# Actor Critic Agents for Wind Farm Control

Claire Bizon Monroc, Ana Bušić, Donatien Dubuc, Jiamin Zhu

# Actor Critic Agents for Wind Farm Control

Claire Bizon Monroc, Ana Bušić, Donatien Dubuc, Jiamin Zhu

***Abstract*— The power output of a wind farm is influenced by wake effects, a phenomenon in which upstream turbines facing the wind create sub-optimal conditions for turbines located downstream. Yaw misaligning strategies have been shown to increase total production. Yet designing efficient methods of cooperative control to find optimal yaw angles is a challenging task. Classical optimization methods become intractable as the size of the farm grows, do not recover from model inaccuracies and ignore the dynamic propagation of the wind inflow in real conditions. Reinforcement learning methods can provide a model-free alternative, but raise issues of scalability when the control is centralized. Existing decentralized RL methods have been shown to significantly increase power production under dynamic conditions, but relied on tabular methods with state and action space discretization. To accelerate convergence, we employ an actor-critic algorithm with linear function approximation for decentralized cooperative yaw control. We validate our method in dynamic simulators for wind farms with up to 32 turbines, and show empirically that, compared to previous tabular algorithms, our method is faster and scales to larger wind farms.**

## I. Introduction

The power output of a wind farm is influenced by wake effects, a phenomenon in which upstream turbines facing the wind create sub-optimal conditions for turbines located downstream. Misaligning the yaw, defined as the angle between the rotor and the wind direction, is an efficient strategy to deflect the wake away from downstream turbines. This technique is known as wake steering, and has been shown to increase total production compared to the naive greedy strategy where all turbines face the wind [1] [2].

Designing efficient methods of cooperative control to find optimal yaw angles is a challenging task. First, the complexity of this optimization problem increases with the number of turbines in the wind farm, making centralized control strategies quickly intractable for real time optimization. Second, several classical model-based optimization methods have been proposed [3], but they cannot recover from model inaccuracies in the field. Finally, deployment of any control method for real-time optimization on wind farms requires accounting for the dynamic propagation of the wind inflow.

To address the first issue, we formulate farm power output maximization as a multi-agent, distributed optimization problem. This solution is computationally time-efficient, has the potential to be applied on any wind farm layout without any change in the structure of the algorithm, and has already been successful in static simulations [3]. For the second issue, we can suppress reliance on inaccurate modeling by designing adaptive solutions that learn online. Reinforcement learning (RL) methods provide a model-free, data-based alternative: they learn to predict the optimal yaws from observations of the system's output. In a previous work, we have developed a decentralized RL algorithm and validated it in a mid-fidelity simulation environment under realistic dynamic conditions [4]. To account for wake propagation times and thus address the third issue, a delayed update component was added to the RL method, allowing agents to take into account the time needed for their input changes to impact neighboring turbines. This algorithm however relies on state and control space discretization. This comes with an important cost in terms of parameters to learn, and raises issues regarding the algorithm's ability to scale to larger wind farms. We show that more efficient learning agents can be designed under the same principles of decentralization and delay-awareness. We employ actor critic agents learning in parallel with linear function approximation, and test our method in WFSim, a control-oriented wind farm simulator that takes into account wake propagation dynamics which has been validated against large eddy simulations [5].

We propose a decentralized actor critic method for yaw control of large wind farms under dynamic wake conditions. Numerical experiments on WFSim under stationary wind conditions for wind farms with up to 32 turbines show that this method has great scaling potential and achieves faster coordination and convergence than existing RL algorithms, leading to more important increases in energy production.

The remainder of this paper is organized as follows: In Section II, we provide background information on reinforcement learning and introduce actor-critic methods. We then show how they can be used to solve a cooperative yaw control problem for wind farm output power maximization in Section III. We finally introduce our simulation setup, and present our experimental results on 4 different WFSim simulations for various layouts and farm sizes in Section IV. Section V summarizes our results and discusses possible paths forward.

## II. Background: Reinforcement Learning

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 $a \in A$

C. Bizon Monroc is with Inria and DI ENS, École Normale Supérieure, CNRS, PSL Research University, Paris, France and IFP Energies nouvelles. E-mail: claire.bizon-monroc@inria.fr

A. Bušić is with Inria and DI ENS, École Normale Supérieure, CNRS, PSL Research University, Paris, France

D. Dubuc and J. Zhu are with IFP Energies nouvelles

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 $s$. The agent's goal is then to find a policy $\pi^*$ 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 $\{s_k, a_k\}_{k=0\dots\infty}$ a trajectory of the agent in the environment under policy $\pi$.

