A Distributed Reinforcement Learning Yaw Control Approach for Wind Farm Energy Capture Maximization

Preprint

Paul Stanfel,[1] Kathryn Johnson,[1] Christopher J. Bay,[2] and Jennifer King[2]

1 Colorado School of Mines
2 National Renewable Energy Laboratory

Presented at the American Control Conference (ACC)
July 1–3, 2020

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated by the Alliance for Sustainable Energy, LLC

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Contract No. DE-AC36-08GO28308

Conference Paper
NREL/CP-5000-75889
August 2020

A Distributed Reinforcement Learning Yaw Control Approach for Wind Farm Energy Capture Maximization

Preprint

Paul Stanfel,[1] Kathryn Johnson,[1] Christopher J. Bay,[2] and Jennifer King[2]

1 Colorado School of Mines
2 National Renewable Energy Laboratory

Suggested Citation
Stanfel, Paul, Kathryn Johnson, Christopher J. Bay, and Jennifer King. 2020. *A Distributed Reinforcement Learning Yaw Control Approach for Wind Farm Energy Capture Maximation: Preprint*. Golden, CO: National Renewable Energy Laboratory. NREL/CP-5000-75889. https://www.nrel.gov/docs/fy20osti/75889.pdf.

Conference Paper
NREL/CP-5000-75889
August 2020

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

NOTICE

This work was authored [in part] by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by the U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Wind Energy Technologies Office. The views expressed herein do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

Cover Photos by Dennis Schroeder: (clockwise, left to right) NREL 51934, NREL 45897, NREL 42160, NREL 45891, NREL 48097, NREL 46526.

NREL prints on paper that contains recycled content.

A Distributed Reinforcement Learning Yaw Control Approach for Wind Farm Energy Capture Maximization*

Paul Stanfel[1], Kathryn Johnson[1], Christopher J. Bay[2], and Jennifer King[2]

***Abstract*— In this paper, we present a reinforcement-learning-based distributed approach to wind farm energy capture maximization using yaw-based wake steering. In order to maximize the power output of a wind farm, individual turbines can use yaw misalignment to deflect their wakes away from downstream turbines. Although using model-based methods to achieve yaw misalignment is one option, a model-free method might be better suited to incorporate changing conditions and uncertainty. We propose an algorithm that adapts concepts of temporal difference reinforcement learning distributed to a multiagent environment that empowers individual turbines to optimize overall wind farm output and react to unforeseen disturbances.**

I. INTRODUCTION

Wind farms extract energy from the wind and convert it to electrical energy. When multiple turbines operate together in a wind farm, an upstream turbine operating at maximum power point tracking (MPPT) can create suboptimal wind conditions for downstream turbines. There are currently many techniques to mitigate the negative impacts of wake interaction. Axial induction control can de-rate individual turbines in order to achieve power reserve maximization [1]. Tilt control [2] involves tilting the rotor plane to deflect the wake above or below downstream turbines. Finally, yaw misalignment is the intentional misalignment of upstream turbines with the prevailing wind direction to deflect the wake laterally away from downstream turbines [3], [4].

Typically, yaw angle control is based on simulated outputs from a wake deflection model, such as the FLOw Redirection and Induction in Steady-State (FLORIS) [5] model. Such models can produce a yaw schedule lookup table (LUT) based on wind farm parameters like wind speed and direction, as in [6]. However, this method is susceptible to irregularities that are difficult to model and that could decrease the benefit of the yaw optimization in the field. One solution is to implement model-free control methods, such as the game-theory-based approach described in [7], which treats wind turbines as agents that optimize a local value function, but also either assumes steady-state conditions or involves large time delays in the field due to wake propagation delays. Gebraad, et al. [8] propose incorporating a time delay into a distributed gradient-descent routine, which achieves faster convergence by only considering a turbine's nearest downstream neighbor. The approach in [8] successfully adapts a model-free, agent-based control method to a system with inherent time delays, but must re-converge every time wind conditions change.

