Reinforcement Learning for Firm Power Tracking in Wind Farms

Aoife Henry[1], Josh Myers-Dean[1], Michael Sinner[2], Joewie Koh[1], Alessandro Roncone[1], Jennifer King[2], Lucy Pao[1]

***Abstract*— Efforts to make wind energy, generated by wind farms, a viable replacement for fossil fuel-based alternatives depend on reliably maximizing the efficiency of wind farm energy generation. Maximizing the utility of such wind farms relies on control methods which can optimally operate the individual turbines on a second-by-second basis to maximize the total farm-level power made available to the grid. Given the time-varying and stochastic nature of wind farms: unpredictable wind fields, wakes generated by turbines, turbulence, obtaining an accurate model of the wind field dynamics is not realistic. We implement a model-free, dual time-scale reinforcement learning (RL) algorithm to learn both the optimal yaw and axial induction factor control actions of each wind turbine to track a given time-varying power reference while limiting the yaw travel and loading experienced by the turbines.**

I. INTRODUCTION

Energy produced by wind farms have potential to satisfy increasing demand for fossil-free energy within the constraints of the electricity grid. They can be controlled to attain different, sometimes competing objectives, such as power maximization, power tracking or wind turbine fatigue minimization [1]. However, in order for wind farms to present as a viable alternative to more carbon-intensive sources of energy, it is critical that they can be controlled to fulfill the reliability demands of the grid.

Typically, the objective of wind farm control is to maximize the total power generated by the wind farm while minimizing the fatigue loading on the turbines. However, in this work we propose an alternative objective, that of *firm power tracking*. This involves tracking a given time-varying power reference, which the wind farm operators offered to the transmission systems operator (TSO) ahead of time, as accurately as possible. If wind energy is to remain competitive in the energy market, they must have the capacity to meet more complex grid needs such as this. The consequence of naively maximizing power generation is that wind farms may be curtailed, thus losing valuable fossil-free energy generation.

[1] University of Colorado Boulder, Boulder, CO
[2] NREL, Golden, CO

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.

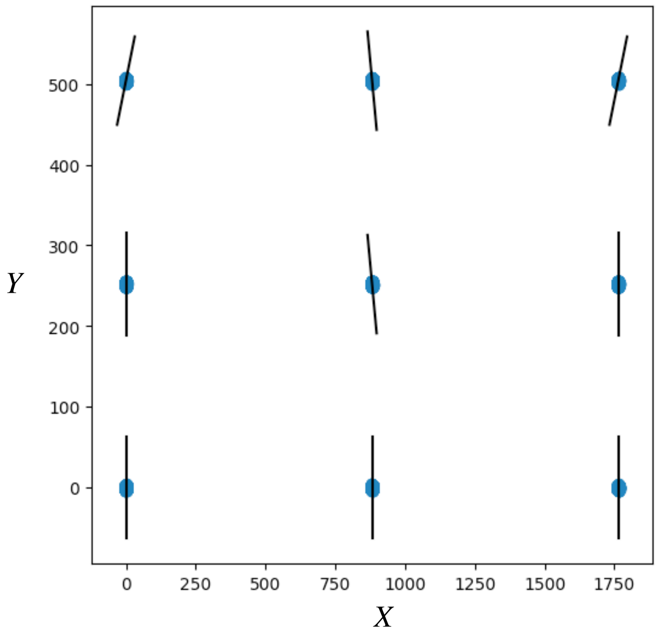

Fig. 1: Example of a Floating Offshore Wind Farm with varying yaw angles. The blue Gaussians represent the possible locations of each turbine.

A wind farm is a time-varying environment with *stochastic* dynamics, in that a) at any moment there is a nonzero probability p_{off} that any given wind turbine could 'go offline', i.e. fail and temporarily cease to operate or be intentionally curtailed by the operator for maintenance reasons, and b) the wind speed and direction incident upon each turbine is subject to variation in time and atmospheric turbulence. To achieve any wind farm objective, we have two control actions for each wind turbine available: the yaw angle γ - which controls the lateral orientation of the wind turbine rotor - and the axial induction factor a - which controls what proportion of the wind's energy the turbine extracts. There are several challenges associated with yaw control. Firstly, there is a significant time-delay between when a the yaw command is transmitted and the time it takes to execute it. Secondly, there is a tight constraint on the rate of change of the yaw angle (which is generally fixed) and the overall yaw travel due to the mechanical loads experienced by the yaw bearing and other wind turbine components during rotation. Thirdly, naive yaw control is *greedy*, in that each wind turbine aligns itself into the wind based on local measurements in order to maximize the power it generates. A consequence of this greedy operation is that wakes generated behind wind turbines, can result in a wind field of reduced velocity and increased turbulence for downstream wind turbines. This is known as the *wake effect* and is difficult to model accurately with analytical models. This motivates the use of a model-free centralized controller to determine the optimal operation of all wind turbines in a farm to manage farm-level power generation.