For a policy $\pi$, we define the state value function $V$ as:

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

with $a_k \sim \pi(s_k)$. For any state $s$, $V_\pi(s)$ is therefore the expected value of following policy $\pi$ from state $s$. It is a solution of the fixed point Bellman Equation:

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

with $a \sim \pi(s)$ and $s' \sim P_{s,a}$. The state-action value function, or q function $Q$, is:

$$Q_\pi(s,a) := \mathbb{E}\left[\sum_{k=0}^{\infty} \beta^k r(s_k, a_k)|s_0 = s, a_0 = a\right]$$

It is the expected value of taking action $a$ in state $s$, and then following policy $\pi$. We define the optimal state value function $V^*$ such that:

$$\forall s, \quad V^*(s) = V_{\pi^*}(s) = \max_{\pi} V_\pi(s).$$

RL algorithms follow different strategies to learn an approximation of the optimal policy $\pi^*$. State-value based methods try to learn the optimal state-action value function $Q^*$ by exploiting the fixed-point property of the Bellman Equation (1). The optimal policy can then be extracted from the learned value functions. Exhaustive search in the case of continuous action spaces is however intractable. Policy gradient methods can solve this issue by directly learning a policy $\pi_\vartheta$ parameterized by a vector $\vartheta \in \mathbb{R}^d$. The learning objective is then the maximization of the expectation of the total return $EJ = \mathbb{E}[J|d_0]$ over the parameter space of the policy. If we can estimate the gradient of the return $\nabla_\vartheta EJ$ at every iteration, a simple gradient ascent would ensure convergence towards a local optimum under minimal assumptions. This can indeed be done with the Policy Gradient Theorem [6], which rewrites this gradient as an expectation:

$$\nabla_\vartheta EJ = \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \nabla_\vartheta \pi(a|s) Q_\pi(s,a) du\, da \tag{2}$$

with $d^\pi(s) = \sum_{k=0}^{\infty} \gamma^k P(s_k = s|d_0, \pi)$ the discounted state distribution under policy $\pi$ [6]. Policy gradient methods can therefore inherit strong convergence properties from the chosen gradient ascent algorithm, but tend to suffer from huge variance in their gradient estimates.

Actor-critic methods combine the advantages of both policy gradient and state-value based methods. Like the former, they easily adapt to continuous action spaces through a parameterized policy - or actor. But building on the latter, they can exploit the value function estimator - or critic, to decrease the variance in the policy gradients [7]. Actor-critic methods can exploit the relationship between the gradient of $J$ and the state-value $Q$ function given in (2). Indeed, since $Q$ is unknown, one could learn a function $h : S \times A \to \mathbb{R}$ to estimate it. It is possible to design an approximation function $h$ such that replacing the unknown value $Q_\pi(s,a)$ in (2) by the estimate $h(s,a)$ keeps the estimate unbiased [6]. Such a function would have an expected value of zero for each state, and can be thought of as an estimate of the advantage function $A_\pi$:

$$A_\pi := Q_\pi(s,a) - V_\pi(s) \tag{3}$$

Intuitively, the gradient would point towards zones of the policy parameter space associated with relative higher returns for each state compared to a baseline given by $V_\pi(s)$. Instead of learning this function, we can note that the Temporal Difference - TD - error:

$$\delta(s, r, s') = r(s,a) + \beta V_\pi(s') - V_\pi(s) \tag{4}$$

is an unbiased estimate of $A_\pi$ since $\forall (s,a), Q_\pi(s,a) = \mathbb{E}[r(s,a) + \beta V_\pi(s')]$. A new gradient estimate can be rewritten:

$$\begin{aligned}\nabla^\delta_\vartheta EJ &= \int_{\mathcal{S}} d^\pi(s) \int_{\mathcal{A}} \pi(a|s) \nabla_\vartheta \ln \pi(a|s) \delta(s,r,s') du\, da \\ &= \mathbb{E}_{s,a\sim\pi,P_{s,a}}[\nabla_\vartheta \ln \pi(a|s) \delta(s,r,s')]\end{aligned} \tag{5}$$

with the switch to log probabilities allowed by the chain rule applied to $\nabla_\vartheta \ln \pi(a|s)$. Written as an expectation, this formula allows us to approximate gradients from samples of any trajectory following policy $\pi$: $\hat{\nabla}^\delta_\vartheta \mathbb{E}[J] = \frac{1}{T}\sum_{t=0}^{T-1} \nabla_\vartheta \ln \pi(a_t|s_t)[r(s_t,a_t) + \beta V_\pi(s_{t+1}) - V_\pi(s_t)]$. This is a one-step equivalent of the approach used in Advantage Actor Critic (A2C) [8].

The state value function of the current policy $V_\pi$ can be estimated by a function $V_\theta$, parameterized by $\theta \in \mathbb{R}^b$. Recall that $V_\pi$ satisfies (1), so that the learning objective for $V_\theta$ can be defined by the minimization of the squared TD error (4) for any state transition $(s, r, s')$:

$$\mathcal{L}_\theta = \delta(s,r,s')^2 \tag{6}$$

with $a \sim \pi(s)$ and $s' \sim P_{s,a}$.

Taking a gradient ascent step with regard to $\theta$ and $\vartheta$ at every iteration, we derive the update rules of an actor-critic algorithm:

$$\begin{aligned}\delta_k &= r_k + \beta V_{\theta_k}(s_{k+1}) - V_{\theta_k}(s_k) \\ \theta_{k+1} &= \theta_k + l_{v,k} \delta_k \nabla_\theta V_{\theta_k}(s_k) \\ \vartheta_{k+1} &= \vartheta_k + l_{q,k} \delta_k \nabla_\vartheta \ln \pi_{\vartheta_k}(a_k|s_k)\end{aligned} \tag{7}$$

with $l_{v,k}$ and $l_{q,k}$ the deterministic, non-increasing learning rates of respectively the critic and the actor.

