Delay-Aware Decentralized Q-learning for Wind Farm Control

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

Delay-Aware Decentralized Q-learning for Wind Farm Control

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

***Abstract*— Wind farms are subject to the so-called "wake effect", where upstream turbines facing the wind create sub-optimal wind conditions for turbines located downstream. One strategy to address this issue is to use yaw actuators to misalign the wind turbines with regard to the incoming wind direction, thus deflecting wakes away from downstream turbines. Tractable models for yaw optimization are however subject to inaccuracies, ignore wake dynamics and lack adaptability. This incentivizes the use of model-free methods. In this paper, we propose a delay-aware decentralized Q-learning algorithm for yaw control on wind farms. We introduce a strategy to handle delayed cost collection, and show that our method significantly increases power production in simulations with realistic wake dynamics. We validate our results for two farm layouts on mid-fidelity wind farm simulator FAST.Farm.**

I. Introduction

A common strategy to maximize the power production of a turbine is to orient its rotor to face the wind. The angle between the rotor and the wind direction, denoted "yaw" in this article, is then $0°$ (cf. Figure 1). In wind farms however, this greedy strategy is vulnerable 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. A number of controllable actuators can be used to reduce this effect: power capture can be influenced by controlling the orientation of the blades or the torque of the generator, the wake of a turbine can be deflected below downstream turbines by tilting the rotor plane, or to their side by modifying the yaw: a technique known as wake steering. In this work, we focus on this last strategy: our goal is to design an algorithm controlling the yaws of all turbines in order to maximize the total power output. Our solution, a delay-aware decentralized Q-learning algorithm, is robust to turbulent wind and wake dynamics, as validated on the simulator FAST.Farm [1].

Traditional control approaches build a model of the wind farm and optimize control inputs with respect to this approximation. Various models using analytical approximations or numerical computations have been proposed in [2]. Several steady-state wake deflection models are available in the popular wind farm simulation environment FLORIS [3], developed by the National Renewable Energy Laboratory (NREL). Such models however lack accuracy and ignore the dynamics of turbulent wind and wake propagation, leading to misestimation of the wake effects in wind farms [2]. A more precise option is the simulator FAST.Farm, also from the NREL, that uses a wake dynamics module to reach results similar to high-fidelity large eddy simulations [4]. Figure 2 provides an example of these realistic wind fields, and can be compared to a similar simulation done with FLORIS on Figure 1. This precision however comes at the price of higher computational cost, and these complex models are not fit for real time optimization.

This has motivated the use of reinforcement learning (RL) algorithms: following a data-driven approach, they directly infer the best actions solely by observing the system's responses to input changes. This online learning approach is particularly interesting because of the need to recover in the field from sub-optimal behaviors due to differences between simulation and reality.

RL methods have already been used to control yaw angles for automatic generation control on a wind farm: [5] uses the centralized deep learning algorithm Deep Deterministic Policy Gradient (DDPG) with the axial induction factor as the control variable. In [6], DDPG is combined with offline supervised learning, leading to a significant increase of power output in high-fidelity simulations. Centralized methods however see the dimension of their problem grow with the number of turbines, raising the issue of scalibility. In [6], the authors exploit a layout-specific symmetry to reduce the dimension of the problem, but this does not transfer to other farm layouts.

To address this issue, decentralized learning schemes are a promising way towards more scalable farm control algorithms. They have been shown to lead to significant increase in total power production on wind farms, but only on low-fidelity simulators. [7] uses a decentralized Deep Q-learning approach in a wind farm with 3 turbines, but only tests the algorithm in a simulation experiment with a simple analytical model ignoring temporal effects. [8] splits the farm in groups of turbines to create several smaller minimization problems, and designs a distributed algorithm based on a linear policy search combined with the alternating direction method of multipliers. The method is evaluated with a 6-turbines layout on FLORIS. Similarly, in [9] and [10], the authors provide a proof-of-concept for a decentralized tabular Q-learning algorithm. It is implemented on a 3-turbines layout in a quasi-dynamic version of FLORIS, but this still ignores the temporal dynamics of turbulent wind and wake propagation.

The validation of a decentralized approach in simulations with realistic dynamics of wake propagations is therefore

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

E. Bouba, D. Dubuc and J. Zhu are with 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

still an open problem. This is especially critical because decentralized algorithms limit the observability of the problem for each turbine, rendering their environment non-stationary. Meanwhile, realistic wake dynamics prevent the observation of a controller's impact until the wake has propagated downstream, and make the assignment of variation to any controller harder because of wake merging.

Few strategies have been used to address wake propagation time in wind farm optimization problems. In [9], the farm is split in subsets within which only one turbine is allowed to operate at a time, but this locking strategy excessively slows down convergence, and is only tested in a semi-dynamic environment. We rather propose to overcome that difficulty with a delayed reward RL approach.

