TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK
PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG
PROF. DR. STEFAN GUMHOLD

Großer Beleg

Reinforcement-Learning Windturbine Controller

Nico Westerbeck
(Mat.-Nr.: 3951488)

Betreuer: Dr. Dmitrij Schlesinger

Dresden, 12. November 2019

Aufgabenstellung

0.1 Background:

Here at the HFI Experimental fluid mechanics group, we have developed an open source project called QBlade. QBlade is a simulation tool used for testing wind turbines in the hostile environment that they normally operate. We normally tackle problems of aerodynamic or structural optimization but we have also a research focus on the development of the control systems of the wind turbines. We currently have a research effort looking at developing cluster-based controllers building on the work of Professor Bernd Noack who is a guest professor at our group. In the last year or so (Nair, A. G., Yeh, C.- A., Kaiser, E., Noack, B. R., Brunton, S. L., & Taira, K. (2018). Cluster-based feedback control of turbulent post-stall separated flows. Journal of Physics Fluid Dynamics, (M), 1-32. Retrieved from http://arxiv.org/abs/1809.07220). AI projects such as openAI have enabled he rapid development of neural network in the field of control using reinforcement learning. The goal of this project is to use QBlade as a wind turbine simulator and attempt to control the pitch and rotor speed in a way that doesn't cause the wind turbine to shatter but instead to yield energy, i.e. reward and death condition. This first stage of work should be considered as exploratory but will hopefully open up avenues of controlling active flow control elements such as flaps.

0.2 Tasks

The major tasks of the project are as follows:

- Build up and interface between QBlade and python the model code so that an external code can run as a controller within a QBlade simulation.
- Gain a rough understanding of the mechanics of wind turbines and their controllers.
- Research reinforcement learning methods suitable for use as a windturbine controller and perform a literature review on these approaches.
- Create a reinforcement learning agent which uses the Qblade interface for controllers to control a windturbine.
  - Inputs to the agent could be defined by the standartized controller input format to Nordex turbines, which consists of 39 real-valued sensor-inputs. However, initial tests can be conducted with whichever inputs are easiest to tackle. If required, further hidden state from the simulation can be exported to enrich data quality. If aiming for industrial quality, more inputs

and also sensor faults could be optionally incorporated.

  - Outputs are in a minimum version pitch angles for the 3 blades and turbine torque. Optionally the agent should be able to control active element such as flaps on the blades.

- Optimize the agent to deliver maximum energy yield.
- Optimize under respect of certain boundary conditions (maximum pitch acceleration, maximum power, maximum blade load, blade touching the tower) and optionally other boundary conditions like long term turbine wear.
- If necessary for the training process, scale the simulation to run at a larger scale.
- Implement and attempt to get the agent to perform something close to sensible control of the wind turbine. Optionally evaluate the results against existing controllers and try to outperform them.
- Optionally, create a conference paper, poster or blog post etc.. on the results.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Arbeit zum Thema:

Reinforcement-Learning Windturbine Controller

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 12. November 2019

Nico Westerbeck

Inhaltsverzeichnis

1 Abstract

Small

2 Intro

Recent advancements in reinforcment learning have managed to tackle more and more complex problems, like StarCraft or Go. The range of topics, RL is applicable to, increases. However, so far, there hasn't been an effort to control a windturbine with a reinforcement learning algorithm. Current state of the art linear controllers are performing well in maintaining turbine control, and the optimization margin for these is small, as they achieve the theoretical maximum energy output up to rated windspeed and manage to keep the turbine intact above rated windspeeds.