## III. Application to Wind Farm Control

We consider a wind farm with $M$ turbines. Each turbine's position can be described by a pair of coordinates: $(C_{x,i}, C_{y,i}), i = 1 \dots M$. At each time step $k$, we assume that the freestream wind conditions $\mathbf{w_k} = (u_k^\infty, \phi_k)$ are measured at the entrance of the farm, with $u_k^\infty$ the speed and $\phi_k$ the direction of the wind. For a space of admissible angles $\mathcal{Y}$, each turbine $i$ has current yaw $\gamma_{i,k} \in \mathcal{Y}$ and generates power $P_{i,k}$. We want to maximize the total power output of the wind farm.

### A. A multi-agent optimization problem

This problem can be seen as a distributed optimization task, where every turbine is an agent interacting with the environment and all maximize a common objective. It should be noted that in a wind farm, sensor data is usually centralized, so it can be assumed that there is full communication of all relevant information between agents. A decentralized approach can however allow us to turn a hard to scale, high dimension optimization problem into several tractable ones. Within a reinforcement learning framework, we have $M$ agents learning local optimal policies to maximize a shared reward: this is known as cooperative multi-agent reinforcement learning (MARL) [9]. As learning agents all concurrently interact with the same environment, the stationarity assumptions supporting convergence guarantees for any individual RL learner no longer hold. Yet running single-agent algorithms in parallel on every agent has been empirically shown to converge to a good equilibrium in certain settings [10], and a good proof-of-concept of this approach has been provided by [11] for Q-learning on the wind farm optimization case.

In this cooperative MARL formalization, each agent observes his local state and action, and receives information about the production of other turbines through the shared reward function. We therefore consider $M$ state spaces $S_{i,1\leq i\leq M}$, $S_i = \mathcal{Y} \times \mathbb{R}^2$, and $M$ action spaces $A_{i,1\leq i\leq M}$, $A_i = [-\delta\gamma, +\delta\gamma] \in \mathbb{R}$. $\delta\gamma$ serves as an upper bound on the difference in yaws for an agent between two timesteps, and matches the real constraints and limitations on turbine yaw actuation. At every timestep $k$, the agent $i$ observes the local state:

$$s_{i,k} = [\gamma_{i,k}, \mathbf{w_k}]^T, \quad \gamma_{i,k} \in \mathcal{Y},$$

and every chosen action $a_{i,k} \in A_i$ leads to an update in the agent's yaw: $\gamma_{i,k+1} = \gamma_{i,k} + a_{i,k}$. The learning algorithm governs the choice of the action.

### B. Delayed Reward

Because of wake propagation time, the impact of an agent's action on other turbines in the farm is not immediately observable. The agents therefore act in a delayed reward environment: the reward associated to an action taken at timestep $k$ cannot be collected until after the next timestep. We define $rd$ the reward delay, or number of timesteps before a reward becomes available. If $rd$ is known and deterministic, we can simply restore temporal matching by delaying algorithm updates accordingly, as shown in [12] in the case of tabular Q-learning with delayed actions. A similar strategy has been tested for wind farm control under dynamic conditions on a previous paper from the authors [4].

We therefore modify updates in (7): at each timestep $k$, updates are performed for the action $a_{k-rd}$ selected at timestep $k - rd$, but whose associated reward has just been collected at timestep $k$. Good empirical results were achieved by approximating the propagation delay between two turbines in the farm using Taylor's frozen wake hypothesis [13][14]. Formally, a wake delay matrix $D_{i,j}(\mathbf{w})$ for all $0 \leq i, j \leq M$ was constructed, where every $D_{i,j}(\mathbf{w})$ is the estimated propagation time from upstream turbine $i$ to downstream turbine $j$ for wind conditions $\mathbf{w}$.

$$D_{i,j}(\mathbf{w}) = \begin{cases} m\frac{c_{i,j}}{u^\infty} & \text{if } j \text{ downstream and } c_{i,j} \leq d_{\text{cutoff}} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

with $c_{i,j}$ the downwind distance between the 2 turbines, $\mathbf{w}$ the freestream wind conditions at the entrance of the farm, and $d_{\text{cutoff}}$ a cutoff distance reflecting the assumption that wake effects should be negligible above a certain distance. A safeguard multiplier $m > 1$ is added to correct Taylor's estimate $\frac{c_{i,j}}{u^\infty}$, which is bound to be an underestimation of the real time of propagation as wind velocity decreases in the wake. The reward delay $rd_i(\mathbf{w})$ for each turbine $i$ under wind condition $\mathbf{w}$ is then the number of time steps of the largest wake delay

$$rd_i(\mathbf{w}) = \left\lceil \max_j \frac{D_{i,j}(\mathbf{w})}{h} \right\rceil, \tag{9}$$

with $h$ the sample frequency.