This paper proposes a yaw control strategy using reinforcement learning (RL) techniques and examines its ability to account for wake propagation delay and stepwise-varying inflow conditions. RL is a subset of machine learning focusing on training autonomous agents to make decisions. The concept of agents, and their interactions with each other in a larger multiagent system (MAS), has proved to be widely useful [9]. However, RL has not been applied in as much depth to the wind industry, although other machine-learning techniques like neural networks have been studied [10]. Wei, et al. [11] used reinforcement learning techniques to implement MPPT for a single turbine, and Graf, et al. [12] used a distributed alternating direction method of multipliers algorithm in conjunction with RL to not only find optimal turbine yaw angles but predict wind condition trends and adjust accordingly. This paper builds on both of these approaches, extending the single-turbine approach from [11] to the wind farm case and adding time delay into an RL algorithm like [12] to increase farm energy capture.

This paper is organized as follows. Section II provides background on RL and how it is adapted to a wind farm application. Section III incorporates wake delay into the wind farm-RL environment. Section IV provides results from simulations examining the effectiveness of this algorithm when applied to a wind farm. Finally, Section V summarizes the work and provides several possible paths forward.

II. REINFORCEMENT LEARNING CONTROL

A. Baseline Controller

The baseline controller in this paper is based on an optimization routine provided with FLORIS [5]. This routine uses the Sequential Least SQuares Programming algorithm

[1]P. Stanfel and K. Johnson are with the Dept. of Electrical Engineering, Colorado School of Mines, Golden, CO 80401 USA (303-273-3914; e-mail: pstanfel@mymail.mines.edu and kjohnson@mines.edu)

*This material is based in part upon work supported by Envision Energy. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of Envision Energy.

[2] C. J. Bay and J. King are with the National Renewable Energy Laboratory, Golden, CO 80401 USA (email: christopher.bay@nrel.gov and jennifer.king@nrel.gov.

*This work was authored in part by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by the U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Wind Energy Technologies Office. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

to iterate through yaw angles for each turbine to maximize farm power. This is repeated for different wind conditions to assemble a LUT, which is then used to schedule yaw angles based on measured wind farm parameters, as in [6].

B. Reinforcement Learning

RL algorithms typically exploit Markov decision processes, or processes in which the future state depends only on the current state. A Markov decision process is characterized by a state and action space, a transition function that determines the probability of moving from one state to another by taking a certain action, α, and a reward function that determines what level of reward the agent receives after taking an action. The goal of RL is to arrive at a policy, $\pi^*(s)$, that determines the optimal action given state, s, such that reward is maximized. Since the goal of this paper is to develop a control algorithm that is less susceptible to model uncertainty and time variation, it is desirable to use a model-free version of RL. A common formulation of model-free RL is Q-Learning, which makes iterative updates of a state-action value function, $Q(s, \alpha)$. The iterative process records information about the system's state and actions taken and measures a reward signal to quantify the value of the state-action pair. The Q-Learning Bellman Equation [11] is:

$$Q_{t+1}(s_t, \alpha_t) = Q_t(s_t, \alpha_t) + l_t \left[r_{t+1} + \beta \max_{\alpha_i} Q_{t+1}(s_{t+1}, \alpha_i) - Q_t(s_t, \alpha_t) \right] \quad (1)$$

where $Q(s, \alpha)$ is the expected value of taking action, α,while in state s, $l_t \in (0, 1]$ is a learning rate determining how quickly the algorithm updates, r_{t+1} is an environmental reward signal, $\beta \in [0, 1)$ is a discount factor that weighs new information more than past information, and t is simulation iteration (or time, if applicable). At each iteration of the learning process, an agent (e.g., a wind turbine) selects an action, α_t, evaluates the reward, r_{t+1}, at the next time step, and updates its internal Q-table $Q(s_t, \alpha_t)$ based on the action and observed state before (s_t) and after (s_{t+1}) taking action, α_t. $Q(s, \alpha)$ is initialized to all zeros, and has dimensions equal to $M \times N \times ... \times size(A)$ where A is the action space (see Section II-C), and M, N, and so on are the lengths of the range of discrete values for each state. For yaw angle optimization, the agent's state consists of its yaw angle, γ and one or both of wind direction ϕ and wind speed U at the turbine. The resulting state vectors are thus $x = [U\ \gamma]^T$, $x = [\phi\ \gamma]^T$, or $x = [U\ \phi\ \gamma]^T$.

C. Action Space

Wei et al. [11] propose an action space of size three, with an action representing an increase or decrease in turbine rotor speed by a certain $\Delta\omega$ or no change in rotor speed. We adapt this approach for yaw angle control using the action space shown in (2):

$$A = \{\alpha(\Delta\gamma) : \alpha \in \{-1, 0, 1\}\} \quad (2)$$

This action space guarantees that an individual turbine will only be able to move by a small $\Delta\gamma$ every iteration.

