ROBUST DEEP REINFORCEMENT LEARNING FOR UNDERWATER NAVIGATION WITH UNKNOWN DISTURBANCES

Juan Parras, Santiago Zazo

Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
Madrid, Spain

ABSTRACT

We study an underwater navigation problem, where an Underwater Autonomous Vehicle must reach a target position in the presence of a disturbance that may be unknown. In order to deal with this problem, we make use of Deep Reinforcement Learning tools, and more concretely, we make use of robust control ideas, which allow training an agent in the presence of uncertainty. We propose a robust Proximal Policy Optimization agent and train it using simulations of an underwater medium: this agent shows an excellent performance when facing unknown disturbances, being able to approach the performance of the optimal agent which had an exact knowledge of the underwater disturbance.

***Index Terms*—** Robust Control, Deep Reinforcement Learning, Underwater Autonomous Vehicle, Proximal Policy Optimization, Underwater Disturbances

1. INTRODUCTION

The problem of autonomous underwater navigation has been traditionally a complex problem. First, because the water strongly attenuates the radio signals, and hence, it is not possible to use GPS underwater. And second, because the underwater medium is subject to disturbances that affect the Underwater Autonomous Vehicle (UAV). The first problem has been subject to an intense research, which has resulted in a set of methods designed to facilitate the underwater navigation, as a recent survey notes [1]. A family of algorithms used to address the underwater navigation problem is Reinforcement Learning (RL), a bio-inspired method for optimal control in which an agent learns by interacting with an environment through trial and error. The objective of the learning process is a set of trajectories that are optimal in some sense, such as time to reach a certain target or battery consumption, to mention some. Although RL was applied to the underwater navigation problem several years ago [2], [3], the recent advances in Deep Learning, and more concretely, on Deep RL (DRL), have facilitate a set of works that apply these methods to high dimensional underwater navigation problems, as can be seen in [4], [5], and the references therein.

However, most of the current works do not pay much attention to the disturbances, and actually many papers do not consider them, such as [3] or [4]. One exception is [2], which however, is limited to a single disturbance type and only considers discrete states and actions. But the role of disturbances should not be ignored: a recent work using DRL for underwater target search shows that the performance of the system worsens significantly when a disturbance is present [5]. Thus, it is important taking disturbances into account for underwater autonomous systems: this is our objective.

However, even though we may simulate the effect of a disturbance, actual disturbances on the underwater medium may significantly differ from the ones used in simulations. We may deal with this uncertainty using Bayesian methods in combination with RL, as in [6] and [7]. However, these methods rely on Gaussian Processes, which are computationally complex, although recent advances in Deep Learning may alleviate this [8]. We use a different approach, based on robust control ideas [9], [10], [11], [12] or [13]. In these methods, the agent is trained in an adversarial fashion, so that it is able to face unknown situations. We apply these ideas to the autonomous underwater navigation: our main contribution is developing an agent that is able to successfully cope with unknown underwater disturbances, which is a problem that has not been satisfactorily addressed in current literature yet. We compare to a simple baseline and an optimal agent that knows the actual disturbance. Our simulations show that the robust agent we propose is able to approach the performance of the optimal agent, thus, robust tools are a good approach to deal with disturbance uncertainty, which is a situation that arises in most real-world underwater navigation systems.

The rest of the paper goes as follows: Section 2 introduces our UAV model and the disturbances used in this work. Then, Section 3 introduces the necessary background on DRL and explains the algorithms we use. Section 4 presents our empirical results, which shows that our ideas are successful, and finally Section 5 draws some conclussions and proposes some possible future lines.

2. SETUP DESCRIPTION

We now present our UAV and disturbances models used in this work.