The RL controls methodology involves training one or more 'agents' to generate optimal control actions by allowing to interact with the environment, rewarding actions that results in positive outcomes and/or penalizing actions that result in negative outcomes. deep reinforcement learning (DRL) is a *model-free* variation that employs artificial neural networkss (ANNs) to learn the value functions central to the approach. It has been shown to perform well in [**?**]

Some work has been published on the application of RL methods to wind farm control. Yang et al present a deep RL method applied to a wind farm with connected energy storage and external reserve purchasing that considers forecasted wind conditions and electricity prices to maximize the farm's long-term revenue. Futakuchi et al similarly optimize the scheduled operation of a wind farm with energy storage using DRL to manage ramp events in [2]. It assumes that the wind turbines can be commanded to extract a given amount of power from the wind, rather than explicitly considering the actuation variables [3]. Saenz-Aguirre et al apply a RL Q-Learning method to yaw control of wind farms [4]. The authors do not consider axial induction factor as an actuation variable or indeed platform relocation considering that the study is based on an onshore wind farm. Zhao et al propose a knowledge-assisted RL framework to control the axial induction factor of turbines in a wind farm based on the freestream wind speed with integrated low-fidelity analyctical wake models to ensure safety of the wind turbines during the online learning process [5]. The authors consider a varying wind-speed but a constant wind direction, which does not fully test the true use-case for yaw control. Dong et al present a RL methodology to control the yaw angles of a collection of wind turbines that is robust to uncertain wind conditions and yaw actuation models [6]. Dong and Zhao develop a robust RL methodology to track a time-varying farm-level power reference by controlling the thrust coefficient (equivalent to controlling the axial induction factor) of the turbines and including previews of the reference signals as augmented states in the system [7]. The authors choose not to consider yaw angles due to the time delay, which limits its capacity to track a time varying power reference on a shorter time scale. Vijayshankar et al propose a SARSA-based DRL approach to track a time-varying power reference signal via yaw control [8]. Stanfel et al present a distributed, multi-agent RL approach to maximize farm-level power generation via yaw control [9]. None of the above works consider both yaw and axial induction control to track power reference changes over a range of time-scales, nor do they consider the additional actuation variable of platform relocation in the offshore wind-farm case. They also assume a fixed wind farm layout and do not consider the possibility of one or more wind turbines going offline for maintenance purposes or due to faulty equipment.

There are various challenges in the field of wind farm control that make it suitable but challenging for an RL approach. Firstly, with offline training of any model, the wake field can only be predicted for the wind farm layout (i.e. fixed turbine locations and online/offline status), to support this, we include wind conditions and control parameters included in the training data. Low-fidelity static wake models are not sufficiently accurate to predict unsteady wake fields and high-fidelity models require too much time to generate predictions in a model-based controls algorithm, making this a suitable environment for RL. Secondly, the full set of inputs that contribute to the total wind-farm level power includes each turbine's control actions and wind speed/direction measurements across the wind farm, and thus is high-dimensional and noisy (QUESTION is RL robust to this). Thirdly, DRL provides a paradigm that is robust to stochasticity in the environment and model uncertainty by leveraging deep learning methods (QUESTION is this true). Finally, centralized control of wind farms is a non-tractable problem, given the large action and state space this presents a challenge for modern control algorithm but an environment that could be suitable for deep RL methods that are capable of dealing with high dimensional data.

In this work, we develop a framework to learn the optimal control actions for each wind turbine in a wind farm using a DRL methodology. Our contributions are as follows:

- We develop a single-agent RL algorithm for stochastic, time-varying control of wind farm environments using the deep deterministic policy gradient (DDPG) algorithm to track a given time-varying power reference. Unlike previous works we adopt a *dual time-scale* approach to generate control actions for both the yaw misalignment and the axial induction factor.
- We employ a reward functions that track a time-varying power reference and limit the yaw travel of the turbines.
- We consider a state-space of autoregressive observations in the wind farm to account for the delayed influence of historic changes in the freestream wind field and wind turbine actuation on the current wind field experienced by individual turbines.
- We employ a replay buffer that is guaranteed to contain training data collected from 'novel' wind farm layouts in which one or more turbines were not operational and thus not significantly influencing the wind farm dynamics.
- We test validate the framework on a simulated 9×9 wind farm across multiple episodes.

II. Methodology

A. Reinforcement Learning

In RL, an *agent* learns how to map observations of its environment to actions so as to maximize a numerical reward via a trial-and-error approach, where actions taken in the present may influence not only the immediate next reward but also subsequent rewards [10].

At each time-step t, the agent tries to select an action $a_t \sim \pi(a_t|s_t)$ that maximizes the *discounted return* with a *discount rate* $\gamma \in [0,1]$:

$$\begin{aligned} G_t &= \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i) \\ &= r(s_t, a_t) + \sum_{i=t+1}^{T} \gamma^{(i-t)} r(s_i, a_i) \\ &= r(s_t, a_t) + \gamma G_{t+1} \end{aligned} \tag{1}$$

The *action-value function for policy* π describes the expected return after taking an action a_t from state s_t and thereafter following a policy π, and is thus is a measure for how good it is to perform a given action in a given state:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t | s_t = s, a_t = a\right] \tag{2}$$

The action-value function satisfies the recursive relationship known as the *Bellman equation*:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}\left[G_t | s_t = s, a_t = a\right] \\ &= \mathbb{E}\left[r_t + \gamma G_{t+1} | s_t = s, a_t = a\right] \\ &= \mathbb{E}\left[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})\right] \end{aligned} \tag{3}$$

If the policy can be described as a deterministic function of the state, $\mu(s)$, then:

$$Q^{\mu}(s,a) = \mathbb{E}\left[r_t + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right] \tag{4}$$

and the expectation depends only on the environment's influence on r_t and s_{t+1} (and not on the influence of a stochastic policy π on a_{t+1}).