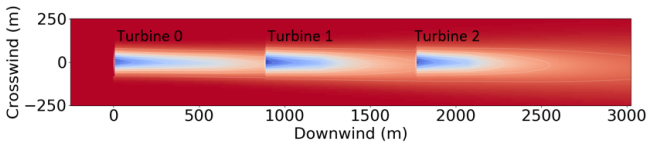

Fig. 1. Top-down view of the wind farm configuration used for simulations of the distributed RL algorithm.

(In this paper γ represents yaw angle with respect to the wind direction, which is a convention shared with FLORIS but is often called yaw error). This convention reduces the complexity of the Q-table by limiting the set of feasible yaw angles to a smaller range centered around 0.

As in [11], we use a Boltzmann action search to stochastically select an action. The Boltzmann search shown in (3) uses the changing Q-table to increase the probability of selecting actions with higher corresponding rewards.

$$p(s_t, \alpha_t) = \frac{e^{\frac{Q(s_t, \alpha_t)}{\tau}}}{\sum_i e^{\frac{Q(s, \alpha_i)}{\tau}}} \quad (3)$$

In (3), $p(s_t, \alpha_t)$ is the probability of choosing action, α_t, in state, s_t, while τ is a learning parameter. Tuning τ balances between exploration, in which seemingly less promising actions are chosen in the hopes of traversing local minima to find higher maxima, and exploitation, in which the action most likely to yield the highest reward is chosen [13].

D. Turbine Clustering and Value Function

The FLORIS flow field and layout used for simulations in this paper is shown in Fig. 1. Additionally, the wind direction convention is such that the wind direction shown in Fig. 1 is 270°, which is the wind direction used for this study. The simulation wind speed, unless otherwise specified, is 8 m/s.

To allow the distributed RL algorithm to maximize farm rather than turbine energy capture, it is necessary to allow some degree of communication between turbines so that agents, rather than a central controller, can accurately evaluate the impact of their actions on each other. Efficient communication requires choosing a subset of agents that are most likely to be impacted by a given agent's actions. Gionfra et al. [14] propose a method of clustering turbines into neighborhoods based on a simple rectangular zone extending in the crosswind and downwind directions. This paper uses a similar concept of clustering. Assuming that an agent knows the position of every turbine in the wind farm, it is possible to rotate the reference frame to the wind direction as in [4] and determine the set of neighbors, $\mathcal{N}(i)$, with $j \in \mathcal{N}(i)$ being a turbine agent that is within a distance downwind, ψ_d, and crosswind, ψ_c, in either direction of turbine i. With this clustering into neighborhoods, the value function, $V_i(t)$, that each turbine i aims to maximize is given by (4):

$$V_i(t) = P_{i,t} + \sum_{j \in \mathcal{N}(i)} P_{j,t} \quad (4)$$

with $P_{i,t}$ representing the power output of turbine i at iteration t. This value function is used to calculate the reward signal, r_{t+1}, as in (5), again drawing from [11]:

$$r_{i,t+1} = \begin{cases} 1 & V_{i,t+1} - V_{i,t} > \delta \\ 0 & \mid V_{i,t+1} - V_{i,t} \mid \leq \delta \\ -1 & V_{i,t+1} - V_{i,t} < -\delta \end{cases} \quad (5)$$

In (5), $\delta > 0$ is a small value that defines a dead zone to ensure that only significant changes generate reward.

By including the power outputs of downstream turbines in the value function of the upstream turbine, each agent must balance its own power with those of the downstream turbines. In the flow field shown in Fig. 1, Turbine 0 has Turbine 1 and Turbine 2 as neighbors, Turbine 1 has Turbine 2 as a neighbor, and Turbine 2 operates greedily, having no downstream turbines to consider. The downwind and crosswind distances, ψ_d and ψ_c, break more complicated wind farms into smaller subgroups based on wind direction.

E. Multiagent Reinforcement Learning

A MAS eliminates an important assumption that provides RL with its mathematical basis, namely stochasticity [13]. Adding more agents to the system creates an environment that an individual agent can not completely observe. While this makes some of the theoretical guarantees of single-agent RL difficult, [15] shows that applying single-agent techniques to a MAS can still improve performance, albeit not necessarily at the global optimum. This approach has been verified in [16]. In a similar manner, this paper aims to show that a distributed RL algorithm can yield energy capture increases while not trying to prove global optimization.