We now seek to design a reward function returning relatively higher values for higher increases. A reward proportional to the observed output power is an intuitive solution, but initial experiments done with such a reward were inconclusive. Indeed, agents operate in a highly non-stationary environment, with several sources of uncertainty in output measures: (1) the lack of observation of other agents at every action choice, (2) the delayed observation of impact (3) the influence of changing nominal power output under turbulent or non-stationary wind conditions on the reward assignment. To decrease the variance of the reward associated to a state-action pair, we design a staircase function. Moreover, like in [4], we measure the power generation of an agent's downstream turbines at the moment of the estimated impact, and consider increases in percentages rather than raw values. We therefore define our reward function, for $i = 1, \cdots, M$ and given reward boundaries $r_{lb}$, $r_{ub}$ such that $\forall a \in A_i, s \in \mathcal{S}_i$, $r(s_i, a_i) \in [r_{lb}, r_{ub}]$.

$$\begin{aligned} r_{i,k} &= \min(\max(\text{sign}(\rho) \left\lfloor \frac{|\rho|}{\Delta} \right\rfloor, r_{lb}) r_{ub}) \\ \rho &= \frac{V_2^i - V_1^i}{V_1^i} \end{aligned} \tag{10}$$

with $\Delta > 0$ a tolerance parameter controlling the size of each step, $V_1^i = \sum_{j=1}^M P_{j,k}$ and $V_2^i = \sum_{j=1}^M P_{j,k+D_{i,j}(\mathbf{w})}$.

### C. Fourier Basis Actor Critic

Our goal is to learn a value function $V_\theta$ and a policy function $\pi_\vartheta$, parameterized by respectively $\theta \in \mathbb{R}^d$ and $\vartheta \in \mathbb{R}^b$. As detailed in II, this can be done by updating these parameters according to (7). Several families of functions have been used in the literature [15]. Neural networks have become the most popular option for a growing number of applications [16], but their performances for RL are highly dependent on the choice of hyperparameters and require a significant amount of tuning [17]. Moreover, as we will see in IV, simpler methods are sufficient to obtain good empirical results on our problem. A widely used approach has been linear function approximations, which represent the function of interest as a linear combination of a set of features extracted from the observed state. The set of the non-linear functions mapping any observed state to an extracted feature are known as basis functions. Linear function approximation then results in a simple update rule, even though the basis functions may be arbitrarily complex ([18]). With a linear parametrization for our state value function and policy, we have:

$$\forall s \in \mathcal{S}, \quad V_\theta(s) = \theta^T \psi(s), \quad \pi_\vartheta(s) = \mathbb{P}(\vartheta, \psi(s))$$

with $\psi = [\psi_1, \dots, \psi_d]$, $\psi_i : \mathcal{S} \to \mathbb{R}$ a basis function, that is a feature extraction function giving a representation of the state, and $\mathbb{P}(\vartheta, \psi(s))$ a probability distribution dependent on $s$ and $\vartheta$. A commonly chosen parametrization of policies for continuous action spaces is that of Gaussian policies [8]. We can define $\pi_\vartheta(s) = \mathcal{N}(\mu(s), \sigma^2)$, with expectation $\mu(s) = \vartheta_\mu^T \psi(s)$, and $\sigma^2$ a state-independent variance controlling exploration such that $\vartheta_{mu} \in \mathbb{R}^d$, $\vartheta = [\vartheta_\mu, \sigma] \in \mathbb{R}^{d+1}$. The design and choice of appropriate basis functions for linear function approximation methods is a critical part of the algorithm, and has long been discussed in the reinforcement learning literature [19][15]. Popular choices have included the use of hand-designed representation, polynomial features or boolean tile coding [15]. Good empirical results have also been obtained with the Fourier basis on several benchmark continuous control tasks for state-value based methods [18][15]. After doing experiments on static simulations (FLORIS) to compare several basis on our problem, we found the Fourier basis to achieve faster and more stable convergence. We therefore use linear function approximation with the Fourier basis for our actor-critic agents, and detail its implementation in the following paragraphs.

The *n*th degree Fourier expansion of a function $f$ of period $T$ is:

$$\mathcal{S}_n(f)(x) = \frac{a_0}{2} + \sum_{k=1}^{n} \left[ a_k \cos\left(k \frac{2\pi}{T} x\right) + b_k \sin\left(k \frac{2\pi}{T} x\right) \right]$$

with $\{a_k, b_k\}_{k \in \mathbb{Z}}$ the Fourier coefficients, which can be computed as a function of $f$. If we desire to estimate $f$, the coefficients can be considered as unknown parameters that we need to learn. When $f$ is not known to be periodic but is bounded, one can rescale $x$ to $[0, 1]$, and restrict the approximation to $[-1, 1]$. Moreover, this scaled even function can then be approximated only with cosine terms. For $f : \mathbb{R}^m \to \mathbb{R}$ a multivariate function of dimension $m$, the *n*th order Fourier Basis is then defined by:

$$\forall i \in \{0, \dots, (n+1)^m\}, \quad \psi_i(x) = \cos(\pi \mathbf{c}^i \cdot x) \tag{11}$$

where $\mathbf{c}^i = [c_1, \dots, c_m], c_j \in \{0, \dots, n\}$.