Our contributions are the following. We show that the wake steering problem can be framed as a Delayed Reward RL problem. This allows us to draw from that literature to design a delay-aware Q-learning algorithm that is decentralized, agnostic to farm layouts, and does not rely on turbine locking. We evaluate this algorithm on 2 different layouts on the mid-fidelity simulation environment FAST.Farm. We are, to the best of our knowledge, the first to validate a decentralized reinforcement learning algorithm for wind farm control in a dynamic simulation. Our experiments show that our method is scalable and robust to simulations with turbulent wind and wake dynamics.

The remainder of this paper is organized as follows. Section II provides background information on reinforcement learning. Section III discusses issues raised by the application of decentralized Q-learning to wind farm control, and introduces a new algorithm based on a Delayed Reward problem formalization. In section IV, we show that our algorithm successfully increases the total power output for two farm layouts in FAST.Farm, and can be combined with steady-state models in a two-step warm start approach. Finally, 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 $r : S \times A \to R$ 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:

$$\max_{\pi} E[G] = \max_{\pi} E\left[\sum_{k=0}^{\infty} \beta^k r(s_k, a_k)\right]$$

with $0 < \beta < 1$ the discount factor, s_0 the initial state, and $\{s_k, a_k\}_{k=0\ldots\infty}$ the trajectory of the agent in the environment under policy π.

For a policy π, we define the state-action value function, or q function Q, as:

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

with $a_k \sim \pi(s_k)$. For any state-action pair (s, a), $Q_\pi(s, a)$ is therefore the expected value of choosing action a in state s, and then following policy π. We define an optimal q function Q^* such that:

$$\forall (s, a), Q^*(s, a) = \max_{\pi} Q_\pi(s, a)$$

A. Q-learning

To search for the best policy in a given environment, one can directly attempt to learn Q^*. The optimal q function satisfies the following fixed point Bellman optimality equation:

$$Q^*(s, a) = E\left[r(s, a) + \beta \max_{\bar{a}} Q^*(s', \bar{a}) \right] \tag{1}$$

with $s' \sim P_{s,a}$. Watkins' Q-learning algorithm [11] keeps estimates of the q values for each (s, a) pair - an approach known as tabular -, and iteratively updates an estimation $\hat{Q}$ of the optimal q function Q at each timestep k:

$$\hat{Q}_{k+1}(s_k, a_k) = \hat{Q}_k(s_k, a_k) + l_k \cdot \mathtt{TD}_k$$

where $\mathtt{TD}_k$ is the Bellman error estimator defined as:

$$\mathtt{TD}_k = r_k + \beta \max_{\bar{a} \in A(s_{k+1})} \hat{Q}_k(s_{k+1}, \bar{a}) - \hat{Q}_k(s_k, a_k)$$

with $\beta \in (0, 1)$ the discount factor and l_k a learning rate at step k. It is proven in [11] that $\hat{Q}$ converges with probability one to Q^* under mild conditions.

Q-learning is an off-policy algorithm: the learning of the optimal q values is decoupled from the policy followed by the agent, and the latter should ensure sufficient exploration of all action-state pairs to guarantee convergence. Many exploring schemes have been introduced in the literature, we chose the Boltzmann strategy which has been found to outperform other strategies for Q-learning in several toy examples [12]. Instead of following a greedy policy which picks the action with the largest q value estimate in each state (i.e. $a' = \arg\max_a \hat{Q}(s, a)$), one samples an action from the Boltzmann distribution:

$$a' \sim b(a|s), \quad \forall a \in A, b(a|s) = \frac{e^{\frac{\hat{Q}(s,a)}{\tau}}}{\sum e^{\frac{\hat{Q}(s,a)}{\tau}}} \tag{2}$$

with the parameter $\tau > 0$ controlling the probability of taking a non-greedy action.

B. Zap Acceleration

It is well known that Watkins' Q-learning converges slowly [13] [14]. Following a stochastic approximation approach, Zap Q-learning [13] proposes a matrix-gain improvement to accelerate the convergence of the algorithm. It is shown to have the optimal rate of convergence in the sense of the minimization of the asymptotic covariance [13].