III. Simulation Setup

A. Simulation Environment

The FLORIS wake deflection model is a sequentially executed steady-state simulation environment. To allow downstream turbines to influence upstream turbines via control and include wake propagation delay, we first modify FLORIS's execution procedure for its wake deflection calculation to incorporate time delays to test and tune the RL algorithm.

Ideally, the FLORIS-based simulation model would operate as in Fig. 2, with FLORIS taking the place of the "Plant" block. This configuration includes a yaw offset error, $\gamma_{e,offset}$, which represents an error in the yaw angle or wind direction measurement. For field operation, the "Plant" block would be replaced by the actual wind farm.

In practice, because of the need for turbine-turbine interaction, the implementation of the algorithm on the augmented FLORIS requires several iterations through the wind farm agents for every simulated time step. This steady-state implementation consists of three primary phases (initialization, action selection, Q update) at each turbine, as shown in Fig. 3. Additionally, for most of the ensuing simulations, unless otherwise noted, the state vector is $x = [\phi\ \gamma]^T$.

Implementing this steady-state RL algorithm on the wind farm in Fig. 1 with instantaneous wake propagation (no delay) shows the performance of the algorithm in ideal, but highly unrealistic, circumstances. Results from this ideal case are shown in Fig. 4. In Fig. 4, yaw angles were limited to positive values, as they are in the FLORIS yaw optimization routine. This simulation achieved 8.2% power gain above the baseline (0° yaw angle for all turbines), which is the same power gain achieved by the FLORIS optimization routine and demonstrates the ability of the RL algorithm to match FLORIS's internal optimization routine in the ideal case.

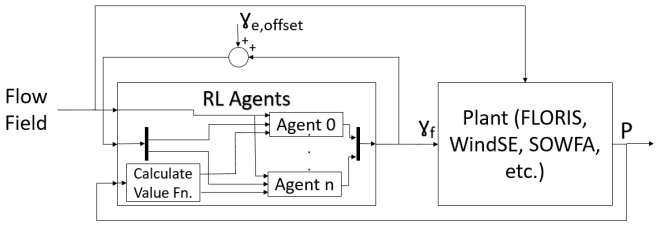

Fig. 2. High-level block diagram implementing the distributed multiagent learning algorithm. Each agent takes in data from the environment and uses it to choose a yaw angle. The output, P, from the plant is a vector of turbine power measurements and is passed to the turbines in the farm, which use their own internal logic to determine which measurements are relevant to their own value function calculation. Each turbine agent updates its Q-table using its own value function and yaw angle.

B. Quasi-Dynamic Simulation

Using Taylor's frozen wake hypothesis, it is possible to build an approximation of wake delay into FLORIS such that a quasi-dynamic environment can be simulated [17]. Bay, et al. [18] use Taylor's hypothesis to incorporate wake delay into FLORIS by using the freestream velocity U_∞, as in (6):

$$t_{delay} = \frac{d}{U_\infty} \quad (6)$$

with d representing the downwind distance between the upstream and downstream turbines. This propagation delay adds additional complexity for the RL algorithm. When a turbine changes its yaw angle from 0° to some non-zero yaw angle, the immediate effect is a decrease in total farm power, since the reduced wake has not yet reached the downstream turbines. This short-term power decrease is shown in Fig. 5, starting at times $t = 250$ s, 500 s, 750 s, and 1000 s. As the wake propagates first to Turbine 1 and then to Turbine 2, however, additional power gain is achieved due to the wake deflection from Turbine 0. For this plot, the dynamics of the yaw drive are not modeled because they are still substantially faster than the wake propagation delay time.

When the quasi-dynamic FLORIS environment is used, the RL algorithm described in Section II cannot achieve a power gain, which we investigate in Section III-C.