Trying it nontheless can still provide for some foundational knowledge for future developments. Park control becomes a bigger topic in wind-energy, where controlling all windturbines of a park together can increase overall energy yield or protect windturbines further down the windstream from wakes by turbines up front. C

3 Background

This section aims at facilitating the background knowledge necessary for understanding this paper. We fill first introduce windturbines and common terms around that

3.1 Windturbine control

There are two mayor types of wind turbine designs, Horizontal Axis Wind Turbine (HAWT) and Vertical Axis Wind Turbine (VAWT), which differ by their rotation axis. In this work, we will only look at HAWT and all mentions of windturbines mean HAWT type turbines. Such a turbine is made up by 3 big components, a tower on which a nacelle is rested which itself has a rotor in front. The joint between tower and nacelle allows for rotation to turn the rotor into the wind. Except for recent ideas [HLD], the best value for the rotational direction was to always yaw the rotor directly into the wind, and for the sake of simplicity we will omit this control parameter from our simulation and leave it at 0 degrees with wind facing straight onto the rotor. In the nacelle there is, most importantly, an electrical generator, which can be given a specific torque. It will then slow down the rotor with that torque and at the same time generate energy proportional to rotor speed and torque. This torque is one of the two more important control parameters, together with blade pitch. The blades are designed to operate on maximal aerodynamic efficiency when they are pitched at 0 degrees, increasing the pitch angle will (except for possible slight improvements in the first few degrees) reduce aerodynamic efficiency of the rotor.

As described in [BJSB, sec 8.3], the normal controller design for windturbines divides the operation of a turbine into two parts, below rated and above rated. Rated is a windspeed, in which the turbine gives maximum energy on highest aerodynamic effiency. At this point, the generator runs on its maximum torque and the blades are pitched to the optimum angle of 0 degrees. At this point, the tip-speed-ratio, which is a good indicator for loads on the blades is at maximum too. Below that windspeed, the rotor generates less rotational torque than the maximum generator torque, thus the generator torque needs to be reduced to not slow the rotor down too much. Slowing the rotor down complicates the energy generation process, and though variable-speed wind turbines allow for some play, all windturbines are limited to a certain operational range. Fixed-speed wind turbines can only run on one single rotational speed and thus

need to change the torque more agressively. Above rated windspeed, the blades need to be pitched out to stop the rotor from spinning at faster speeds than what the turbine was designed for. The generator torque can be kept at maximum in this area, but to protect the turbine from damage, aerodynamic efficiency of the rotor is reduced by turning the blades along their long axis and thus moving their angle of attack against the incoming air into a less optimal range.

Additionally to this trivial control, optimizations to reduce vibrations and stress on the structures are implemented. The blades can be pitched individually and quickly enough to allow for different blade pitches during a single rotation of the rotor, which can be used to reduce the turbulence that hits the tower or to account for different windspeeds closer to the ground and further up in the air. Also, both generator torque and blade pitch can be used to counteract natural resonant frequencies of the structure, reducing material stress through extensive swinging of structural parts.

Usually this control is implemented by two PID controllers, which are hand-designed as described in [BJSB, sec 8.4]. Below rated windspeed, the controller for torque is active, above rated the one for pitch. The parameters to those controllers can be calculated according to the laws of control theory, and the resulting controllers work reasonably close to theoretical maximum. Still, in this work, we are trying to replace these hand-designed controllers with a reinforcement-learning algorithm.

3.2 QBlade

Our data source in this work is the open-source simulation tool QBlade [MWP$^+$] developed at TU Berlin. QBlade is an accessible and performant tool with the primary purpose of designing and simulating wind turbines in a graphical user interface. Its simulation results are on par with current state-of-the-art proprietary simulation tools and it yields good computational efficiency. It uses an algorithm which approximates the 3D blade structures with 2D structures and corrects the results through several error terms. Though a full CFD simulation would yield higher accuracy, our computational resources don't allow for that. To be used in reinforcement learning, we designed an interface, over which the QBlade simulation can be embedded into an external environment. Most machine-learning frameworks are written in python, so we decided to compile QBlade into library format and expose the most fundamental functions to allow it to communicate with any programming language that can load libraries. As python has a ctypes interface to load c-code, we could link a python agent to the QBlade C++ environment.

QBlade allows different simulation scenarios, for our testing we decided to only use the NREL 5MW [JBMS] turbine with the default structural model and using all implemented correction mechanisms to achieve the most realistic data possible. As reference data to this turbine is easily available and publicly

published by NREL, this allows us for good cross-validation.