Each extraction function $\psi_i$ in the basis corresponds to a weight vector $\mathbf{c}^i$ that attributes an integer weight to every element of the input vector $x$. The design of these weight vectors can be the product of careful engineering [15]. However, a more systematic approach is to use every possible combination for a given order *n*, and let the learning algorithm find the appropriate weights. For any set of integers $[n] = \{0, \dots, n\}$, we write $[n]^m$ the set of all *n*-tuples - allowing for repetition - from $[n]$ to $\mathbb{N}^m$. We define a Fourier basis matrix $F_n^\psi \in \mathcal{M}_{m \times (n+1)^m}$ whose columns are made of integer weight vectors:

$$\begin{aligned} F_n^\psi &= [\mathbf{c}^1, \dots, \mathbf{c}^{(n+1)^m}] \\ \text{st.} \quad \cup_{i=1}^{(n+1)^m} \mathbf{c}^i &= [n]^m \end{aligned} \tag{12}$$

The basis vector used for the function approximation of the value function and policy can then be written as:

$$\begin{aligned} \bar{s} &= \frac{s - s^{lb}}{s^{ub} - s^{lb}} \\ \psi(s) &= \cos(\pi \bar{s} \cdot F^\psi) \end{aligned} \tag{13}$$

for $\forall s \in \mathcal{S}$, with $s^{lb}, s^{ub}$ respectively the lower and upper bound vectors on $\mathcal{S}$. We have $\theta \in \mathbb{R}^d, \vartheta \in \mathbb{R}^{d+1}$, with $d = (n+1)^m$. The number of possible features therefore grows exponentially in the size of the state-space, which would make this approach intractable for large farms if the size of the state space concurrently increased with the number of turbines. But our decentralized approach allows us to keep small local state spaces, each of which can then be represented by a vector encoding all possible Fourier features for a given order.

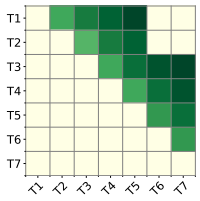
Using the basis defined by (13)-(12), we approximate the critic and the gaussian actor with the following parametrization: $\forall s_i \in \mathcal{S}_i$, and $\bar{s}_i \in [0, 1]^3$ the scaled state defined in (13).

$$\begin{aligned} V_{i,\theta}(s_i) &= \theta^T \psi(s_i) = \theta^T \cos(\pi F_n^\psi \cdot \bar{s}_i) \\ \pi_{i,\vartheta}(s_i) &= \mathcal{N}(\vartheta_\mu^T \psi(s_i), \sigma^2) \\ &= \mathcal{N}(\vartheta_\mu^T \cos(\pi F_n^\psi \cdot \bar{s}_i), \sigma^2) \end{aligned} \tag{14}$$

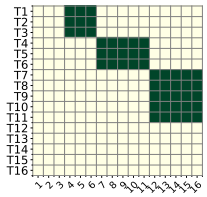
with $\theta \in \mathbb{R}^d, \vartheta = [\vartheta_\mu, \sigma], \vartheta_\mu \in \mathbb{R}^d$, $\sigma \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}$ parameters to learn. We update parameters $\theta$ and $\vartheta$ following the standard actor-critic updates (7), modified to account for the delayed rewards following the methodology introduced in III-B:

$$\begin{aligned} \theta_{k+1} &= \theta_k + l_{v,k} \delta_k \nabla_\theta V_{\theta_k}(s_{k-rd}) \\ \vartheta_{k+1} &= \vartheta_k + l_{q,k} \delta_k \nabla_\vartheta \ln \pi_{\vartheta_k}(a_{k-rd} | s_{k-rd}) \end{aligned} \tag{15}$$

with $rd$ defined in (9). Inserting (14) in (15), we derive the updates of our decentralized Fourier basis actor critic




(a) Layout 2 - 7 turbines




(b) Layout 3 - 16 turbines

Fig. 1: Representation of reward delay matrix for every agent on Layout 2 with 7 turbines (a), and Layout 3 with 16 turbines. Shorter delays are represented by brighter hues. For each row, the algorithm updates for an action are executed when the feedback from the agent corresponding to the darkest hue is ready.

algorithm, called DFAC in the sequel:

$$\begin{aligned}
\delta_{i,k} &= r_{i,k} + \beta V_{\theta_k}(s_{i,k-rd_i+1}) - V_{\theta_k}(s_{i,k-rd_i}) \\
\theta_{i,k+1} &= \theta_{i,k} + l_{v,k}\delta_{i,k}\psi(s_{i,k-rd_i}) \\
\vartheta_{\mu,i,k+1} &= \vartheta_{\mu,i,k} + \\
&\quad l_{q,k}\delta_{i,k}\frac{1}{\sigma^2}\psi(s_{i,k-rd_i})^T[a_{i,k-rd_i} - \vartheta_\mu^T\psi(s_{i,k-rd_i})] \\
\sigma_{i,k+1} &= \sigma_{i,k} + \\
&\quad l_{q,k}\frac{1}{\sigma}\left[\left(\frac{a_{i,k-rd_i} - \vartheta_\mu^T\psi(s_{i,k-rd_i})}{\sigma}\right)^2 - 1\right]
\end{aligned} \tag{16}$$

Algorithm DFAC (16) runs in parallel at every agent, and is able to adapt to any wind farm irrespective of size or layout without any change in structure. Its decentralized approach reduces the dimension of the search space for all agents, making it fit for real-time online optimization. We now turn towards evaluating the performance of DFAC on wind farm simulations under dynamic wake conditions.

## IV. Simulation Results

In this section, we run several experiments to validate our new approach. We first compare the performance of our new algorithm DFAC to the decentralized tabular Q-learning algorithm DADQ previously proposed in [4] on a small 3 turbines layout. We then evaluate both algorithms on various wind farm layouts of increasing size, assessing their ability to scale to large wind farms with up to 32 turbines.