C. Reinforcement Learning Algorithm Locking

Many yaw angle optimization routines assume unrealistic steady-state conditions and then fail in the quasi-dynamic environment. Therefore, it is necessary to adapt the algorithm accordingly. Guestrin et al. [19] show that multiagent

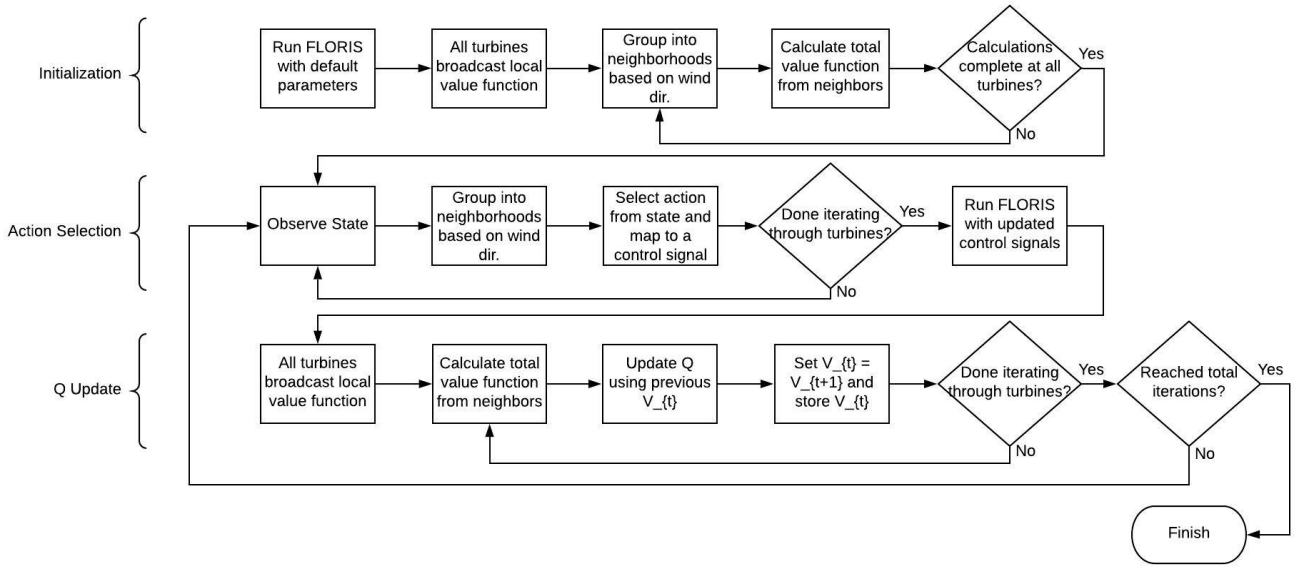

Fig. 3. Simulation of RL turbine operation using the augmented steady-state FLORIS software package The initialization stage is performed just once, while the action selection and Q update stages are repeated for the desired number of iterations.

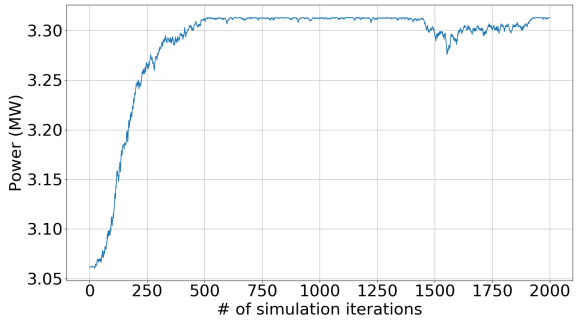

Fig. 4. Total wind farm power output of the steady-state distributed Q-learning algorithm using FLORIS and assuming no wake propagation delay.

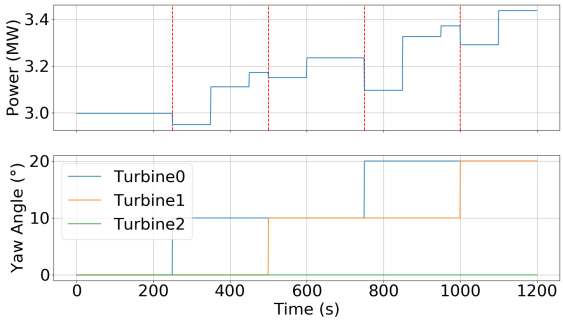

Fig. 5. Visualization of the dynamic modifications to FLORIS. The yaw angles of either Turbine 0 or Turbine 1 were changed at 250 s increments. At each yaw angle step, the wind farm yaw angle configuration moves progressively closer to the optimal value from the baseline FLORIS optimization. Power does not immediately increase with a change in yaw angle due to the wake propagation delay time.

learning can be made more efficient if the system can be broken into subgroups that can be optimized efficiently. If the subgroups are not known or are time-varying, then "locking" agents until certain environmental conditions are satisfied is an effective substitute.