The algorithm relies on the following updates, presented here for the tabular case:

$$\begin{aligned}\theta_{k+1} &= \theta_k - l_k \hat{A}_{k+1}^{-1} \mathtt{TD}_k \psi(s_k, a_k) \\ \hat{A}_{k+1} &= \hat{A}_k + \zeta_k [A_{k+1} - \hat{A}_k] \\ (A_{k+1})_{l,j} &= \begin{cases} -1 & \text{if } l = j \text{ and } \psi(s_k, a_k)_l = 1 \\ \beta & \text{if } \psi(s_k, a_k)_l = 1 \\ & \quad \text{and } \psi(s_{k+1}, \pi(s_{k+1}))_j = 1 \\ 0 & \text{else} \end{cases} \end{aligned} \tag{3}$$

with

$$\pi(s) \in \arg\max_a Q^{\theta_k}(s, a), \quad \forall s \in S,$$

ζ_k a sequence to chose such that $\lim_{n\to\infty} \frac{l_k}{\zeta_k} = 0$, $d = |S||A|$, $\theta \in R^d$ the vector storing q-function estimates for all state action pairs $\hat{Q} = Q^\theta$, and $\psi(s_k, a_k) \in \{0, 1\}^d$ an indicator vector for (s_k, a_k). To avoid stability issues, we can compute $\hat{A}_k^{-1}$ using the Moore–Penrose inverse as suggested in [15].

C. Delayed Reward Problem

When there is a known delay between the moment an action is sent to the environment and the moment the associated reward is collected by the agent, the environment is said to have a delayed reward. Following the method proposed in [16], this problem can be formalized as a deterministic delayed MDP (DDMDP) $\{S, A, P, r, c, o, a\}$, for an observation delay o, an action delay a , and a reward delay c. We moreover define the timestep delays (c_d, o_d, a_d), such that with h the sampling period, $c_d = \frac{c}{h}, o_d = \frac{o}{h}, a_d = \frac{a}{h}$. $\lceil c_d \rceil$ is then the number of timesteps before the reward becomes available. Acting in a DDMDP is in fact equivalent to acting in a standard MDP with a modified reward function, but at the cost of augmenting the state space with all actions taken during the delay [16]. This increase of the state space to inject a memory of passed events would lead to a particularly strong increase of the problem's dimension in our tabular case. In [17], the authors rather propose a modification of Q-learning called $dQ(0)$ for the case of the action delay: at timestep k, instead of updating $\hat{Q}(s_k, a_k)$, updates are performed for the action $\hat{a}_k$ that took effect at timestep k, but had been selected at timestep $k - \lceil a_d \rceil$. By restoring the temporal match between states and actions, $dQ(0)$ ensures convergence to the optimal q function of the equivalent undelayed MDP.

D. Multi-Agent RL for distributed optimization

When several agents seek to learn optimal policies to maximize a shared reward, the task is one of cooperative multi-Agent RL (MARL). The agents can equally be thought of as solving a distributed optimization problem [18]. This is a challenging task: because all agents concurrently interact with the environment, the stationarity assumptions supporting convergence guarantees do no longer hold. Several algorithms have been designed to inject awareness of other agents, or force coordination [19]. They however increase the complexity of the problem and are harder to implement. A naive approach can be to simply apply the single-agent Q-learning algorithm to the multi-agent case: much simpler, this decentralized approach has also been empirically shown to converge to a good equilibrium in certain settings [19], and a good proof-of-concept has been provided by [10] for the wind farm optimization case.

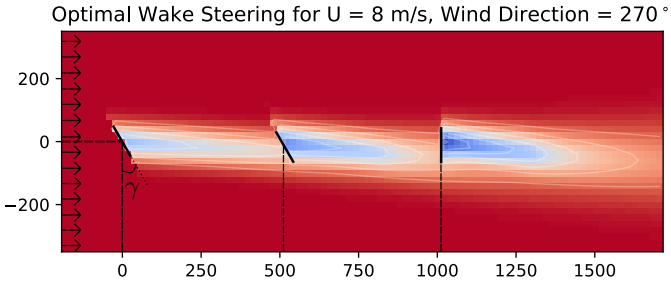


Fig. 1: Simulation of our 3-turbines layout on low-fidelity simulator FLORIS. The wind is blowing from the left. The wind velocity deficit is visible behind every turbine, but the time-averaged predictions remain smooth. The first turbine's yaw γ is reported. The last turbine is still facing the wind, its yaw is of $0°$.

III. Application to Wind Farm Control

A. Multi-agent optimization problem

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, the freestream wind conditions $\mathbf{w_k} = (u_k^\infty, \phi_k)$ are observed at the entrance of the farm, with u_k^∞ the speed and ϕ_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 $P_{i,k}$. For any given k we want to maximize the total power output of the wind farm $P_k = \sum_{i=1}^M P_{i,k}$. 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. Under the cooperative multi-agent RL framework mentioned in II-D, this can be done in a decentralized fashion with M RL agents independently learning local optimal policies to maximize a shared reward. With each agent strictly observing his local state and action, we consider M state spaces $S_{i,1\leq i\leq M}$, $S_i = \mathcal{Y} \times R^2$, such that at each timestep k we have M local states:

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

If wind conditions are supposed stationary, this can be restricted to $s_{i,k} = [\gamma_{i,k}]$.