The interface is made of 7 functions, which allow for loading a project, resetting the simulation, setting controller inputs and getting environment measurements and advancing the simulation a timestep. The observations returned in getControlVars match those that are visible to standart industry controllers, which in turn are modelled after what can be measured on a real windturbine. Concretely, we measure:

```
* rotational speed [rad/s]
* power [kW]
* wind velocity [m/s]
* yaw angle [deg]
* pitch blade 1 [deg]
* pitch blade 2 [deg]
* pitch blade 3 [deg]
* tower top bending local x [Nm]
* tower top bending local y [Nm]
* tower top bending local z [Nm]
* out-of-plane bending blade 1 [Nm]
* out-of-plane bending blade 2 [Nm]
* out-of-plane bending blade 3 [Nm]
* in-plane bending blade 1 [Nm]
* in-plane bending blade 2 [Nm]
* in-plane bending blade 3 [Nm]
* out-of-plane tip deflection blade 1 [m]
* out-of-plane tip deflection blade 2 [m]
* out-of-plane tip deflection blade 3 [m]
* in-plane tip deflection blade 1 [m]
* in-plane tip deflection blade 2 [m]
* in-plane tip deflection blade 3 [m]
* current time [s]
```

Because it has no semantic meaning, we mask away current time from our reinforcement learning agent. The inputs we can give to the turbine are:

```
* generator torque [Nm]
* yaw angle [deg]
```

* pitch blade 1 [deg]
* pitch blade 2 [deg]
* pitch blade 3 [deg]

As mentioned before, we set the yaw angle to always 0 degrees, as in reality controlling the orientation of the nacelle is trivial. The generator torque on our NREL 5MW turbine has a sensible range of 0 to 4×10^6 NmBecause changing torque from zero to maximum in one timestep is not realistic for a real turbine, we restrict it to move a maximum of 3×10^5 Nm per one timestep of 0.1 seconds. If our agent chooses a value outside of that area, we set it to the closest value inside of that area. Blades can be pitched between 0 and 90 degrees, 0 being not pitched out at all and operating at maximum efficiency and 90 resulting in no aerodynamic torque from the rotor whatsoever. A sensible pitch motor can only turn the blades at a limited speed, we assume a limit of 5 degrees per second and again set controller inputs to a sensible value inside that. The simulation theoretically accepts a full pitch change in one timestep, however as inertia on the blade is so high for such a change, the blades break instantly and the rest of the simulation needs to be reset.

QBlade simulates blade damage, also high rotational speeds cause blades to fall off. QBlade is not validated to realistically simulate these extreme conditions and thus posed a problem to our controllers. In the course of our experiments, we decided to reset the simulation at an rotational speed of 4 rad/s, as our turbine operates at ca 0.8 rad/s. Also, high generator torque inputs cause the simulation to rotate the rotor backwards with a negative energy yield, effectively creating a multimillion-dollar leafblower. As this is neither a realistic scenario, we also reset the simulation at -0.5 rad/s.

3.3 Reinforcement learning

This section should aim to give a brief overview into Reinforcement Learning assumptions and derive briefly a couple of methods until in the end, we will describe Deep Deterministic Policy Gradients (DDPG), the method we utilized to control our virtual windturbine.

3.3.1 Environment assumptions

At first, we need to do some assumptions on our environment. The concept environment encapsulates the reality or simulation that the reinforcement learning algorithm works in. This could be an actual cyber-physical system or a simulation. The first assumption to this environment is that this system operates over time and we can discretize time into timesteps. Our environment here is the simulated wind turbine, but it