The wind farm can readily be divided into subgroups, or neighborhoods, based on wake interactions caused by the wind direction, as in [14]. However, because these neighborhoods change with wind direction, the system structure cannot be completely predicted at any given moment, so the agent locking mechanism described above is promising.

Ideally, a time-delayed algorithm would simply calculate the time it takes for the wake delay to reach downstream turbines, as in (6) and wait for that period of time before determining the reward signal. This strategy might work if it were not possible for one turbine to be upstream relative to some turbines and downstream relative to others. However, due to multiple turbine interactions and the need to remain flexible to changing wind direction when defining neighborhoods, simply waiting a specified period of time before performing an update on the Q-table is insufficient, as demonstrated in the following scenario.

Assume that Turbine 0 and Turbine 1 initiate a yaw misalignment control action at approximately the same time step. Immediately, farm output decreases (e.g. at $t = 250$ s, as shown in Fig. 5), but it eventually rises. Even if both turbines calculate the approximate wake delay using (6) and wait for this interval to update their Q-tables, similar to the approach in [8], when both turbines evaluate their reward signal using

(4), Turbine 2 will have experienced wake effects from both Turbine 0 and Turbine 1. Therefore, the reward signal broadcast to Turbines 0 and 1 will be a combination of both turbines' actions, and the algorithm cannot distinguish between Turbine 0's and Turbine 1's actions. Additionally, when Turbine 1 calculates its value function, it will also be experiencing the impacts of the wake deflection of Turbine 0, so its own value function calculation will be skewed.

To avoid the confusion created within this scenario, a system of two-way turbine communication is implemented. When a turbine yaws, it sends a "lock" signal to all downstream turbines, preventing them and itself from moving until the time determined by (6), at which point the downstream turbines unlock. A turbine cannot move until both itself and all turbines in its neighborhood are unlocked. Once a turbine unlocks after initiating a control action, it calculates the reward signal as before. The algorithm is outlined in Algorithm 1. Note that `completed_action` represents a Boolean value that tracks whether or not a particular turbine has initiated a control action. The turbine only updates its Q-table when the Boolean is true.

Algorithm 1 The locking algorithm for turbine i. This algorithm runs simultaneously for all turbines in the farm.

if i *and* $j \; \forall \; j \in \mathcal{N}(i)$ *are unlocked* **then**
  select action
  calculate wake delay for turbines in neighborhood
  send lock signal to downstream neighbors
  set `completed_action` boolean
end if

if *turbine* i `completed_action` **then**
  calculate value function and reward signal
  update Q-table
  unset `completed_action` boolean
end if

Using the locking procedure in Algorithm 1 allows each turbine to assess the impacts of its actions on the reward signal (5). The benefit of the locking procedure is shown in Fig. 6, which shows results from both the original unlocked RL and the RL with locking when simulated in the wake-delay-augmented FLORIS. The unlocked RL was unable to increase power, whereas the RL with locking can. Although earlier figures used "# of simulation iterations" as the x-axis, the FLORIS updates for wake propagation delay necessitate a time-based independent axis, as shown in Fig. 6.

IV. Results

A. Model-Based/Model-Free Hybrid

The preceding sections have demonstrated that a distributed RL algorithm can effectively improve the performance of a wind farm in both steady-state and quasi-dynamic environments. These results suggest an opportunity for a two-stage hybrid control approach. Because the RL algorithm is able to adjust to modeling inaccuracies, it can be combined with a fixed LUT of optimal yaw angles. A simplified model such as FLORIS can be used offline to create a "good-enough" estimate of what the yaw schedule should be, and the dynamic Q-learning algorithm then can take over in the field. The results from this combination of model-based optimization followed by model-free learning are shown in Fig. 7. In this case, the steady-state RL training takes place outside of "real" time, but time in seconds begins upon implementation on the "farm" (FLORIS model). First, the three-turbine wind farm was trained on the simple steady-state FLORIS model for 2000 iterations. Then, the Q-table was moved to agents "in the field," represented by the quasi-dynamic version of FLORIS. The "field" simulation included a -10° yaw offset error, which would cause suboptimal performance for the fixed FLORIS-based LUT. This error implementation is shown by the $\gamma_{e,offset}$ signal in Fig. 2.