The local action spaces are defined as $\forall i, A_i = A = \{-1, 0, 1\}$. Each chosen action $a_i \in A$ leads to an update

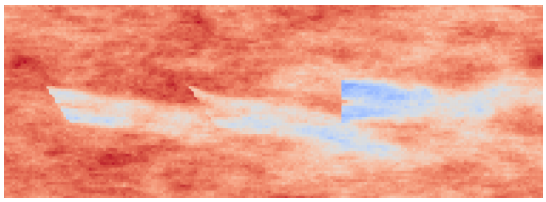


Fig. 2: Simulation of the 3-turbines layout on mid-fidelity simulator FAST.Farm with TurbSim generated turbulent wind at 8 m/s. Compared to FLORIS, FAST.Farm simulates turbulence and wake dynamics, including wake meandering and merging.

in the corresponding yaw defined by $a \times (\Delta\gamma)$, with $\Delta\gamma$ a predefined step size matching the real constraints and limitations on turbine yaw actuation. As mentioned in II-D, we can then learn M single-agent Q-learning algorithms simultaneously.

This approach has been shown to lead to an increase of 8 % in the total power output for a farm of 3 turbines simulated with simplified wake models on the FLORIS environment [10], but it fails as soon as the farm's dynamics are considered, even partially [9]. Indeed, value updates executed on successive time steps $(k, k+1)$ become decoupled from the assessment of the real impact of a yawing action a_k: positive effects of the induced wake deflection are delayed in time and ignored by the immediate update, while only the immediate decrease in power production at the turbine can be assigned to the action. A turbine locking scheme has been proposed in [9], but this is an unnecessary slowdown of the algorithm: the problem of delayed reward assignation is recurrent in many control problems, and we can draw from this literature to find a faster solution.

B. Delayed Reward Q-learning

We adapt the method in [17] introduced in II-C to the case of the deterministic reward delay c, which similarly leads to a temporal mismatch between state-action pairs and rewards. Like in [17] we can restore that matching by waiting for c_d time steps before updating $\hat{Q}$. Note that this does not suffice to guarantee convergence, because the non-stationarity induced by the multi-agent configuration remains.

This approach requires the definition of a reward delay for each agent c, which should correspond to the time it takes for the impact of an action to be fully observed on downstream turbines. This time should differ according to the location of the turbine in the layout, and the decentralized approach allows us to define different reward delays for all agents.

Inspired by [9], we approximate this time delay from upstream turbine i to downstream turbine j with Taylor's frozen wake hypothesis [20]

$$d_{i,j} = \frac{c_{i,j}}{u^\infty}$$

with $c_{i,j}$ the downwind distance between the 2 turbines and u^∞ the freestream velocity at the entrance of the farm. Wake effects should be negligible above a certain distance, so a cutoff distance d_{cutoff} is introduced. Finally, we add a safeguard by using a multiplier $m > 1$. This gives us a wake delay matrix $D_{i,j}$ for all $0 \leq i, j \leq M$:

$$D_{i,j} = \begin{cases} m d_{i,j} & \text{if } j \text{ downstream and } c_{i,j} \leq d_{\text{cutoff}} \\ 0 & \text{otherwise} \end{cases}$$

It is then natural to define the timestep reward delay c_i for each turbine i as the number of time steps of the largest wake delay, ie. $c_i = \lceil \max_j \frac{D_{i,j}}{h} \rceil$. Our Q-learning update is therefore for all $i \in 1 \dots M$:

$$\begin{aligned}
\theta_{k+1} &= \theta_k - l_k \hat{A}_{k+1}^{-1} \cdot \texttt{TD}_{dadQ,k} \cdot \psi(\kappa_{k-c_i}) \\
\hat{A}_{k+1} &= \hat{A}_k + \zeta_k [A_{k+1} - \hat{A}_k] \\
\pi(s) &\in \arg\max_a Q^{\theta_k}(s,a), \quad \forall s \in S_i \\
\texttt{TD}_{dadQ,k} &= \Big(r_{i,k} + \beta \max_{\bar{a} \in A(s_{k-c_i+1})} \hat{Q}_k(s_{k-c_i+1}, \bar{a}) \\
&\qquad - \hat{Q}_k(\kappa_{k-c_i}) \Big) \\
(A_{k+1})_{l,j} &= \begin{cases} -1 & \text{if } l = j \text{ and } \psi(\kappa_{k-c_i})_l = 1 \\ \beta & \text{if } \psi(\kappa_{k-c_i})_l = 1 \\ & \quad \text{and } \psi(s_{k-c_i+1}, \pi(s_{k-c_i+1}))_j = 1 \\ 0 & \text{else} \end{cases}
\end{aligned} \tag{4}$$

with $\kappa_k = (s_k, a_k)$, $a_k \sim b(a|s_k)$ following (2), $\hat{A}_0 = I$ and the Zap acceleration activated only if $\zeta_k > 0$. Following [14], we use a polynomial decrease for our learning rate: $l_k \propto \frac{1}{k^p}$, $0 < p < 1$.