Temporal-Difference (TD) RL methods update estimates of value functions, such as $Q^{\mu}(s,a)$, based on other current estimates. In particular, *one-step TD* ($TD(0)$) performs the following update for a given non-terminal state, s_t:

$$\begin{aligned} Q(s_t, a_t) \leftarrow & Q(s_t, a_t) \\ & + \alpha^Q \underbrace{\left(\underbrace{r_t + \gamma Q(s_{t+1}, a_{t+1})}_{\text{target value}} - \underbrace{Q(s_t, a_t)}_{\text{current estimate}} \right)}_{\text{TD error, } \delta} \end{aligned} \tag{5}$$

Policy-gradient methods involve learning a parameterized policy, $\pi(a|s,\theta)$, by updating the parameters based on the gradient of a performance measure, $J(\theta)$:

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \hat{J}(\theta) \tag{6}$$

An *actor-critic* method is a policy-gradient approach which involves learning a policy, $\mu(s)$, (the *actor*) and a value function, $Q^{\pi}(s,a)$, (the *critic*) to assess the action with. deterministic policy gradient (DPG) [11] is one such method which learns

- a deterministic parameterized actor function, $\mu(s|\theta^{\mu})$ with a policy-gradient update:

$$\theta^{\mu} \leftarrow \theta^{\mu} + \alpha^{\mu} \nabla_{\theta^{\mu}} \hat{J}(\theta^{\mu}) \tag{7}$$

  where the *performance measure* is defined as the expected discounted return from the first-time step:

$$J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \mu}\left[G_1\right] \tag{8}$$

  and the gradient with respect to the actor function's parameters is found by applying the chain rule of differentiation:

$$\begin{aligned} \nabla_{\theta_{\mu}} J &= \mathbb{E}_{s_t \sim E}\left[\nabla_{\theta_{\mu}} Q^{\mu}\left(s_t, \mu(s_t|\theta^{\mu}) | \theta^Q\right)\right] \\ &= \mathbb{E}_{s_t \sim E}\left[\nabla_{a} Q^{\mu}\left(s_t, \mu(s_t|\theta^{\mu}) | \theta^Q\right) \nabla_{\theta^{\mu}} \mu(s_t|\theta^{\mu})\right] \end{aligned} \tag{9}$$

  and,
- learns a parameterized critic function, $Q^{\mu}(s,a|\theta^Q)$ with the $TD(0)$ update:

$$\begin{aligned} \theta^Q \leftarrow & \theta^Q + \alpha^Q \nabla_{\theta^Q} Q^{\mu}(s_t, a_t|\theta^Q) \\ & \left(\underbrace{r_t + \gamma Q^{\mu}(s_{t+1}, a_t|\theta^Q)}_{\text{target value}} - \underbrace{Q^{\mu}(s_t, a_t|\theta^Q)}_{\text{current estimate}} \right) \end{aligned} \tag{10}$$

Employing ANNs for the nonlinear function approximation of the action-value function and/or the policy is known as *DRL*. In particular, learning the action-value function with an ANN like this is known as deep Q network (DQN) [12]. In an actor-critic method, the parameterized critic can be learned by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}\left[\left(\underbrace{r_t + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}|\theta^{\mu}))}_{\text{target value}} - \underbrace{Q^{\mu}(s_t, a_t|\theta^Q)}_{\text{current estimate}} \right)^2\right] \tag{11}$$

,

and the parameterized policy can be learned by minimizing the loss:

$$L(\theta^{\mu}) = \mathbb{E}\left[-Q^{\mu}(s_t, \mu(s_t|\theta^{\mu})|\theta^Q)\right] \tag{12}$$

Directly implementing DQN has proven to be unstable in many environments since the learned critic network $Q^{\mu}(s,a|\theta^Q)$ being updated is also used to calculate the target value, so the update is prone to divergence.

DDPG is an actor-critic, model-free RL algorithm. It is based on principles of DQN [12] and DPG [11] that can robustly control systems with continuous action spaces and model uncertainty [13]. It resolves the instability issues of DQN by:

- creating a copy of the actor and critic networks, $\mu'(s|\theta^{\mu'})$ and $Q'(s,a|\theta^{Q'})$, respectively to calculate the target values,

- training the networks off-policy with samples from a replay buffer to minimize temporal correlations between samples, and,
- applying "soft" updates to the weights of these target networks to give consistent targets during the updates:

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{13a}$$

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \tag{13b}$$

  where a small value of $\tau << 1$ means that the target weights are constrained to be updated very slowly, greatly improving the stability of learning at the cost of slower convergence.

