution for velocity and direction, or otherwise that we can extract SCADA wind measurements from a currently operating farm to extract an empirical distribution. The construction of the dataset then follows the following procedure: we first sample C wind conditions from the training data, with C being an hyperparameter of the method. To better

cover the space of wind conditions, we assume that velocity and direction are independent, and sample them separately. This gives a set of wind condition $\mathbf{W} = \mathbf{w}^j_{j=1,\dots,C}$.

We then use the FLORIS software to simulate the farm with a low-fidelity model, and the Serial-Refine algorithm [31] to find the optimal yaws with respect to the steady-state model. The Serial-Refine (SR) algorithm has been introduced by NREL in 2022 specifically for wake-steering optimization problems in FLORIS models, it is a search algorithm that finds solutions equivalent to the previously used default routine, but at a fraction of the computation time. Once the optimal yaws are found, we use the policy (3) and value (4) equations to create a training dataset. For every integer in the yaw space $\gamma_i \in \mathcal{Y}$, we create the corresponding pair (x_i, y_i) representing a data point and its label

$$x_i = (\gamma_i, \mathbf{w}), \qquad y_i = \left(\pi^{\mathcal{M}}(x_i), V^{\mathcal{M}}(x_i)\right).$$

For every wind condition, we obtain a dataset of $N = |\mathcal{Y} \cap \mathbb{Z}|$ points. The final dataset concatenates the $C \times N$ samples.

4.2. Learn offline

We now define the Mean Squared Error losses for the policy $\mathcal{L}_\theta$ and for the value function $\mathcal{L}_\nu$:

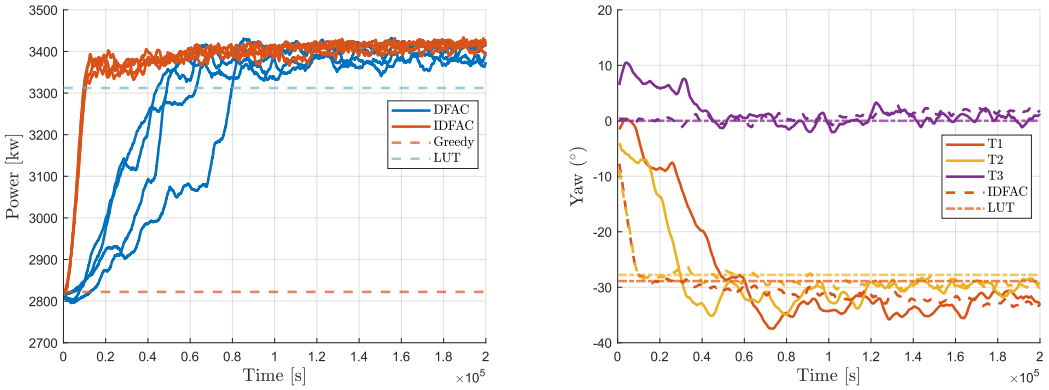
$$\mathcal{L}_\theta = \sum_{i=0}^{N} \left[\pi_\theta(x_i) - \pi^{\mathcal{M}}(x_i)\right]^2, \qquad \mathcal{L}_\nu = \sum_{i=0}^{N} \left[V_\nu(x_i) - V^{\mathcal{M}}(x_i))\right]^2. \tag{5}$$

We then use a mini-batch gradient descent algorithm ([32]) to learn the parameters θ, ν that best approximate the function.

A small subset of our dataset will now be used to adjust the value estimate (step (iii)). For a randomly chosen subset of E points x_i, with $E \ll C \times N$, we sample local actions from Gaussian distributions centered around $\pi^i_\theta(x_i)$ with a standard deviation of 1. The corresponding rewards are then computed in FLORIS, and the mini-batch gradient descent algorithm used to minimize the TD loss eq. (2).