One issue with decentralized cooperative multi-agent learning tasks is the assignment of the observed reward to the right agent. This is amplified by the reward delay: when any change of total production is observed by an agent after a delay, several factors compete as a probable explanation: the delayed impact of the agent's own action, the immediate consequence of other agents changing their yaws, but also the combined delayed effects of previous wake deflections by all agents. We reduce this uncertainty by measuring the power generation of an agent's downstream turbines at the moment of the estimated impact. Every agent therefore receives a slightly different reward function. Moreover, to account for the variance in instantaneous power due to turbulent wind, we average the measures of power output at every turbine: we define $\lambda \geq 1$, the size of the averaging window. Finally, to reduce the influence of the nominal power output on the evaluation of yawing strategies, increases in percentage rather than raw values are considered. Inspired by [9], we use the sign of the measured variation as the reward signal, and apply a threshold to filter out noise. We define for $i = 1, \cdots, M$ the reward function:

$$r_{i,k} = \begin{cases} 1 & \text{if} \quad \frac{V_2^i - V_1^i}{V_1^i} > \delta \\ -1 & \text{if} \quad \frac{V_2^i - V_1^i}{V_1^i} \leq -\delta \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

with $\delta > 0$, $V_1^i = \sum_{j=1}^M P_{j,k}$ and $V_2^i = \sum_{j=1}^M P_{j,k+D_{i,j}}$.

Algorithm 1 Delay-Aware Decentralized Q-learning

Require: Agent ID i, Delay Matrix $D_{i,j,1\leq i,j\leq M}$, Sampling period h, RL parameters $\beta \in (0,1), \tau > 0, T \geq 0, w, \delta, \lambda$
Init γ_0, *stack* S, *table* $\hat{Q}$, *List* P
$c \leftarrow \max_j \frac{D_{i,j}}{h}$
for t = 1 ... T **do**
  if $t \equiv 0 \pmod{w}$ **then**
    S.push(t)
    $s_t, p_t \leftarrow$ observe()
    $p_{\lambda,t} \leftarrow compute\_average(\lambda)$
    $a_t \leftarrow$ select_action($s_t, \hat{Q}, \tau$)
    take_action(a_t)
    P.push($p_{\lambda,t}$)
  end if
  if S not empty AND $t \geq$ S.top() $+c$ **then**
    k $\leftarrow$ S.pop()
    $r_k \leftarrow$ get_reward(i, k, D, P, δ) (following (5))
    update($\hat{Q}$) (following (4))
  end if
end for

The pseudo-code of our final algorithm can be found in Algorithm 1. It runs simultaneously at every agent.

Information about the region of optimal yaws can be injected in the algorithm by modifying two initial values: the initial yaw vector γ_0, and the initial Q-table values $\hat{Q}(s,a)$. In a two-step warm start approach, such information can be obtained from time efficient simulations in environments models like FLORIS, and allow for significant increase of the total energy production over the period. We expand on this approach in Section IV-C.

IV. Simulation Results

We first seek to validate our approach: in the absence of any turbine locking, is our delay aware Q-learning robust to delayed wake propagation with turbulent wind? To answer that question, we first test Algorithm 1 without acceleration or any prior knowledge of optimal yaws in an environment with simulated wake dynamics. Then, we evaluate convergence acceleration with Zap updates, and a two-step warm start approach with yaw and Q-table initialization.

A. Simulation Setup

The choice of the 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 [3] estimate the time-averaged features of the wind flow while ignoring the dynamics of short-term effects. FAST.Farm provides a higher fidelity simulation environment [1]: it takes into account the wake deficits, advection, deflection, meandering, and merging, and its results are validated against high-fidelity large-eddy simulations. All our subsequent experiments are done on FAST.Farm [4].