### A. Simulation Setup

The choice of a wind farm simulator is a trade-off between computation complexity and fidelity of the predicted wind fields. Steady-state models like the ones used in the FLORIS simulator [20] estimate the time-averaged features of the wind flow while ignoring the dynamics of short-term effects. WFSim provides a higher fidelity simulation environment by accounting for time-varying non-linear wake dynamics, and has been validated against high-fidelity large eddy simulators [5]. All our subsequent experiments are done on WFSim.

We consider four farms of NREL offshore 5-MW turbines with a diameter $D = 126m$. On the first farm (Layout 1), 3 turbines are spread in a row in the center of the farm, separated by a distance of $4D$ meters. The second farm (Layout 2) has 7 turbines arranged along a diagonal, each downstream turbine slightly shifted with regard to its most upstream neighbor. It is a layout inspired from a real operating onshore wind farm located in France [21]. The two remaining layouts (Layout 3, 4) have 16 and 32 turbines respectively, spread on 5 columns. A snapshot of the 3 last simulations can be seen on the first row of Figure 3. Experiments are run under stationary wind conditions with a wind inflow of an average freestream velocity of $u_k^\infty = 12m/s$. The wind is directed orthogonally to the axis of the turbine rows, i.e $\phi_k = 0°$. The sampling period of the simulation is set to 20s and the space of available yaws $\mathcal{Y}$ is $[-40°, 40°]$.

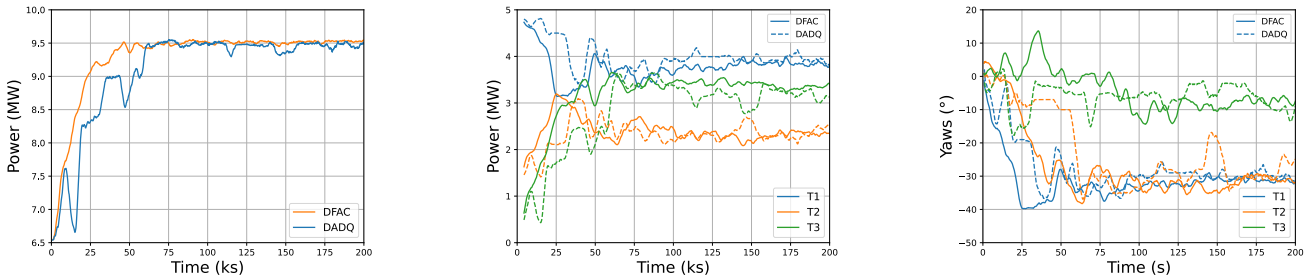
For our actor critic algorithm, we use a Fourier Basis of order $n = 8$, which corresponds to parameters $\theta$ and $\vartheta_\mu$ of dimension $d = 81$. In practice, accounting for an added bias term for the two linear regressions, as well as the policy's standard deviation variable learned separately, this raises the number of parameters to learn to $M \times 165$, with $M$ being the number of turbines. The parameters are randomly initialized following a uniform law on $[-0.1, 0.1]$.

For DADQ, we discretize the action space into 3 bins corresponding to 3 options (increase yaw, decrease yaw, stay still), and learn a Q-table of dimension $81 \times 3$ for every agent, a total of $M \times 243$ parameters. We use a constant Boltzmann exploration strategy during the training, and all values in the Q-table are initialized at $q_0 = 0.15$.

For all experiments, the discount factor is $\beta = 0.75$, and an averaging window of $\lambda = 2$min is used. To evaluate the performance of the algorithm without any prior knowledge, the yaws are initialized at $0°$, which corresponds to a naive greedy strategy where all turbines are made to face the wind. The frequency of the algorithm, that is the time between two actions are taken, is a multiple of the simulator sampling frequency $h$. Since the frequency of updates is in any case limited by the delay described in III-B, we are encouraged to take a slower frequency. We follow the heuristic of choosing the median of all unique values in the delay matrix $D$.