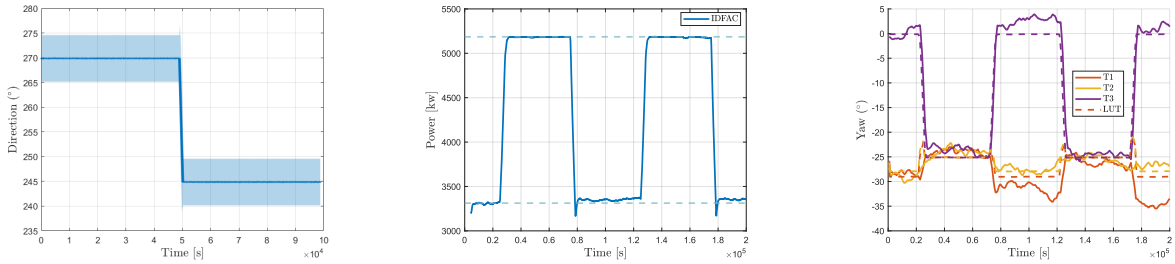
5. Results

We now evaluate our method on two FAST.Farm experiments with with our 3-turbine layout. To work on realistic wind data, we sample $C = 500$ wind conditions from SCADA wind velocity and direction measurements acquired during SMARTEOLE experimental campaigns. The base DFAC algorithm used features extracted with a Fourier basis to represent the state. To adapt it to varying wind conditions, we feed theses features to a neural network of 2 hidden layer of 81 units with a hyperbolic tangent activation. We first evaluate imitation-based DFAC (IDFAC) on our 3-turbine FAST.Farm experiment introduced in section 3. We train the policies and value functions using a batch size of 32, and 20 passes on the full data for the mini-batch gradient descent algorithm. We then deploy the algorithm online and compare it to the initial DFAC without imitation. We run IDFAC 4 times with different random seeds, and measure the 1h-average energy produced during each simulation. We compare the results with the runs of DFAC, as well as the evolution of yaws during the first experiment for each algorithm, on Fig. 1a. Both DFAC and IDFAC improve the greedy baseline by 21%, but all runs of IDFAC achieve a 18% increase in 11h of simulation time, while it takes between 14h and 22h of simulation for DFAC to reach the same threshold. Moreover, the yaws during the IDFAC experiments exhibit less oscillation during exploration, as can be seen on Fig. 1b. We then extend the imitation approach to a range of wind directions. We use a time series with a turbulence intensity of 8%. Its wind velocity is always centered around $8m/s$, while the wind direction varies between two averages in steps of 50ks, that is approximately 14h. The first mean is 270° from the NW,



(a) Evolution of total power output (b) Evolution of yaws

Figure 1: Performance of the DFAC and imitation-based DFAC (IDFAC) algorithms on the 3-turbines layout case simulated on FAST.Farm with stationary turbulent wind. 1h average of total power output (a) and yaws during the first experiment of each algorithm (b) on 56h of simulation.



(a) Turbulent wind direction (b) Averaged power output (c) Yaws

Figure 2: Performance of the DFAC and imitation-based DFAC (IDFAC) algorithms on the wind farm of the 3-turbines layout case, simulated on FAST.Farm under turbulent wind.

the direction aligned with the turbines row that corresponds to the most important wake. The second mean is 244°, a direction that causes no wake effect for the turbine row, and under which the optimal strategy is simply the greedy one. The step wind direction profile is shown on Fig. 2a, with its mean and standard deviation measured over 10 minutes. We expect a good online learning method to maintain the greedy strategy where it is optimal, and still recover from modeling errors to find the optimal yaws elsewhere. The result of running IDFAC in this experiment is shown on Fig. 2. As expected, IDFAC maintains the greedy yaws under the 244° step. Under 270°, it first explores around the LUT yaws during the first wind direction period period (28h of simulated time), and then reaches an increase of 19.23% over the greedy strategy and of 1.6% over the LUT by the second wind direction period (56h simulated time). Importantly, power production under IDFAC never falls below the LUT threshold that was used to initialize it. It only progressively increases over the baseline as the algorithm explores and learns to recover from model errors.

6. Conclusion

Integrating knowledge from wind farm models with decentralized RL methods is a promising way to design fast and adaptive algorithms, that can learn optimal wake steering strategies in

the field. To achieve this, we propose to exploit an imitation-based approach that can exploit steady-state models to derive initialization policies, and then refine them online. We validate it in two experiments on the FAST.Farm simulator: our approach accelerates learning compared to RL-only algorithms and is able to recover from model errors online even under varying wind conditions. Although our experiments on FAST.Farm under turbulent wind are good proxies for real environments, our results will need to be validated in Large Eddy Simulations (LES). Moreover, a LUT derived from a single steady-state model was used to create the training imitation dataset. Instead, learning offline from several models could guide us towards more robust policies, and such an approach could draw from the rich RL literature on learning from expert demonstrations [33].

Acknowledgments
Funding provided through the France 2030 Plan, PEPR TASE, AI-NRGY project ANR-22-PETA-0044 is gratefully acknowledged.