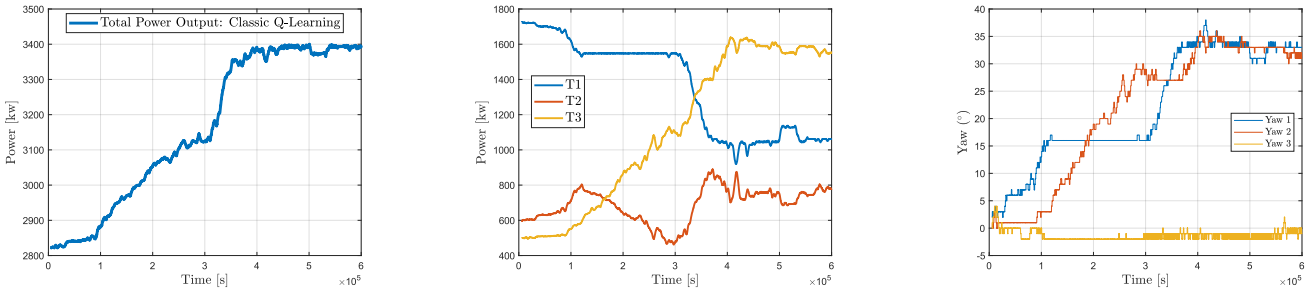
We consider two farms of NREL offshore 5-MW turbines with a diameter $D = 126m$. On the first farm (Layout 1), 3 turbines are spread in a row in the center of the farm, separated by a distance of $4D$ meters. The second farm (Layout 2) has 6 turbines, spread in 2×3 layout. The distance between the 2 rows in of $4D$ meters. On this layout, because the two rows are mirror images of each other, spread around an axis that coincides with the reference point of the simulated inflow wind, we expect symmetrically opposed turbines to reach similar yaws: this is a primary test of the consistency of our algorithm when the number of turbines increases. A snapshot of the first simulation can be seen on Figure 2

In order to replicate realistic wind conditions, we use the turbulent-wind simulator TurbSim [21] 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. Since the wind conditions are stationary, we can restrict our local state spaces S_i to the admissible yaw space $\mathcal{Y}$. It is adapted to the tabular case with a discretization of the interval $[-40, 40]$ in bins of $1°$. We therefore learn 3 (resp. 6) Q-tables of dimension 81×3 for the first (resp. second) layout. We use a constant Boltzmann exploration parameter of $\tau = 0.1$, a discount factor $\beta = 0.75$, and an averaging window of $\lambda = 10$min. To evaluate the performance of the algorithm without any prior knowledge, all values in the Q-table are initialized at $q_0 = 0.15$. The yaws are initialized at $0°$, which corresponds to a naive greedy strategy where all turbines are made to face the wind.

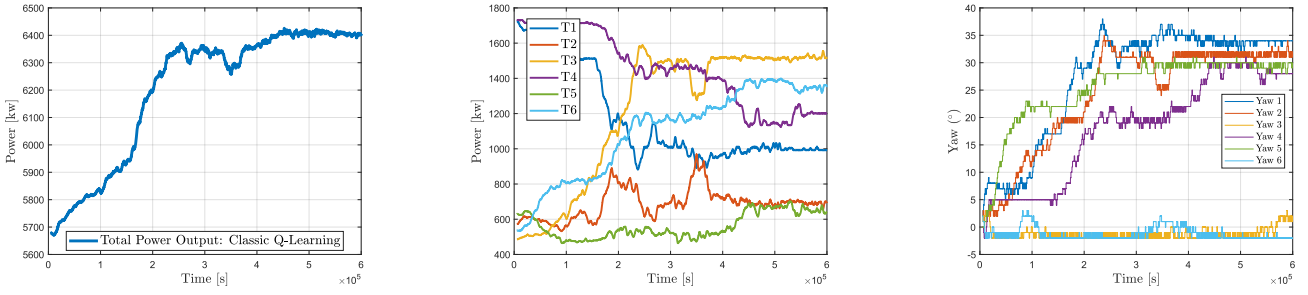
B. Results

We first train Algorithm 1 in FAST.Farm for 600ks, corresponding to 2.3×10^5 iterations. The complete simulation in FAST.Farm takes 72 hours for Layout 1, and 120 hours on Layout 2, on a single 2.6GHz Intel Xeon Gold CPU with 36 cores. The results are depicted on Figure 3. Compared to the greedy strategy baseline, our algorithm increases the total power output of the farm of 20% for Layout 1 and of 14% for Layout 2. In both cases, the algorithm reaches convergence around $450ks$, corresponding to 150k iterations. Despite doubling the size of the farm, we are therefore able to sensibly keep the same time of convergence, demonstrating a key advantage over both centralized or locking approaches. Figures 3c and 3b show the efficient yaw misalignment strategy learned by the algorithm on Layout 1, with the first turbine upstream of the farm yawing towards $33°$, significantly decreasing its own power output to deflect the wake away from downstream turbines. The most downstream turbine 3 correctly learns that yawing leads to no significant increase, and keeps its rotor axis aligned with the inflow wind at $0°$. Results for Layout 2 are displayed on Figures 3f and 3e. As expected, pairs of turbines with the same y-axis coordinates converge towards similar values. These preliminary results validate our decentralized, delay-aware Q-learning approach even with realistic turbulent wind and accurate wake dynamics.