2.1. Underwater navigation model

We consider a UAV moving in a 2-D space (i.e., constant depth), where x and y denote the horizontal and vertical coordinates, and v_x and v_y represent the velocity components in x and y. The UAV control variable is the acceleration angle θ (i.e., the heading angle), and the time index is n. We consider that the disturbances affect the acceleration of the UAV and its effect is modelled by two terms $d_x(x, y)$ and $d_y(x, y)$. We model the friction as velocity dependent (i.e., a low velocity case, [14]) using a parameter k_f as in the Isotropic Rocket problem [15]. Hence, we consider that the velocities depend on the acceleration (i.e., the control variable), the disturbance and a limit due to friction (so that the velocity does not grow

The work is supported by the Spanish Ministry of Science and Innovation under the grant TEC2016-76038-C3-1-R (HERAKLES).

unbounded). Thus, our dynamic system is:

$$\begin{aligned} x_{n+1} &= x_n + \Delta \cdot v_{x,n} \\ y_{n+1} &= y_n + \Delta \cdot v_{y,n} \\ v_{x,n+1} &= v_{x,n} + \Delta \cdot (\cos(\theta_n) + d_x(x_n, y_n) - k_f \cdot v_{x,n}) \\ v_{y,n+1} &= v_{y,n} + \Delta \cdot (\sin(\theta_n) + d_y(x_n, y_n) - k_f \cdot v_{y,n}) \end{aligned}, \quad (1)$$

where Δ is the time step and $s_n = (x_n, y_n, v_{x,n}, v_{y,n})$ is the state of the system.

2.2. Disturbance models

We use three significant disturbances that appear in the underwater environment: swirls, currents, and constant fields. The constant field is a fixed disturbance modeled using two scalar parameters $a \in \mathbb{R}$ and $\alpha \in [0, 2 \cdot \pi)$, where the absolute value of a controls the strength of the field, and α its orientation as:

$$(d_x, d_y) = a \cdot \Big(\cos(\alpha), \quad \sin(\alpha) \Big). \quad (2)$$

A swirl vector field can be modeled using three scalar parameters $b \in \mathbb{R}$, $x_0 \in \mathbb{R}$ and $y_0 \in \mathbb{R}$ as:

$$(d_x, d_y) = \frac{b}{\sqrt{(x - x_0)^2 + (y - y_0)^2}} \cdot \Big(y - y_0, -x + x_0 \Big), \quad (3)$$

where x_0 and y_0 control the location of the vortex of the swirl, the absolute value of b controls the strength of the swirl and the sign of b controls whether the swirl rotates clock or counterclockwise.

In this work, for simplicity, we consider only horizontal and vertical currents. A horizontal current can be modeled using three scalar parameters $c \in \mathbb{R}$, $w \in \mathbb{R}$ and $y_0 \in \mathbb{R}$ as follows:

$$(d_x, d_y) = \left(c \cdot e^{-\frac{(y - y_0)^2}{w^2}}, \quad 0 \right), \quad (4)$$

where y_0 locates the maximum current strength y-coordinate, the absolute value of w indicates the breadth of the current, the absolute value of c controls the strength of the current and the sign of c controls whether the current direction is positive or negative. In an equivalent way, a vertical current can be modeled as follows:

$$(d_x, d_y) = \left(0, \quad c \cdot e^{-\frac{(x - x_0)^2}{w^2}} \right). \quad (5)$$

3. DEEP REINFORCEMENT LEARNING METHODS

We formulate our navigation problem as a discrete time Optimal Control Problem (OCP), in which we want to minimize the time that it takes to the UAV to reach the origin of coordinates in and environment with disturbances. If we know the disturbance, we can approximate the optimal control by making use of DRL methods. However, in the real world this is seldom the case, and we face unknown disturbances: we propose using robust control to deal with this case.

3.1. Markov Decision Processes

A convenient framework to model discrete time OCPs are Markov Decision Processes (MDPs), which are defined as [16], [17]:

Definition 1 (Markov Decision Process)**.** *An MDP is a 5-tuple $\langle S, A, P, R, \gamma \rangle$ where:*

- *S is the state set, containing all the possible states $s \in S$.*
- *A is the action set, containing all the possible actions $a \in A$.*
- *$P : S \times S \times A \to [0, 1]$ is the transition probability function, where $P(s_{n+1}|s_n, a_n)$ denotes the probability of transitioning to the next state s_{n+1} given that the agent is in state s_n and takes action a_n.*
- *$R : S \times A \to \mathbb{R}$ is the reward function, where $r(s_n, a_n)$ denotes the reward that the agent receives when it is in state s_n and takes action a_n.*
- *$\gamma \in (0, 1)$ is a discount factor.*