B. Parameter Sharing

Naively employing DDPG in a multi-agent context, would involve an actor network, critic network, target actor network and target critic network for each agent. A less computationally-expensive alternative is to implement *parameter-sharing* [14]. Since the observation and action space of each turbine are heterogeneous, rather than maintaining four networks for every agent, we use one of each for all agents. This can be achieved by adding an *agent indicator* variable to the observation space to distinguish different agents.

III. Wind Farm Environment

The objective of the wind farm control presented in this work is to implement and evaluate a working RL algorithm to learn the optimal operation of a 3×3 wind-farm given stochastic variations in freestream wind speed, wind directions and online status of each turbine. To this end, we define the features of the wind farm environment, observation space and action space in this section for the purposes of integration with a RL-based controls framework such as DDPG.

A. Continuous Action Space

Each turbine, denoted T_i for $i \in \mathcal{T}$, is controlled by an agent, which chooses the axial induction factor, $a_i \in [0, \frac{1}{3}]$, and yaw angle, $\gamma_i \in [-30°, 30°]$, for that turbine at each time-step. These values are normalized to be between 0 and 1 during learning to aid convergence of the neural networks.

B. Auto-Regressive & Previewed Observation Space

The observations received by the agent at each time-step, o_t, include the yaw and axial induction factor settings executed by each turbine, the binary 'offline' variable corresponding to each turbine, the freestream wind speed and the freestream wind direction over the last k_{delay} time-steps. k_{delay} is chosen as $\frac{\overline{U}_\infty}{\delta t \underline{x}_d}$ based on Taylor's frozen wake hypothesis, where we assume that, given a maximum expected freestream wind speed $\overline{U}_\infty$ and the minimum downstream distance between any two turbines, $\underline{x}_d$, the wake induced by an upstream turbine will take at most k_{delay} time-steps to propagate downstream. Considering that historic adjustments to yaw misalignment and axial induction factor in upstream turbines influence the wind field experienced, and thus the power generated, by downstream turbines, we need to consider these *auto-regressive* inputs in our observation space. Wind farm operators know the power reference signal ahead of time since they provided it to the TSO. We can exploit this *preview information* by including the values from a given future number of time-steps, k_{preview}, in our observation vector. As mentioned in Section II-B, an additional variable as appended to the observation vector for each turbine denoting its unique identifier (i.e. a number between 1 and the number of turbines, $|\mathcal{T}|$). Similar to the action values, the observations values are normalized to be between 0 and 1.

C. Wind Farm Dynamics

Several features of wind farm environment considered in this work are stochastic and time-varying:

- the mean freestream wind-speed and direction changes with probability $p_{\delta U}$ in positive or negative steps of δU_∞ and $\delta \hat{U}_\infty$, respectively,
- the turbulence added to the mean freestream wind-speed and direction is sampled from a Gaussian distribution with zero-mean and variances σ^2_{ws} and σ^2_{wd}, respectively.
- the online status of any given turbine could be set to `false` with a probability p_{offline} at each time-step, such that it will cease to operate and effectively have no influence on the wind field dynamics within the wind farm

D. Training Episodes

For each of N_{tr} training episodes, $E_j, j \in \{1, \ldots, N_{\text{tr}}\}$ we randomly set the initial freestream wind speed and direction (which is thereafter stochastically incremented or decremented randomly at each time-step). Each episode is set to be 1 hour in length with a discrete time-step of 1 s.

E. Dual Time-Scale Control

The yaw actuator works on a slow time-scale (minutes) and it is preferable to limit the yaw travel (the total rotational distance traveled over a given time period) such that the yaw bearing will not need to be prematurely replaced. It is thus not feasible to track a fast time-varying power reference with power alone. The purpose of the yaw control in this work is to make approximately enough wind power available to track the power reference without incurring excessive rotor thrust or yaw travel. The actuators which support axial induction factor are comparatively fast (on the order of seconds). In this work we control the axial induction factor to closely track a given power reference once the yaw control has been executed to make sufficient power available. These distinct time-scales are implemented by holding a constant yaw angle command until the yaw angle time interval, δt_γ has passed, and equivalently holding a constant the axial induction factor command until the axial induction factor time interval, δt_a, has passed.

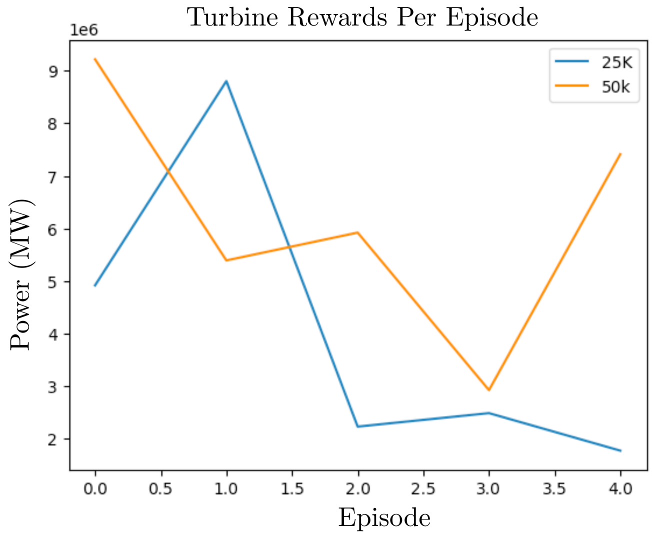

Fig. 2: Rewards when evaluating our learned policy from PPO trained for 25,000 and 50,000 time steps, respectively, across 5 episodes. Overall, we observe a deviation from turbine power between the models in each episode.