could be a computer game [LHP+], a physics simulation[BCP+] or even an actual existing wind turbine [KJT]. In every timestep, the environment supplies us with a state, also sometimes called observation, and a scalar reward. The observed state doesn't have to represent the entirety of the state of the simulation, in fact it usually is only a small subset. For our wind turbine example, the observed state is limited to what can be measured with sensors on a wind turbine, in an arcade game it could be the pixel output or in a physical simulation it could be some key values in the simulation. The reward is a scalar value which judges the current state of the simulation. A higher reward means the current state is better than that of a lower reward. Sometimes, the reward function is obvious from the system, as in an arcade game it would surely be the score or in chess it could be 1 for win, -1 for lose and 0 for not decided yet. However in our paper, we will dedicate a section on how we designed our reward function. In general reinforcement learning theory, it is assumed to exist and be supplied by the environment. In every timestep, the agent supplies an action in the form of a tensor. This action could be button presses in an arcade game or blade pitch in a windturbine simulation. The mission of the agent is to pick actions which maximize cumulative reward. In other words, it should steer the environment to achieve best performance. Additionally to the state and reward, an environment can optionally send out a done signal, which indicates a state in which further simulation is not possible and the environment wants to be reset to an initial state. In a chess game, this would be a loss or a win, and with a windturbine that could for example be fatal structural damage.

The algorithms we looked at further assume that the state progressions can be modelled as a Markov-Chain, as in that what happens at timestep t+1 is only dependent on the hidden state at timestep t, not also on for example t-10. As the hidden state could for example be the full RAM of the simulation, we can safely assume that a simulation only calculates the next timestep on what is in the RAM at the current timestep. Also, Markov-Chains allow for several states as an outcome of any state, and that we can model the transition between these state with a probablistic model.

3.3.2 Q-Learning

At first we will present a very basic algorithm for a machine learning agent, which is based on the basic Bellman-Equation [Bel]

$$Q(s,a) = r + \max_{a'} Q(s',a') \tag{3.1}$$

3.3.3 PG

3.3.4 DDPG

3.4 RL on windturbines

4 Early experimentation

At the start of our experimentation, we took off with a well-known problem and a proven implementation. We used the OpenAI-Pendulum task, in which a pendulum in gravity needs to be held upright by applying a force to it. OpenAI Gyms generally serve as benchmarks for reinforcement learning algorithms and most foundational papers and articles evaluate their solutions based on how well they perform in OpenAI Gyms. Starting with a reference-implementation in pytorch by [Dee], we experimented with the pendulum environment. We observed that the algorithm got stuck in a local optimum, where it rotates the pendulum instead of holding it upright, for long periods of the training process. Only with high noise, we could get it to hold a pendulum upright. We also tried to run the HalfCheetah task, where a two-dimensional dog-like being should be brought to running and moving its joints, but the results were more a crawling and shaking towards the goal. However, we hoped that the implementation might yield results on our windturbine environment already and exchanged the OpenAI gym for qblade.

The first runs looked marvellous, perfectly holding rated speed, until we found out that those perfect values were prescribed by a setting in the qblade project file, and disabling this setting marked the beginning of our actual experimentation phase.

5 Experimentation

On our first experimentation steps, we did not yet have experience with how the qblade simulation behaves with a random or not well designed controller. Our observation was, that the simulation was resting between two different states, which we want to call Forward Maniac Mode (FMM) and Backward Maniac Mode (BMM).

FMM is characterized by a rotational speed of ca 5 times rated speed, at 4.3 rad/s. At this speed, for some reason, the rotor will not speed up anymore. However, severe vibrations and sometimes the case of a disjointed blade happen. Especially in the case of a blade flying off, deflection values skyrocket up to several orders of magnitude higher than values seen in normal operation, and with one or more missing blades, the rotor will quickly stop rotating. This state is usually reached by a too low generator torque while not pitching out blades and sometimes stays stable for several thousand steps until a blade falls off.

BMM is characterized by a negative rotational speed, up to -3.8 rad/s. This can be reached by setting a higher generator torque than the rotor torque which is produced by aerodynamic lift on the blades. Also in this state, extreme vibrations and sometimes blades flying off can be observed, and like FMM this state is stable until a blade falls off.