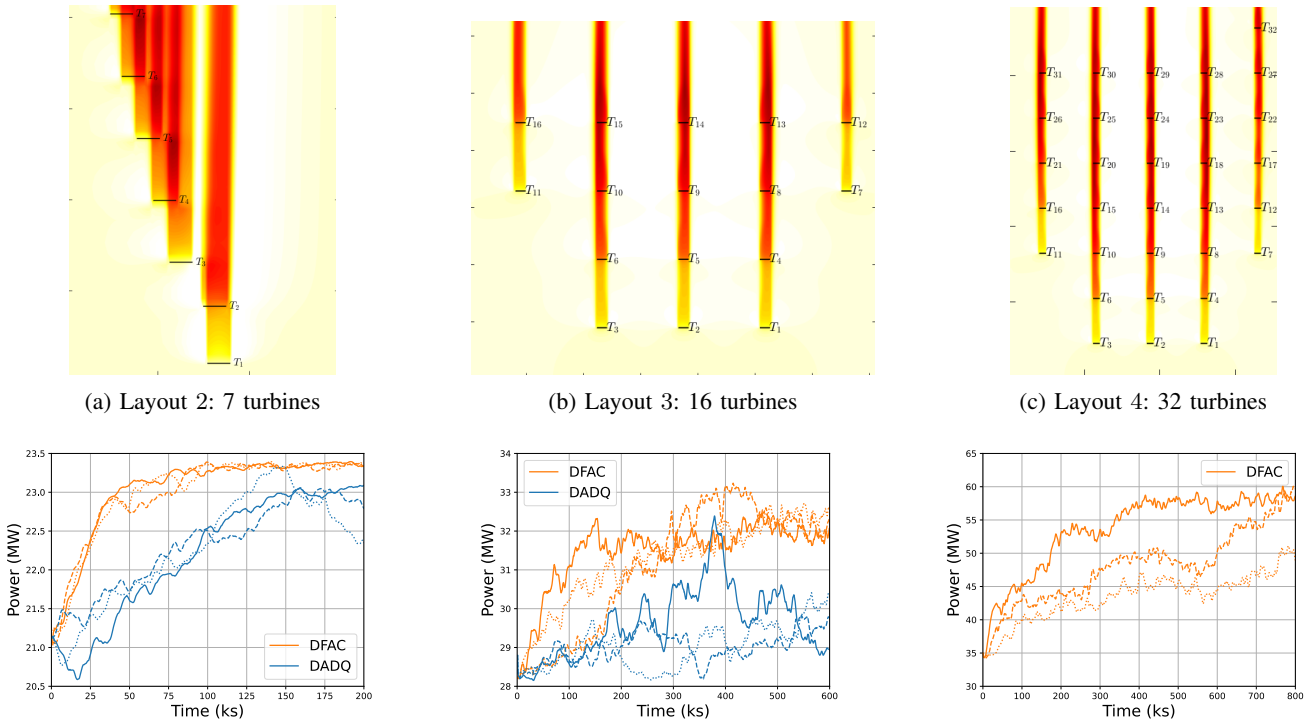
### B. Results

The comparative results of our 2 algorithms under stationary conditions for Layout 1 are reported on Figure 2, and a summary of both average energy produced and final performance at convergence can be found in table I. The two algorithms are run for 200ks, which corresponds to $10^4$ iterations. Compared to the greedy strategy baseline, both algorithms greatly increase the power output of the farm, by $44.76\%$ for the tabular Q-learning, and $45,8\%$ for Fourier actor-critic. The latter reaches convergence earlier, needing $50k$ seconds as opposed to $75k$ for the Q-learning. It is also more stable: it exhibits much less oscillatory behavior, both in terms of total power and turbine yaws. Both algorithms converge towards the same strategy: with




(a) Total farm power output (b) Individual turbine power production (c) Turbine yaws

Fig. 2: Comparing DFAC with a DADQ on a 3-turbine wind farm simulated on WFSim. Algorithms run without two-step warm start. Evolution of total power output [W], individual turbine power [W], and turbine yaws [°]. Power measures are averaged over 1 hour.




(a) Layout 2: 7 turbines (b) Layout 3: 16 turbines (c) Layout 4: 32 turbines

(d) Total farm power output (e) Total farm power output (f) Total farm power output

Fig. 3: Results of DFAC on 3 wind farms simulated on WFSim: 7 turbines (first column), 16 turbines (second column) and 32 turbines (third column) run for $200k$, $600k$ and $800k$s respectively for 3 different seeds. Evolution of total power output [W] compared to DADQ. Power measures are averaged over 1 hour.

the two first turbines upstream of the farm yawing towards $-30°$, decreasing their own power output to deflect the wake away from the downstream turbine.

### C. Scaling Experiments

The DFAC algorithm is run on 3 other farms with respectively 7, 16 and 32 turbines until convergence, for respectively $200k$, $600k$ and $800k$ simulated seconds. These experiments use various layouts, corresponding to more realistic wind farms scenarios and providing a challenging test of scaling capacity in decentralized conditions. Indeed, one way to interpret the delay matrix (8) is that every agent delays its DFAC updates only until its impact on a pre-

TABLE I: DFAC in the 3 turbines layout case under stationary conditions, compared to FLORIS routine and DADQ. Results after 200ks. Mean power output measured during the learning period, and final percentage increase compared to the $0°$ yaw initialization.

| **Method** | **Avg Energy [MWh]** | **Final Performance (%)** |
|---|---|---|
| FLORIS Routine | 9.0 | 37.5 |
| DADQ | 9.25 | 44.89 |
| DFAC | **9.34** | **45.83** |

TABLE II: Average energy produced during 1h on scaling experiments with Fourier Actor-Critic agents on layout with 3, 7, 16 and 32 turbines. Increase over baseline and over tabular Q-learning after 200ks (Layout 1, 2) and 600ks respectively.

| Layout | Energy [MWh] | Over baseline (%) | Over Q-learning (%) |
|---|---|---|---|
| Layout 1 (3T) | 9.34 | 45.83 | 2.20 |
| Layout 2 (7T) | 23.02 | 9.35 | 3.74 |
| Layout 3 (16T) | 31.44 | 10.99 | 6.86 |
| Layout 4 (32T) | 47.90 | 37.02 | - |

defined subset of other agents can be evaluated. On a farm with two turbines, this would be equivalent to all agents waiting until their wakes have traveled to the end of the farm. As the number of agents increases however, this set contains only a smaller fraction of all agents. This allows for more frequent updates to speed up convergence, but at the cost of making cooperation among agents much more difficult. Examples of matrices for two different farm sizes are illustrated on Figure 1. Larger wind farms therefore provide an interesting test of the ability of the delay-aware reward designed in III to induce cooperation among agents.