F. Reward Functions for Firm Power Tracking, Rotor Thrust Attenuation & Yaw Travel Attenuation

The reward function depends on the power tracking error, the yaw travel undergone by each wind turbine and the rotor thrust force experienced by each wind turbine:

$$r_t = \beta_P e^{-\alpha_P \|P_t^{\text{wf}} - P_t^{\text{ref}}\|} + \beta_\gamma e^{-\alpha_\gamma \max_i \Gamma_t^i} + \beta_T e^{-\alpha_T \max_i T_t^i} \tag{14}$$

where $P_t^{\text{wf}} = \sum_{i \in \mathcal{T}} P_t^i$ is the total power generated by the wind farm, P_t^{ref} is the power reference signal, Γ_t^i is the yaw travel experienced by turbine i over the last 10 minutes, T_t^i is the rotor thrust force experienced by turbine i at the current time-step, and $\beta_P, \beta_\gamma, \beta_T$ are weighting coefficients.

G. Deliverable and Implementation Plan

See Table I.

Task	Deadline	Intended Deadline	Member
Define Observations, State Space, Action Space, Reward Function, Transition Dynamics	-	Done	Aoife & Josh
Install Libraries	-	11/01/2022	Aoife & Josh
Write Gym Environment subclass	-	11/01/2022	Aoife & Josh
Write Agent Class	-	11/03/2022	Aoife & Josh
Integrate QMIX Algorithm	-	11/04/2022	Aoife & Josh
Transfer workflow to RC	-	11/07/2022	Josh
Generate training episodes	-	11/10/2022	Josh & Aoife
Train model	-	11/11/2022	Josh
Submit final report	12/07/2022	11/18/2022	Aoife & Josh

TABLE I: Project Timeline

H. Software Tools

We use a suite of wind turbine simulation and RL software packages throughout our project. Specifically, we used FLORIS [15] to model the wake and turbine dynamics (e.g., controls, positions) for our environment. We used PyTorch [16] as the deep learning framework back-end for our experiments. We implement our RL environment with the popular OpenAI Gym [17] framework to allow for ease of supplying actions and receiving observations for our agents. To conduct our experiments, we implemented the QMIX algorithm in RLLIB [18], during which we unfortunately encountered incomplete evaluation support. Finally, we implemented our single-agent algorithm, PPO, using the Stable Baselines 3 [19] API.

I. Simulating a Stochastic Environment

There are multiples sources of uncertainty considered in the wind farm model implemented in this work. We simulate the setting where a turbine can go offline, how turbines vary in the ocean, and wind dynamics. First, The probability of any given turbine going 'offline' (i.e. either being purposefully switched off by the operator or failing due to a mechanical issue) at each time-step is given by $p_{\text{off}} = 0.001$. We use this value as a heuristic and future work could involve drawing this probability from Bernoulli distribution for each turbine.

Given that the turbines float in the water, they do not have a fixed location at any given time. Formally, we model the location of each wind turbine t at each time-step, k, is drawn from a two-dimensional Gaussian distribution:

$$t_k \sim \mathcal{N}\left(\begin{bmatrix} \mu_{x,t} & 0 \\ 0 & \mu_{y,t} \end{bmatrix}, \sigma_{xy}^2 \mathbf{I_2}\right) \tag{15}$$

where $\mu_{x,t}$ and $\mu_{y,t}$ are the location set-points of turbine t in the x and y planes, respectively, and $\sigma_{xy}^2 = 1$ is the variance of the location distribution, which we consider to be constant across both planes and all turbines.

Another element of our environment is how the wind speed changes over time. The initial mean freestream wind speed (i.e. the wind speed outside of the wind farm, before it is influenced by the wake effect), $\|\bar{U}\|_\infty$ is randomly selected from a uniform distribution of the integers between 8 and 16 m/s, inclusive. At each time-step thereafter, there is a probability of $\frac{p_{\delta u}}{2} = 0.05$ that the mean wind speed will increase by $\delta\|u\| = 0.5$ m/s and a probability of $\frac{p_{\delta u}}{2} = 0.05$ that the mean wind speed will decrease by $\delta\|u\| = 0.5$ m/s. The mean wind speed is saturated between the limits of 8 and 16 m/s. Gaussian noise with zero mean and a variance of $\delta\|u\| = 0.5$ m/s is then added to the mean value to simulate stochastic turbulence effects.