5.1 Designing reward functions

As qblade doesn't provide a built-in reward function, we needed to craft our own reward function based on the observed state. We considered several possible variants. We will first introduce all of them and give a more detailed explanation later

- **Rated speed** Hold a rated speed. Observe current rotational speed s_{rot} and with given rated speed s^*_{rot} calculate reward $r(s) = -\left|\frac{s_{rot}-s^*_{rot}}{s^*_{rot}}\right|$
- **Rated power** Hold a rated power. Observe current power s_{pow} and with given rated power s^*_{pow} calculate reward $r(s) = -\left|\frac{s_{pow}-s^*_{pow}}{s^*_{pow}}\right|$
- **Maximal power** Generate the maximum power possible. Observe current power s_{pow} and with given normalization constant c calculate $r(s) = s_{pow} * c$

- **Penalize current stress** Penalize high blade bending. Observe current bending s_{bend} and with given normalization constant c and another reward function $r(s)$ calculate new reward $r'(s) = r(s) - c * s_{bend}$
- **Penalize accumulated stress** Penalize accumulated structural stress. With $x = rainflow(s_0...s_t$ being the rainflow table and $p(x)$ being a function that maps from rainflow outputs to a scalar penalty, calculate $r'(s_0...s_t) = r(s_t) - p(rainflow(s_0...s_t))$

Rated speed is the easiest of the rewards, as holding a rated speed can be achieved by either pitch or torque or a combination of the two. So both a high pitch and a high torque will stop it from running faster than it should. We used this reward for our first working version. **Rated power** is a bit more difficult. For reaching rated power, a certain aerodynamic torque is required, so pitching the blades out completely won't deliver rated power. However especially if setting an artificial rated power lower than what the turbine was built for, there is a certain play in how strongly to use blade pitch and how much needs to be done through torque. **Maximal power** is an intuitive reward, but might yield unrealistic results. This rewards high power regardless of the stress that is induced to the turbine, so a good policy might spin the turbine at high speeds beyond any turbine specification and then apply maximum torque. So this reward makes more sense when combined with a stress penalty. **Penalize current stress** is the easiest stress penalty, where a penalty proportional to the current bending of the structure can be used. As there are several bending modes on most of the components of the windturbine, a combination or selection of modes and components should be taken before. We decided to limit us to out-of-plane bending of the blade tips. This is a simple variant which leaves out bending in the middle of the blades, bending in the rotational plane, vibration or bending of the tower and torsional stress on the shaft. **Penalize accumulated stress** is a more complete, but time dependent variant. Rainflow Counting (RFC) is a commonly used method for calculating structural fatigue in windturbines [BW]. RFC takes in a bending signal over time and returns a table with amplitudes and their frequency, as in how often the structure swung how far. It filters out some higher-frequency swinging in between two lower frequency swings and thus also works for structures that swing in more than one frequency. To use it as a penalty, we needed to create a function that maps from that table to a scalar penalty. This penalty accumulates past stress, so it increases over time. We suspect this penalty to be hard to learn, as it reacts slowly to changes and a once given penalty will never be lifted again.

Most of our experimentation we did with rewarding rated speed.

5.2 Aiding exploration

As in the beginning we observed our algorithm to get stuck in either FMM or BMM, we wanted to improve exploration. Exploration is a common problem in reinforcement learning, as algorithms tend to stay in local maxima.

5.2.1 Action Noise

We first added a gaussian noise to our actions and after not seeing any improvemnts, switched to an Ornstein-Uhlenbeck process [UO] as recommended in the DDPG paper [LHP$^+$, formula (7)]. Additionally to the DDPG paper, we let the sigma parameter decay over time, so that the noise gets less when training has proceeded.

5.2.2 Random exploration

The TD3-paper [FHM] utilizes a random exploration phase, in which a completely random policy [MB] initially explores the environment, before the actual agent switches in. We implemented the random walk at first through gaussian noise and then through an Ornstein-Uhlenbeck process. The later version creates less vibration through less excessive changes in control parameters. As we found that both random policies rarely explore a sensible operational range, we later added an expert PID controller, which implements torque control, and combined that with our Ornstein-Uhlenbeck process. We set the parameters to that controller by experimentation.