Because of the wider variance expected on more complex layouts, we run each experiment with 3 different randomization seeds for the initialization of the algorithm, and report all results. We run the same experiments on decentralized tabular Q-learning for comparison. The evolution of power output and turbine yaws during learning is available on Figure 3. Algorithms are compared on the one hour average energy metric, comparing the average power produced during one hour of the simulation, and averaged over the 3 seeds. This metric summarizes overall performance by accounting for rapidity of convergence, final performance upon convergence, and stability. The Fourier basis actor-critic largely improves over the Q-learning algorithm on Layout 2 and 3, and the magnitude of its improvement increases as the size of the wind farm grows. Based on these results, no further comparison was made on Layout 4. Table II sums up the increased percentages over both the baseline and the tabular Q-learning alternative with respect to average energy produced during one hour.

## V. Conclusions and future works

We proposed actor-critic agents for decentralized cooperative yaw control on wind farms. We showed that a linear function approximation approach with a Fourier basis, combined with a careful design of the reward function, is sufficient to obtain fast coordination of agents on dynamic simulations, and adapts to different layouts on wind farms up to 32 turbines. Our method is faster than previous decentralized reinforcement learning algorithms for wind farm control and uses less parameters. Experimental results point towards a subexponential convergence time in the number of turbines, and investigation of theoretical guarantees should be undertaken. Future work will moreover have to address adaptation to more realistic wind conditions, which might require exploring more complex architectures for function approximation.

## References


[1] P. Fleming, J. Annoni, J. J. Shah, L. Wang, S. Ananthan, Z. Zhang, K. Hutchings, P. Wang, W. Chen, and L. Chen, "Field test of wake steering at an offshore wind farm," *Wind Energy Science*, 2017.

[2] M. F. Howland, S. K. Lele, and J. O. Dabiri, "Wind farm power optimization through wake steering," *Proceedings of the National Academy of Sciences*, no. 29, pp. 14495–14500, 2019.

[3] A. C. Kheirabadi and R. Nagamune, "A quantitative review of wind farm control with the objective of wind farm power maximization," *Journal of Wind Engineering and Industrial Aerodynamics*, 2019.

[4] C. B. Monroc, E. Bouba, A. Bušić, D. Dubuc, and J. Zhu, "Delay-aware decentralized q-learning for wind farm control," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, IEEE, 2022.

[5] S. Boersma, B. Doekemeijer, M. Vali, J. Meyers, and J.-W. van Wingerden, "A control-oriented dynamic wind farm model: Wfsim," *Wind Energy Science*, no. 1, pp. 75–95, 2018.

[6] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems* (S. Solla, T. Leen, and K. Müller, eds.), MIT Press, 1999.

[7] I. Grondman, L. Buşoniu, G. A. D. Lopes, and R. Babuška, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, pp. 1291–1307, 2012.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, 20-22 Jun 2016.

[9] D. Lee, N. He, P. Kamalaruban, and V. Cevher, "Optimization for reinforcement learning: From a single agent to cooperative agents," *IEEE Signal Processing Magazine*, no. 3, pp. 123–135, 2020.

[10] L. Busoniu, R. Babuska, and B. De Schutter, *Multi-agent Reinforcement Learning: An Overview*, ch. 7, pp. 183–221. 2010.

[11] P. Stanfel, K. Johnson, C. J. Bay, and J. King, "Proof-of-concept of a reinforcement learning framework for wind farm energy capture maximization in time-varying wind," *Journal of Renewable and Sustainable Energy*, no. 4, 2021.

[12] E. Schuitema, L. Busoniu, R. Babuska, and P. Jonker, "Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2010.

[13] G. I. Taylor, "The spectrum of turbulence," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, no. 919, pp. 476–490, 1938.

[14] P. Stanfel, K. Johnson, C. J. Bay, and J. King, "A distributed reinforcement learning yaw control approach for wind farm energy capture maximization," in *2020 American Control Conference (ACC)*, pp. 4065–4070, 2020.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[16] Y. Li, "Deep reinforcement learning: An overview," *ArXiv*, vol. abs/1701.07274, 2017.

[17] R. Islam*, P. Henderson*, M. Gomrokchi, and D. Precup, "Reproducibility of benchmarked deep reinforcement learning tasks for continuous control," in *Reproducibility in Machine Learning Workshop (ICML)*, 2017.

[18] G. D. Konidaris, S. Osentoski, and P. S. Thomas, "Value function approximation in reinforcement learning using the fourier basis.," in *AAAI* (W. Burgard and D. Roth, eds.), AAAI Press, 2011.

[19] P. Sabes, "Approximating Q-values with basis function representations," in *Proceedings of the 1993 Connectionist Models Summer School* (M. Mozer, P. Smolensky, D. Touretzky, J. Elman, and A. Weigend, eds.), pp. 264–271, Lawrence Erlbaum, 1993.

[20] P. Gebraad, F. Teeuwisse, J. van Wingerden, P. Fleming, S. Ruben, J. Marden, and L. Pao, "Wind plant power optimization through yaw control using a parametric model for wake effects - a cfd simulation study," *Wind Energy*, no. 1, pp. 95 – 114, 2016.

[21] T. Duc, O. Coupiac, N. Girard, G. Giebel, and T. Göçmen, "Local turbulence parameterization improves the jensen wake model and its implementation for power optimization of an operating wind farm," *Wind Energy Science*, no. 2, pp. 287–302, 2019.