Finally, to simulate a realistic environment, we account for how the *direction* of the wind changes in a **FOWF!** (**FOWF!**) over time. The initial mean freestream wind direction (i.e. the wind direction outside of the wind farm, before it is influenced by the wake effect), $\angle U_\infty$ is randomly selected from a uniform distribution of the integers between 250° and 290°, inclusive, where 270° corresponds to wind coming from west to east, directly incident on the turbines in their neutral yaw position. At each time-step thereafter, there is a probability of $\frac{p_{\delta u}}{2} = 0.05$ that the mean wind direction will increase by $\delta\angle u = 5°$ and a probability of $\frac{p_{\delta u}}{2} = 0.05$ that the mean wind direction will decrease by $\delta\angle u = 5°$. The mean wind direction is saturated between the limits of 250° and 290°. Gaussian noise with zero mean and a variance

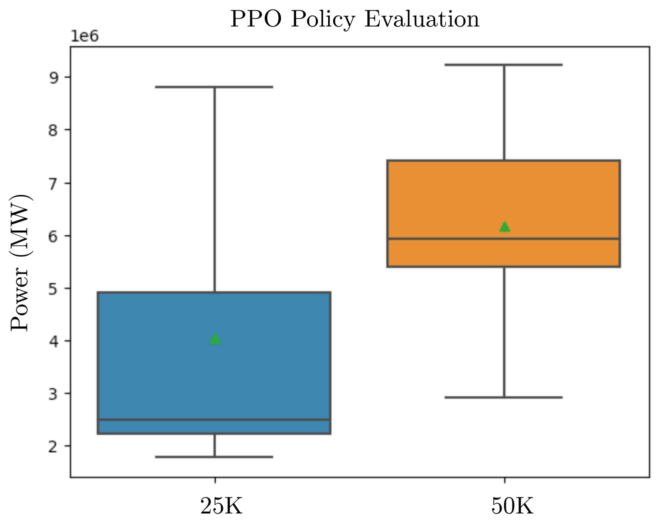

Fig. 3: Distribution of PPO trained for 25,000 time steps (left) and PPO trained for 50,000 time steps (right). Means are marked with a green triangle. We observe less variation when training for more time steps and a higher mean episode reward.

of $\delta\angle u = 5°$ is then added to the mean value to simulate stochastic turbulence effects.

J. *Single- vs. Multi-Agent*

We considered and tested multiple 'agent structures' in this project:

a) Single Agent: The *single-agent* case involves training a single agent to learn the optimal set-points for all $2T$ control variables (the axial induction factor and yaw angle for each of T turbines). On one hand, this is simple to implement, but on the other hand can result in many states and control variables, leading to a very complex problem.

b) Per-Turbine Agent: The *per-turbine multi-agent* case involves training a separate agent for each turbine to learn the 2 optimal set-points for that turbine. While this means that each agent only has 2 control variables to consider, T times as many agents need to be trained and each agent still requires an observation space consisting of every turbine's location and control set-points. This means that the per-turbine multi-agent is less memory efficient than the single-agent case.

c) Neighborhood Agent: The *neighborhood multi-agent* case involves dividing the wind farm into 'neighborhoods' of $\tilde{T} < T$ turbines and assigning an agent to control all of the set-points for the turbines in that neighborhood. This is effectively a compromise between the single-agent and per-turbine multi-agent cases. For this approach, the selection of the neighborhoods may prove to be a critical decision. We propose quantifying the influence of each turbine on the others with a graph structure and dividing the wind farm based on this.

Quantifying the relative advantages and disadvantages of the different approaches is left for further work. A useful comparative metric could be the total training time and computational resources required by each method to achieve the same mean episode reward.

K. *Utilizing Supercomputer Resources*

All experiments were done using the Alpine High Performance Computing Cluster hosted on CU RC Computing. Model training was performed on a single NVIDIA A100 GPU, 32 CPUs, and CUDA 11.3 using Python 3.9.

IV. EXPERIMENTS

Prior to simulating the RL environment, we first generate 500 freestream turbulent wind speed and direction time-series (see Sec. III of 24 hours each from which we can derive achievable power reference signals. We then conduct a number of experiments of varying complexity to test our approach:

1) considering a static wind-farm environment (in which there is no delay between when a change in wind or control set-points occurs upstream and the resulting wake has propagated downstream), constant power-reference and online status of wind turbines, and the reward function includes only the power-tracking error
2) same as experiment 1, with a full reward function (i.e. including the power tracking, rotor thrust attenuation and yaw travel attenuation terms)
3) considering a static wind-farm environment, full reward function, and time-varying power-reference and online statuses
4) considering a dynamic wind-farm environment (in which there is a delay between when upstream changes occur and resulting wakes propagate to downstream locations according to Taylor's frozen wake hypothesis), full reward function, and time-varying power-reference and online statuses

A. *Multi-Agent RL*

For our multi-agent approach to wind farm control, we implemented QMIX in the RLLIB [18] framework. We use a batch size of 600 for all experiments, a learning rate of .0001 using the Adam [20] optimizer, a discount factor (i.e., γ) of 0.99, and decay ϵ from 1 to 0 over 10,000 timesteps for choosing actions using ϵ-greedy. Due to existing issues with evaluating multi-agent algorithms in RLLIB, we use the training reward as a proxy to compare against PPO.

B. *Single-Agent RL*

For our single-agent approach to wind farm control, we implemented PPO in the Stable Baselines [19] framework. For all experiments we use a batch size of 64, a γ of 0.99, no ϵ-greedy, and a learning rate of .0003 with the Adam optimizer. For evaluation, we take the average episode reward over 5 episodes using Equation 16. Due to time and resource constraints, we only run PPO for 25,000 and 50,000 time steps to compare trends across the same algorithm.