5.2.3 Parameter noise

As we still didn't see good results, we tried adding parameter-space noise [PHD$^+$] to the actor function, as the paper promised better exploration. We could not observe any improvements and deactivated parameter space noise for the working version.

5.3 Zero Gradients

All our efforts to aid exploration by adding noise did not yield better exploration, so we had a look at our gradients and found that all our actor gradients are zero. The critic had gradients, but only in the later levels of the net. Eventually, we solved this problem by both simplyfing the architecture and utilizing normalization.

5.3.1 Simplifying the architecture

We tackled this problem by at first reducing the number and size of layers from the example implementation to one fully connected layer of 64 neurons in the critic and one fully connected layer of 32 neurons in the actor. The reference-implementation of [Dee] utilized 3 levels in the critic with 1024, 512 and 300 fully connected neurons and in the actor 2 levels of 512 and 128 fully connected neurons. Reducing complexity partially tackled the problem of zero gradients, we could then observe small gradients which however vanished over time.

5.3.2 Normalization

More effectively, we added data normalization. We normalized the data so states, actions and reward always stayed between 1 and -1. This finally fixed our problem of zero gradients and we could observe normal learning. We recommend this technique as a default technique for every reinforcement learning algorithm, as we can not imagine any drawbacks of normalizing input data. As in our simulation, where our state reflects measurements from very different parts of the simulation, some values in the state-array were 8 orders of magnitude different from others. As most of the high-magnitude values are vibrations, the net could not create a link between these and the state-action, while low-magnitude values like rotational speed were not considered due to their small absolute values.

5.4 High action gradients

We observed that our controller delivers high action gradients and reacts strongly between timesteps. This creates vibrations in the structure and especially high blade pitch changes lead to blades breaking or falling off. We tried different methods to circumvent this.

5.4.1 Gradient actionspace

At first, we implemented gradient actionspaces, where the controller output is not an absolute pitch or torque value but a difference to the last output, starting with everything set to 0. This way, we could clip the action gradients by reducing the gradient action space, so that in each timestep, only a fraction of the actual action space could be traversed. If the controller would output a positive change at maximum pitch/torque or a negative change at minimum pitch, this would be ignored. This did effectively limit action gradients, but also stopped us from getting sensible results.

5.4.2 Feeding past timesteps

Another idea was that we feed the last n timesteps concatenated to the current observations. We hoped that this way, the agent could derive its own gradients internally. The result of this were high and very noisy q-losses and no convergence whatsoever. Though we still had gradients, we suspect that our relu-activations might have drifted far off to the low end, and thus could not recover.

5.4.3 Clipping action gradients

We eventually solved the problem with the high action gradients by still letting the controller output absolute actions across the entire action spaces, but if gradients exceed a certain threshold, the actual action taken is set to the nearest value with sensible gradients. Though the actions returned from the policy are still wildly unstable, they at least didn't result in death of the turbine anymore and still we didn't hinder convergence through our gradient actionspace.

5.5 Improvements without sense

5.5.1 Prioritized experience replay

5.5.2 Data augmentation

We still had some parts of the code left where there wasn't any noise involved, so we decided to add some to the training process. Data-augmentation with noise is a common technique when using neural networks [Bis, p.347] to reduce overfitting, so we expected it to also make our model generalize better. There have been some experiments with data augmentation in reinforcement learning [**?**], but it is not often mentioned on tutorials or blog posts. We implemented by adding a small noise term to the sampled states, actions and next states (s, a, s') when performing experience replay. We didn't really see any indication of bad generalization, but as implementing it wasn't a big issue and normally doesn't bring any disadvantages, we still added this feature and set the noise level to a very small value.

5.6 Exploding Q-Loss

We observed that our Q-Loss, especially after a death, explodes into e15 magnitude values, while most of the times it stayed around e2 magnitude. Instable Q-Losses are a general problem with DQN and DDPG, and some literature [FHM] proposes to use two Q networks, of which only the minimum is used

for training. We could already greatly improve our results by using huber-loss [**?**] for the critic instead of mean squared error loss as described in DDPG paper. Also, we believe that adding double Q networks such as in [FHM] would also solve this problem.