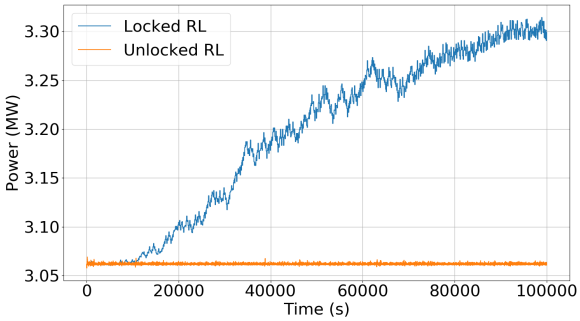

Fig. 6. Comparison of the performances of the dynamic and steady-state locking RL algorithms in a quasi-dynamic FLORIS environment.

The quasi-dynamic phase also involved changing the learning rate, l_t, used in (1). In [11], l_t is given as:

$$l_t = \frac{k_1}{k_2 + k_3 n(s_t)} \tag{7}$$

with k_1, k_2, and k_3 being tuning coefficients and $n(s_t)$ being the number of times that the agent has visited state s_t. This means that the learning rate decreases as the turbine visits a state more often. In our RL algorithm, however, because agents have to "unlearn" past information, the learning rate is fixed at $l_t = 0.9$, not decreased with each visit to the state.

As shown in Fig. 7, the algorithm successfully adjusts to the yaw offset error, eventually reaching almost the same optimal power value as the ideal case (Fig. 4).

B. Changing Wind Conditions

The preceding simulations, although including wake delay, are still very unrealistic in that they assume that wind conditions stay constant for unreasonably long times. Changing wind speeds pose an interesting problem for the yaw control algorithms, as at higher wind speeds it is less important for the upstream turbines to be misaligned since the waked wind is still sufficient for maximum power operation. Fig. 8 illustrates the effect of changing wind speeds on each turbine's yaw angle. As the ambient wind speed increases, the optimal

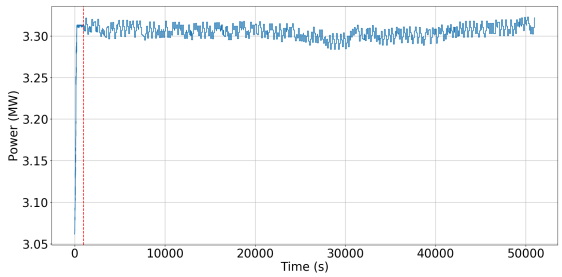

Fig. 7. Steady-state RL training with implementation in a quasi-dynamic simulation environment with a yaw offset error. In this simulation the turbines are at their optimal yaw angles when the quasi-dynamic simulation begins, so power does not drop.

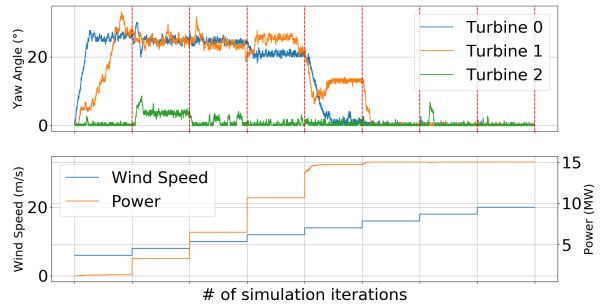

Fig. 8. Yaw angle plots for each of the three turbines in the wind farm, with respect to a changing wind speed.

yaw angles for Turbine 0 and Turbine 1 move to 0°. This means that the RL algorithm correctly identifies that the upstream turbines do not need to yaw as much out of the wind to redirect the wake. For this simulation, the state was given as $x = [U\ \gamma]^T$ with no yaw offset error.

V. Conclusions and Future Work

This paper presents a proof of concept of the ability of a distributed reinforcement learning algorithm to maximize energy capture potential at a farm, rather than a turbine, level, assuming sufficient communication architecture is in place.

We presented results from four cases, starting with an ideal case with no wake propagation delay, then incorporating wake delay and an updated, "locking" version of the RL algorithm. The third case considered a two-step procedure that started with a "good-enough" estimate using the FLORIS wake deflection model and later fine-tuned itself via the wake delay learning algorithm, and in the fourth case we showed the algorithm's responsiveness to changing wind speeds.