C. Baseline Firm Power-Tracking Control

We compare results from the proposed RL-based approach to a baseline controls algorithm. This involves generating a lookup table (LUT) of optimal yaw angles for a grid of freestream wind-speeds and directions via least squares programming, and choose axial induction factors based on the $C_{p,i}$ curve for each turbine to achieve $\frac{1}{|\mathcal{T}|}$th of the farm-level power reference at each time-step, P_{ref}, i.e. $\frac{1}{|\mathcal{T}|}P_{\text{ref}} = C^{\star}_{p,i}\frac{1}{2}\pi R^2 \rho U_i^3 \Rightarrow C^{\star}_{p,i} = \frac{\frac{1}{|\mathcal{T}|}P_{\text{ref}}}{\frac{1}{2}\pi R^2 \rho U_i^3}$, where can find $a_i^{\star}$ by finding the roots of the third-order polynomial, $C^{\star}_{p,i} = 4a_i^{\star}(\cos\gamma_i - a_i^{\star})^2$.

V. Results

A. Results

We now evaluate both single-agent and multi-agent RL algorithms compared to baseline control algorithms. We show results in Table II.

a) Baseline.: We consider two baselines for optimal control of a wind farm. First, we consider the naïve baseline of all turbines having a yaw angle of 0 and an axial induction factor of 0.33. We call this *naïve-baseline*. The baseline controller in this work is based on an optimization routine provided with FLORIS [15]. This routine uses the Sequential Least Squares Programming algorithm to iterate through yaw angles and axial induction factors 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 [21]. We call this *control-baseline*.

b) Single-Agent RL.: We provide results for the best performing PPO model evaluated on our wind farm environment. Specifically, we evaluate two different PPO models trained for varying amount of time steps.

c) Multi-Agent RL.: We provide *training* results from QMIX to supplement the issues with multi-agent evaluation in RLLIB.

d) Evaluation Metric.: We compare the mean episode reward (MER, i.e., Equation **??**) across 5 episodes with a fixed random seed. Specifically, we measure success as:

$$r(\mathcal{E}) = \frac{1}{|\mathcal{E}|}\sum_{E\in\mathcal{E}} R_{total}(E) \tag{16}$$

where $\mathcal{E}$ is the set of all episodes for evaluation. In theory, Equation 16 is in the range $[0, \inf)$, indicating that a higher reward (i.e., power generated) signifies better algorithm performance.

B. Analysis with Respect to Algorithm Choice

First, we observe that QMIX has a significantly higher MER than any other algorithm. We attribute this to potential bugs in the implementation of QMIX as it is infeasible for our turbines to generate that much power. When examining correctly implemented algorithms, we observe that the naïve baseline outperforms the optimized baseline by generating 10 more MegaWatts on average. A potential reason for this is that chosen optimizer (i.e., sequential least squares) did not

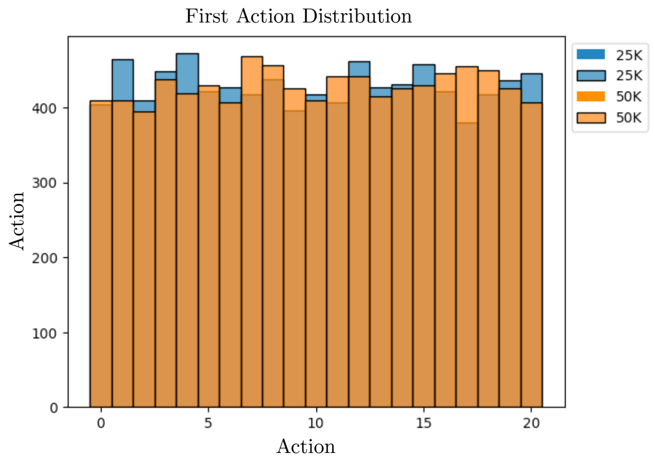

Fig. 4: Distribution of first action chosen for PPO trained for 25,000 time steps and 50,000 time steps, respectively, across 1,000 environment initializations. Overall, we observe uniform action choices across both models.

lead to a sufficient solution. We observe that PPO performs poorly compared to the baseline regardless of training time. Specifically, we observe ~ 47 more MegaWatts when using the naïve baseline compared to PPO trained for 50,000 time steps. A potential reason for this is a single agent is not sufficient to control all the turbines, thus our system does learn useful information while training.

C. Analysis with Respect to Training Time

We compare the effect of training PPO for 25,000 time steps (i.e., *PPO 25K*) and training for 50,000 time steps (i.e., *PPO 50K*). Both PPO implementations have the same hyperparameters and use the same optimizer, with training time being the only dependent variable.

When examining the performance of PPO in Table II, we observe a higher mean episode reward (MER) when training for a longer amount of time. Specifically, we observe a 2.13KW increase when training for 50,000 time steps compared to training for 25,000 time steps. This matches our intuition that by allowing RL agents to gain more experience within an environment, algorithms are able to generalize better and extract a higher reward. However, despite doubling the amount of training steps, we only observe a small increase in the average reward in addition to both models drastically under performing compared to both baseline approaches.