References

[1] Kheirabadi A C and Nagamune R 2019 *Journal of Wind Engineering and Industrial Aerodynamics* **192** 45–73

[2] Fleming P, King J, Dykes K, Simley E, Roadman J, Scholbrock A, Murphy P, Lundquist J K, Moriarty P, Fleming K, van Dam J, Bay C, Mudafort R, Lopez H, Skopek J, Scott M, Ryan B, Guernsey C and Brake D 2019 *Wind Energy Science* **4** 273–285

[3] Abkar M, Zehtabiyan-Rezaie N and Iosifidis A 2023 *Theoretical and Applied Mechanics Letters* **13** 100475

[4] Graf P, Annoni J, Bay C, Biagioni D, Sigler D, Lunacek M and Jones W 2019 Distributed reinforcement learning with admm-rl *2019 American Control Conference (ACC)* pp 4159–4166

[5] Xu Z, Geng H, Chu B, Qian M and Tan N 2020 *IFAC-PapersOnLine* **53** 12103–12108 21st IFAC World Congress

[6] Schuitema E, Buşoniu L, Babuška R and Jonker P P 2010 *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* 3226–3231

[7] Bizon Monroc C, Bušić A, Dubuc D and Zhu J 2023 Actor critic agents for wind farm control *2023 American Control Conference (ACC)* pp 177–183

[8] Kadoche E, Gourvénec S, Pallud M and Levent T 2023 *Renewable Energy* **217** 119129

[9] Stanfel P, Johnson K, Bay C J and King J 2021 *Journal of Renewable and Sustainable Energy* **13**

[10] Bizon Monroc C, Bouba E, Bušić A, Dubuc D and Zhu J 2022 Delay-aware decentralized q-learning for wind farm control *2022 IEEE 61st Conference on Decision and Control (CDC)* pp 807–813

[11] Jonkman J M, Annoni J, Hayman G, Jonkman B and Purkayastha A 2017 Development of fast. farm: A new multi-physics engineering tool for wind-farm design and analysis *35th wind energy symposium* p 0454

[12] Grondman I, Buşoniu L, Lopes G A D and Babuška R 2012 *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **42** 1291–1307

[13] Dong H and Zhao X 2021 *IEEE Transactions on Control Systems Technology* **30** 1281–1295

[14] Dong H and Zhao X 2022 *2022 European Control Conference (ECC)* 993–998

[15] Zhao H, Zhao J, Qiu J, Liang G and Dong Z Y 2020 *IEEE Transactions on Industrial Informatics* **16** 6912–6921

[16] Dong H, Zhang J and Zhao X 2021 *Applied Energy* **292**

[17] Dong H and Zhao X 2022 *IEEE Transactions on Control Systems Technology*

[18] Oroojlooyjadid A and Hajinezhad D 2019 *Applied Intelligence* **53** 13677–13722

[19] Lee D, He N, Kamalaruban P and Cevher V 2020 *IEEE Signal Processing Magazine* **37** 123–135

[20] Buşoniu L, Babuška R and De Schutter B 2010 *Innovations in multi-agent systems and applications-1* 183–221

[21] Tan M 1993 Multi-agent reinforcement learning: Independent vs. cooperative agents *Proceedings of the tenth international conference on machine learning* pp 330–337

[22] de Witt C S, Gupta T, Makoviichuk D, Makoviychuk V, Torr P H, Sun M and Whiteson S 2020 Is independent learning all you need in the starcraft multi-agent challenge?

[23] Kadoche E, Gourvénec S, Pallud M and Levent T 2023 *Renewable Energy* **217** 119129 ISSN 0960-1481

[24] Gebraad P, Teeuwisse F, van Wingerden J, Fleming P, Ruben S, Marden J and Pao L 2016 *Wind Energy* **19** 95 – 114

[25] Pedersen M M, Forsting A M, van der Laan P, Riccardo Riva L A A R, Javier Criado Risco M F M, Quick J, Christiansen J P S, Rodrigues R V, Olsen B T and Réthoré P E 2023 Pywake 2.5.0: An open-source wind farm simulation tool (DTU Wind, Technical University of Denmark) URL `https://gitlab.windenergy.dtu.dk/TOPFARM/PyWake`

[26] Boersma S, Doekemeijer B, Vali M, Meyers J and van Wingerden J W 2018 *Wind Energy Science* **3** 75–95

[27] Liew J, Göçmen T, Lio W H and Larsen G C 2023 *Wind Energy*

[28] Liew J, Andersen S J, Troldborg N and Göçmen T 2022 *Journal of Physics: Conference Series* **2265** 022069

[29] Jonkman B and Jr B 2007 *TurbSim User's Guide* (National Renewable Energy Laboratory)

[30] Zhao W, Queralta J P and Westerlund T 2020 Sim-to-real transfer in deep reinforcement learning for robotics: a survey *2020 IEEE symposium series on computational intelligence (SSCI)* pp 737–744

[31] Fleming P A, Stanley A P J, Bay C J, King J, Simley E, Doekemeijer B M and Mudafort R 2022 *Journal of Physics: Conference Series* **2265** 032109

[32] Hinton G, Srivastava N and Swersky K 2012 Lecture 6a: overview of mini-batch gradient descent

[33] Nair A, Dalal M, Gupta A and Levine S 2020 Accelerating online reinforcement learning with offline datasets