MDPs can be used to pose discrete time OCPs, and their solution is a policy $\pi : S \to A$, where $\pi(s_n)$ is a probability distribution over A denoting the probability that the agent selects action $a_n \in A$ when it is in state s_n. The optimal policy π^* is the policy which returns the highest cumulative reward to the agent, defined as:

$$\mathbb{E}_{\pi, P} \sum_{n=0}^{\infty} \gamma^n r(s_n, a_n), \quad (6)$$

where $\mathbb{E}$ denotes the mathematical expectation. A classical way of solving OCPs is Dynamic programming [18], [19], although if the action and / or action spaces are large, the problem becomes computationally intractable. Also, note that in many real cases, P and R may be unknown. Both problems can be addressed by using RL tools, which are biologically inspired and learn by interacting with the system using trial and error. A complete introduction to the field is given in [20], and this field has attracted a lot of attention recently as Deep Learning advances have been applied to RL to provide DRL methods that tackle with high dimensional problems.

3.2. Proximal Policy Optimization

In this work, we use Proximal Policy Optimization (PPO) [21] as DRL algorithm, which is an evolution of Trust Region Policy Optimization (TRPO, [22]). Both algorithms use a Deep Neural Network (DNN) of parameters ω to approximate the policy π_ω, and then, ω is optimized in order to maximize the cumulative reward. The maximum variation between the policy DNN in two consecutive optimization steps is bounded in order to avoid a performance collapse. TRPO is a successful algorithm due to its good performance [23], but it is computationally intensive: PPO was derived to alleviate that computational load. The optimization problem that PPO solves is:

$$\max_{\omega} \quad \mathbb{E}_{\pi_\omega, P} \left[\sum_{n=0}^{\infty} \gamma^n L(\pi_\omega, \pi_{old}, A_{\pi_\omega}) A_{\pi_\omega}(s_n, a_n) \right], \quad (7)$$

where π_{old} refers to the value of the DNN policy in the previous iteration, and $A_{\pi_\omega}(s_n, a_n)$ is the advantage function, used to estimate how good is action a_n when used in state s_n, estimated using another DNN. The key term is L, defined as:

$$L(\pi_\omega, \pi_{old}, A_{\pi_\omega}) = \min \left(\frac{\pi_\omega(a_n|s_n)}{\pi_{old}(a_n|s_n)}, g(\epsilon, A_{\pi_\omega}(s_n, a_n)) \right), \quad (8)$$

where

$$g(\epsilon, A_{\pi_\omega}(s_n, a_n)) = \begin{cases} 1 + \epsilon & \text{if } A_{\pi_\omega}(s_n, a_n) \geq 0 \\ 1 - \epsilon & \text{if } A_{\pi_\omega}(s_n, a_n) < 0 \end{cases}. \quad (9)$$

Intuitively, the basic idea of (8) is to limit the maximum difference between the old policy and the new one: the g term clips the

maximum variation in terms of the advantage and an ϵ parameter (9), and L chooses as minimization objective the smaller term between the quotient of policies and the clipped g term, hence, avoiding a too large difference between iterations. By properly adjusting ϵ, we get a trade between faster training, as policies are allowed to change more between iterations, and a safer training, as an abrupt change between policies may lead to a performance collapse in the new policy. Note that if we train PPO with a certain disturbance, then PPO will learn the optimal policy for that concrete disturbance, but if the disturbance changes, then we would no have any guarantees about its performance in the new environment.

3.3. Robust control

In real-life, a UAV faces uncertainty regarding the actual disturbance, that is, uncertainty regarding the transition model P. As mentioned, we can deal with this uncertainty using Bayesian methods, which may be used to estimate the value function [24] or the transition model [25], [26], but in this work we do not use these methods due to their high computational cost. Instead, we deal with uncertainty in the transition model using ideas of robust control, also known as robust MDPs or robust dynamic programming. The basic idea in this approach consists in using an additional fictitious agent, denoted as nature, that "controls" P, while the UAV selects the actions a, and their combined action control the transition to the next state. Agent and nature have opposite objectives: while the UAV wants to maximize the cumulative reward, the nature wants to minimize it. Hence, the underlying mathematical model is a two-player zero-sum game, in which the UAV maximizes and the nature minimizes. Thus, what the UAV has to solve is no longer a maximization problem, but a max-min problem, in which it maximizes the worst possible transition in terms of value. This approach presents advantages, such as having a more stable training [27], facilitating the transfer from simulations to real world [13], and allowing the agent to face unknown situations, which in our case means that the UAV can face disturbances it has not seen before. One disadvantage is that robust control may learn an excessively conservative policy [28].