Next we examine trends when evaluating PPO for 5 episodes, demonstrated in Figure 2. First we observe that PPO 25K only outperforms PPO 50K in one episode, producing roughly 4 more mega-wattss (MWs) than PPO 50K at episode 1. For all other episodes, PPO 50K consistently generates a higher reward than PPO 50K, indicating that the additional training time is beneficial. We also empirically observe in Figure 2 that the PPO 50k has less variation in it's rewards, which matches our observations in Figure 3.

Finally, we examine the distribution of episode rewards during validation for both PPO algorithms. In Figure 3, we

Method	Naïve Baseline	Control Baseline	QMIX*	PPO 25K	PPO 50K
MER (MW)	**53.88**	43.91	25885.29	4.04	6.17

TABLE II: Mean Episode Reward (MER) across 5 episodes measured in MWs. When observing stable algorithms (i.e., PPO), we observe a higher MER when training for longer. *We observe a significantly higher reward when using QMIX but attribute this to bugs in the RLLIB implementation. The naive baseline outperforms all other algorithms.

observe a tighter distribution when training for a longer amount of time (i.e., 50,000 time steps) compared to training for a shorter amount of time. Moreover, we observe a slightly lower upper bound on the episode reward when training for 25,000 time steps (8.8¡W) whereas we observe a higher upper bound (9.2 giga-watts (GW)) when training for longer. We hypothesize the extra training time allows for the model to learn a slightly better policy, despite both algorithms learned approximately uniform policies as shown in Figure 4. The tighter distribution potentially indicates that even if a model is not learning useful information, training for longer can help fit a tighter distribution within episodic rewards.

D. Analysis with Respect to First Action

We now analyze the first action taken by each PPO model over 1000 first actions within the same environment, shown in Figure 4. Overall, we observe approximately uniform distributions between both models. There is slight variation in the actions chosen, but a single action does not appear to dominate in either scenario. This provides further evidence that our PPO models do not learn useful information during training and instead learn a uniform random policy. This is a potential reason for the poor performance compared to our baseline control methods. We hypothesize that a non-uniform policy would yield significantly better results, potentially closing the gap between RL methods and our baseline.

VI. CONCLUSION

In this work we present initial findings for the use of RL in wind-farm environments for the purposes of power tracking with both yaw angle and axial induction factor.

REFERENCES

[1] Sjoerd Boersma et al. “A tutorial on control-oriented modeling and control of wind farms”. In: *2017 American control conference (ACC)*. IEEE. 2017, pp. 1–18.

[2] Mamoru Futakuchi, Satoshi Takayama, and Atsushi Ishigame. “Scheduled operation of wind farm with battery system using deep reinforcement learning”. In: *IEEJ Transactions on Electrical and Electronic Engineering* 16.5 (2021), pp. 687–695.

[3] JJ Yang et al. “A deep reinforcement learning method for managing wind farm uncertainties through energy storage system control and external reserve purchasing”. In: *International Journal of Electrical Power & Energy Systems* 119 (2020), p. 105928.

[4] Aitor Saenz-Aguirre et al. “Artificial neural network based reinforcement learning for wind turbine yaw control”. In: *Energies* 12.3 (2019), p. 436.

[5] Huan Zhao et al. “Cooperative wind farm control with deep reinforcement learning and knowledge-assisted learning”. In: *IEEE Transactions on Industrial Informatics* 16.11 (2020), pp. 6912–6921.

[6] Hongyang Dong, Jincheng Zhang, and Xiaowei Zhao. “Intelligent wind farm control via deep reinforcement learning and high-fidelity simulations”. In: *Applied Energy* 292 (2021), p. 116928.

[7] Hongyang Dong and Xiaowei Zhao. “Wind-farm power tracking via preview-based robust reinforcement learning”. In: *IEEE Transactions on Industrial Informatics* 18.3 (2021), pp. 1706–1715.

[8] Sanjana Vijayshankar et al. “Deep reinforcement learning for automatic generation control of wind farms”. In: *2021 American Control Conference (ACC)*. IEEE. 2021, pp. 1796–1802.

[9] Paul Stanfel et al. “A distributed reinforcement learning yaw control approach for wind farm energy capture maximization”. In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 4065–4070.

[10] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[11] David Silver et al. “Deterministic policy gradient algorithms”. In: *International conference on machine learning*. Pmlr. 2014, pp. 387–395.

[12] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[13] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[14] Justin K Terry et al. “Revisiting parameter sharing in multi-agent deep reinforcement learning”. In: *arXiv preprint arXiv:2005.13625* (2020).

[15] 2019. URL: https://github.com/NREL/floris.

[16] *PyTorch*. URL: https://pytorch.org/docs/stable/index.html.

[17] URL: https://beta.openai.com/docs/.

[18] Ray-Project. *RLlib: Industry-Grade Reinforcement Learning with TF and Torch*. URL: https://github.com/ray-project/ray/tree/master/rllib.

[19] Antonin Raffin et al. “Stable-Baselines3: Reliable Reinforcement Learning Implementations”. In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[20] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[21] Paul Fleming et al. “Initial results from a field campaign of wake steering applied at a commercial wind

farm–Part 1". In: *Wind Energy Science* 4.2 (2019), pp. 273–285.