5.7 Automatized hyperparameter search

6 Evaluation

7 Future work

8 Conclusion

Literaturverzeichnis

[BCP+] BROCKMAN, Greg ; CHEUNG, Vicki ; PETTERSSON, Ludwig ; SCHNEIDER, Jonas ; SCHULMAN, John ; TANG, Jie ; ZAREMBA, Wojciech: OpenAI Gym.

[Bel] BELLMAN, Richard: The Theory of Dynamic Programming. 60, Nr. 6, S. 503–516. – ISSN 0002–9904

[Bis] BISHOP, Christopher M.: *Neural Networks for Pattern Recognition*. Clarendon Press ; Oxford University Press. – ISBN 978–0–19–853849–3 978–0–19–853864–6

[BJSB] BURTON, Tony ; JENKINS, Nick ; SHARPE, David ; BOSSANYI, Ervin: *Wind Energy Handbook: Burton/Wind Energy Handbook*. John Wiley & Sons, Ltd. – ISBN 978–1–119–99271–4 978–0–470–69975–1

[BW] BERGLIND, J. J. B. ; WISNIEWSKI, Rafael: Fatigue Estimation Methods Comparison for Wind Turbine Control.

[Dee] *Deep Deterministic Policy Gradients Explained*

[FHM] FUJIMOTO, Scott ; VAN HOOF, Herke ; MEGER, David: Addressing Function Approximation Error in Actor-Critic Methods.

[HLD] HOWLAND, Michael F. ; LELE, Sanjiva K. ; DABIRI, John O.: Wind Farm Power Optimization through Wake Steering. 116, Nr. 29, S. 14495–14500. – ISSN 0027–8424, 1091–6490

[JBMS] JONKMAN, J. ; BUTTERFIELD, S. ; MUSIAL, W. ; SCOTT, G. *Definition of a 5-MW Reference Wind Turbine for Offshore System Development*

[KJT] KOLTER, J. Z. ; JACKOWSKI, Z. ; TEDRAKE, R.: Design, Analysis, and Learning Control of a Fully Actuated Micro Wind Turbine, IEEE. – ISBN 978–1–4577–1096–4 978–1–4577–1095–7 978–1–4577–1094–0 978–1–4673–2102–0, S. 2256–2263

[LHP+] LILLICRAP, Timothy P. ; HUNT, Jonathan J. ; PRITZEL, Alexander ; HEESS, Nicolas ; EREZ, Tom ; TASSA, Yuval ; SILVER, David ; WIERSTRA, Daan: CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING.

[MB] MOZER, Michael C. ; BACHRACH, Jonathan: Discovering the Structure of a Reactive Envi-

ronment by Exploration. 2, Nr. 4, S. 447–457. – ISSN 0899–7667, 1530–888X

[MWP+] MARTEN, D ; WENDLER, J ; PECHLIVANOGLOU, G ; NAYERI, C N. ; PASCHEREIT, C O.: QBLADE: AN OPEN SOURCE TOOL FOR DESIGN AND SIMULATION OF HORIZONTAL AND VERTICAL AXIS WIND TURBINES. 3, Nr. 3, S. 6

[PHD+] PLAPPERT, Matthias ; HOUTHOOFT, Rein ; DHARIWAL, Prafulla ; SIDOR, Szymon ; CHEN, Richard Y. ; CHEN, Xi ; ASFOUR, Tamim ; ABBEEL, Pieter ; ANDRYCHOWICZ, Marcin: Parameter Space Noise for Exploration.

[UO] UHLENBECK, G. E. ; ORNSTEIN, L. S.: On the Theory of the Brownian Motion. 36, Nr. 5, S. 823–841. – ISSN 0031–899X

Danksagung

Die Danksagung...

Erklärungen zum Urheberrecht

Hier soll jeder Autor die von ihm eingeholten Zustimmungen der Copyright-Besitzer angeben bzw. die in Web Press Rooms angegebenen generellen Konditionen seiner Text- und Bildübernahmen zitieren.