There are several avenues for future work. One is to make the simulation environment more realistic, or to use higher-fidelity simulators. Another is to create more realistic test cases, such as using wind distributions that exhibit larger-scale patterns or less constant wind conditions to examine, respectively, reinforcement learning's ability to recognize and its susceptibility to more variable wind conditions. This approach also creates new opportunities for wind farms to respond to disturbances, such as loss of a turbine. The assumptions of stepwise constant wind speeds and directions simulated in this paper are unrealistic, and the wind farm size of just 3 turbines is uncommon in a U.S. context. However, our results show that it is at least possible in theory to "learn on the fly" in a multiagent, time-delayed environment and achieve significant increases in performance at a farm level.

References

[1] S. Siniscalchi-Minna, F. D. Bianchi, M. De-Prada-Gil, and C. Ocampo-Martinez. A wind farm control strategy for power reserve maximization. *Renewable Energy*, 131:37–44, 2019.

[2] J. Annoni, A. Schoolbrock, M. Churchfield, and P. Fleming. Evaluating tilt for wind farms. In *2017 American Control Conf.*, pages 693–698, 2016.

[3] P. Fleming, P. MO Gebraad, S. Lee, JW van Wingerden, K. Johnson, M. Churchfield, J. Michalakes, P. Spalart, and P. Moriarty. Simulation comparison of wake mitigation control strategies for a two-turbine case. *Wind Energy*, 18(12):2135–2143, 2015.

[4] P. MO Gebraad, FW Teeuwisse, JW Van Wingerden, P. A. Fleming, SD Ruben, JR Marden, and LY Pao. Wind plant power optimization through yaw control using a parametric model for wake effects – a cfd simulation study. *Wind Energy*, 19(1):95–114, 2016.

[5] National Renewable Energy Laboratory. FLORIS. Version 1.0.0. `https://github.com/NREL/floris`, 2019.

[6] P. Fleming, J. King, K. Dykes, E. Simley, J. Roadman, A. Scholbrock, P. Murphy, J.K. Lundquist, P. Moriarty, K. Fleming, J. van Dam, C. Bay, R. Mudafort, H. Lopez, J. Skopek, M. Scott, B. Ryan, C. Guernsey, and D. Brake. Initial results from a field campaign of wake steering applied at a commercial wind farm – part 1. *Wind Energy Science*, 4(2):273–285, 2019.

[7] J.R. Marden, S.D. Ruben, and L.Y. Pao. A model-free approach to wind farm control using game theoretic methods. *IEEE Trans. on Control Systems Technology*, 21(4):1207–1214, 2013.

[8] P. MO Gebraad, F.C. van Dam, and JW van Wingerden. A model-free distributed approach for wind plant control. In *2013 American Control Conf.*, pages 628–633, 2013.

[9] X. Meng and Z. Pian. *Intelligent Coordinated Control of Complex Uncertain Systems for Power Distribution and Network Reliability*. Elsevier, 11 2016.

[10] A.P. Marugán, F.P.G. Márquez, J.M.P. Perez, and D. Ruiz-Hernández. A survey of artificial neural network in wind energy systems. *Applied Energy*, 228:1822–1836, 2018.

[11] C. Wei, Z. Zhang, W. Qiao, and L. Qu. Reinforcement-learning-based intelligent maximum power point tracking control for wind energy conversion systems. *IEEE Trans. on Industrial Electronics*, 62(10):6360–6370, 2015.

[12] 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 Conf.*, pages 4159–4166, 2019.

[13] K. Tuyls and G. Weiss. Multiagent learning: Basics, challenges, and prospects. *AI Magazine*, 33(3):41–41, 2012.

[14] N. Gionfra, G. Sandou, H. Siguerdidjane, D. Faille, and P. Loevenbruck. Wind farm distributed pso-based control for constrained power generation maximization. *Renewable Energy*, 133:103–117, 2019.

[15] L. Buşoniu, R. Babuška, B. De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[16] M. Riedmiller, A. Moore, and J. Schneider. Reinforcement learning for cooperating and communicating reactive agents in electrical power grids. In *Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, pages 137–149, 2000.

[17] G.I. Taylor. The spectrum of turbulence. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 164(919):476–490, 1938.

[18] C.J. Bay, J. Annoni, T. Taylor, L. Pao, and K. Johnson. Active power control for wind farms using distributed model predictive control and nearest neighbor communication. In *2018 Annual American Control Conf.*, pages 682–687, 2018.

[19] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234, 2002.