We now turn to accelerate the learning speed of the algorithm. We repeat the experiments, changing the updates

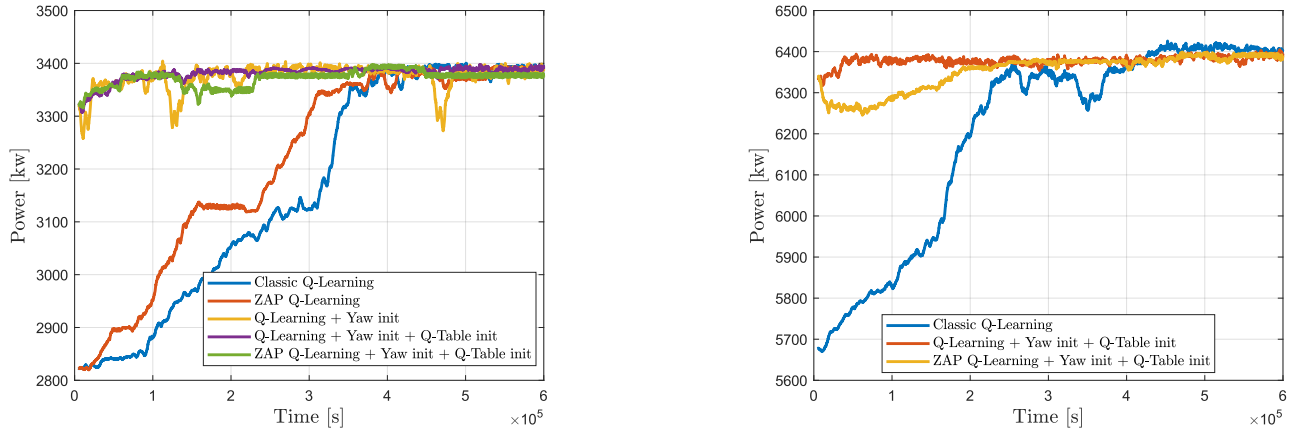


(a) Total farm power output (b) Individual turbine power production (c) Turbine yaws



(d) Total farm power output (e) Individual turbine power production (f) Turbine yaws

Fig. 3: Results of Algorithm 1 without two-step warm start on 2 wind farms simulated on FAST.Farm: 3 turbines (first row) and 6 turbines in 2×3 layout (second row). Evolution of total power output [kW], individual turbine power [kW], and turbine yaws [°]. Power measures are averaged over 1 hour.



(a) Layout 1: 3 turbines (b) Layout 2: 6 turbines

Fig. 4: Performance of Algorithm 1 for different initialization schemes: evolution of total farm power output averaged over 1 hour on the 3-turbines (a) and 6-turbines (b) wind farm simulated on FAST.Farm.

in Algorithm 1 to use Zap acceleration, by setting $\zeta_k = (l_k)^{0.85}$ for all timesteps. A comparison of total farm power outputs with and without acceleration is provided in Figure 4a for Layout 1. The Zap Q-learning experiment indeed converges faster toward the final 20% increase, with the total production increasing of 43kW per hour on average. We use Zap Q-learning in our subsequent experiments.

C. Two-Step Warm Start Approach

The time needed for Algorithm 1 to converge creates a significant loss of production. We could increase the total power output over the period by exploiting known information about the region of optimal yaws in the search space. A valuable clue can be given by industry practices, that use low-fidelity steady-state models to tune the yaws in a simplified environment. As reported in Table I, that simple heuristic performs relatively well: an increase of 17% over the baseline is observed when yaws obtained in FLORIS are subsequently evaluated on FAST.Farm. Unsurprisingly, modeling uncertainties however clearly prevent this method from finding the optimal yaws. In a two-step warm start approach, we can use Algorithm 1 to fine-tune initial results.

TABLE I: Algorithm 1 with and without initialization in the 3 turbines layout. Results after 600ks. Mean power output measured during the learning period, and final percentage increase compared to the 0° yaw initialization.

Method	Avg Measure [kW]	Final Performance (%)
FLORIS Routine	3312.22	17.3
Classic Q learning	3164.30	**20.29**
+ Yaw init	3374.11	20.04
+ Yaw init + Q-Table init	**3381.25**	20.18
Zap	3213.77	20.17
+ Yaw init + Q-Table init	3371.63	19.59

We evaluate two different initialization techniques based on this approach:

- *Yaw initialization in low-fidelity environments*. Initialize the yaws with FLORIS's optimization routine.
- *Yaw and Q-table initialization in low-fidelity environments* Train a simple decentralized Q-learning algorithm in the steady-state simulation environment before pursuing the learning on the dynamic environment.

A comparison of total power output evolution under the various initialization methods is displayed on Figure 4a, along with the classic Algorithm 1 with uniform initialization described in IV-B. Results for Layout 2 are displayed on Figure 4b.

All initialization methods compensate for the loss of production of the classic algorithm's learning phase: initialized values are quickly adapted to dynamic conditions and all experiments have a production within 1% of the classic algorithm's maximal power output after 220ks. However, initializing the yaws in the neighborhood of that local maximum make them less robust to the persistent exploration of the policy, and the learned yaws are less stable under the Two-Step Warm Start methods. With that regard, Zap updates seem to lead to more stable policies, especially on Layout 1, which is consistent with its theoretical role as variance minimizer. We report for all methods the increase over the baseline for the last measure of total power output, which represents the performance of the algorithm at convergence. To account for different strengths and weaknesses of all approaches in terms of rapidity of convergence, stability, and final performance, we also compute the average value of all measures during the learning period. All measures are one hour averaged. These results are reported in Table I.

V. CONCLUSIONS AND FUTURE WORKS

We introduced a new reinforcement learning algorithm for wake steering by yaw control in wind farms. We showed that decentralized single agent Q-learning algorithms can be made robust to realistic wake dynamics when augmented with a simple delayed update component. The similar convergence times observed for 3 and 6 turbines are a promising indication towards subexponential complexity in the number of turbines, and further experiments on larger farms should be pursued to validate that intuition. Moreover, because it relies on one measure of freestream wind conditions, this method can in principle also be extended to changing wind conditions while maintaining deterministic delays, and future work could address this adaptation to non-stationary wind conditions.

REFERENCES

[1] J. M. Jonkman, J. Annoni, G. Hayman, B. Jonkman, and A. Purkayastha, *Development of FAST.Farm: A New Multi-Physics Engineering Tool for Wind-Farm Design and Analysis*.

[2] C. L. Archer, A. Vasel-Be-Hagh, C. Yan, S. Wu, Y. Pan, J. F. Brodie, and A. E. Maguire, "Review and evaluation of wake loss models for wind energy applications," *Applied Energy*, vol. 226, pp. 1187–1207, sep 2018.

[3] 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*, vol. 19, no. 1, pp. 95 – 114, 2016.

[4] J. Jonkman, P. Doubrawa, N. Hamilton, J. Annoni, and P. Fleming, "Validation of FAST.farm against large-eddy simulations," *Journal of Physics: Conference Series*, vol. 1037, p. 062005, jun 2018.

[5] S. Vijayshankar, P. Stanfel, J. King, E. Spyrou, and K. Johnson, "Deep reinforcement learning for automatic generation control of wind farms," in *2021 American Control Conference (ACC)*, pp. 1796–1802, 2021.

[6] H. Dong, J. Zhang, and X. Zhao, "Intelligent wind farm control via deep reinforcement learning and high-fidelity simulations," *Applied Energy*, vol. 292, no. C, 2021.

[7] Z. Xu, H. Geng, B. Chu, M. Qian, and N. Tan, "Model-free optimization scheme for efficiency improvement of wind farm using decentralized reinforcement learning," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 12103–12108, 2020. 21st IFAC World Congress.

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

[9] 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.

[10] 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*, vol. 13, 8 2021.

[11] C. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 05 1992.

[12] A. D. Tijsma, M. M. Drugan, and M. A. Wiering, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, 2016.

[13] A. M. Devraj and S. Meyn, "Zap q-learning," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[14] E. Even-dar and Y. Mansour, "Learning rates for q-learning," in *Journal of Machine Learning Research*, pp. 1–25, 2001.

[15] A. Devraj, A. Bušić, and S. Meyn, "Zap Q-Learning - A User's Guide," in *ICC 2019 - Fifth Indian Control Conference*, Proceedings of the Fifth Indian Control Conference (ICC 2019), (New Delhi, India), pp. 10–15, IEEE, Jan. 2019.

[16] K. Katsikopoulos and S. Engelbrecht, "Markov decision processes with delays and asynchronous cost collection," *IEEE Transactions on Automatic Control*, vol. 48, no. 4, pp. 568–574, 2003.

[17] E. Schuitema, L. Busoniu, R. Babuska, and P. Jonker, "Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach," pp. 3226 – 3231, 11 2010.

[18] D. Lee, N. He, P. Kamalaruban, and V. Cevher, "Optimization for reinforcement learning: From single agent to cooperative agents," *CoRR*, vol. abs/1912.00498, 2019.

[19] L. Busoniu, R. Babuska, and B. De Schutter, *Multi-agent Reinforcement Learning: An Overview*, vol. 310, pp. 183–221. 07 2010.

[20] G. I. Taylor, "The spectrum of turbulence," *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, vol. 164, no. 919, pp. 476–490, 1938.

[21] B. Jonkman and B. Jr, "Turbsim user's guide," 01 2007.