We adapt the popular robust control framework from [27] to our UAV problem: we use PPO to model both the UAV and the nature, where the UAV chooses its acceleration θ_n and the nature chooses the disturbance value (d_x, d_y). Both agents are trained alternatively in an adversarial fashion: first, the UAV is trained while the nature policy is kept fixed, and then, the nature is trained while the UAV policy is kept fixed. This allows us training a robust policy which, as our simulations will show next, is able to face successfully unknown disturbances.

4. RESULTS

In this Section, we show the experimental results obtained. For each disturbance model presented in Section 2.2, we first train a PPO UAV to obtain the optimal policy for that case, and compare it both to robust agents and a simple baseline policy θ_b, which consists in accelerating towards the origin. Even though θ_b is a simple policy, which does not take into account disturbances or the velocity of the UAV, nonetheless it provides very good results in many cases, as it correspond to the intuitive case in which the acceleration points towards the origin, and as we will see, in some cases its performance is close to the optimal. Mathematically:

$$\theta_b = \arctan\left(\frac{-y_n}{-x_n}\right). \tag{10}$$

4.1. Simulation setup

For our simulations, we consider that $x, y \in [-10, 10]$ m, and $v_x, v_y \in [-2, 2]$ m/s, and implement a simulator following (1), with $\Delta = 0.1$ s and $k_f = 0.5$. At each time step n, the UAV observes its current state s_n, chooses an action θ_n, and the environment transitions to the next state s_{n+1} and returns the UAV a reward of -1 if the distance of the UAV to the origin is larger than 1 m. Note that it may happen that the UAV wanders without reaching the origin: in order to avoid lockouts, if the UAV does not find the origin within 100 time steps, the episode is finished. We set $\gamma = 0.9999$, so that by the end of the episode, the value of γ^n has not become negligible.

The PPO agent is implemented following [29]. The DNN policy is a feedforward NN, with an initial layer whose size is the state size, two hidden layers of 64 neurons, and a final layer which contains the mean and variance for the two action components, which we normalize so that the actions are the sine and cosine of θ_n. The cumulative reward is estimated using another DNN, with the same layers and sizes than the policy DNN, but the output is an scalar that estimates (6). We use the Generalized Advantage Estimation [30], with the default ϵ and $\lambda = 0.97$, to estimate A_{π_ω} by using the cumulative reward estimation. The UAV using PPO interacts with an environment with a certain disturbance which is kept fixed during training, but which is unknown a priori to the agent. The training allows the PPO agent to find the optimal policy for each disturbance, but note that this policy needs not be optimal if the disturbance varies. We train each PPO agent using 2000 iterations: in each one, the agent first interacts with the environment and stores 10000 experience vectors, formed by pairs of (s_n, a_n, r_n, s_{s+1}), which are then used to train the value and policy DNN. Both DNNs are trained using Adam [31].

It has to be emphasized that PPO needs to train using an environment with the concrete disturbance it has to face. In most real-life cases of interest, the disturbance that the agent will face is unknown. We address this by training a robust policy, in which two PPO agents with opposite interests are trained iteratively. The first agent is the UAV, while the second is the nature, that controls the disturbance: we consider that the actions of the nature are $(d_x, d_y) = (0.5 \cdot \cos(\beta), 0.5 \cdot \sin(\beta))$. We limit the maximum value of the actions of the nature to be smaller than the UAV action, so that the UAV is obstaculized by the nature, but not overpowered. We train both agents using 2000 iterations: in each one, we first train the UAV gathering 10000 experience vectors and then training the UAV as described in the previous paragraph. Then, we gather another 10000 experience vectors, and train the nature agent, which is a PPO agent with the same parameters as the UAV agent.

As we have mentioned, robust methods may provide too conservative policies [28], due to the fact that the adversary they trained against had a worse effect than the actual disturbance that they may face. In order to address this situation, that arises in our simulations, we decided to test the robust methods using two different set of observations. In the first one, the UAV and the nature observe the current state only, s_n (as with PPO): we denote this case as robust PPO without memory. In the second case, we provide as observation to both the UAV and the nature the states s_n and s_{n-1}: we denote this case as robust PPO with memory. Note that this allows both agents to infer what kind of adversary they face, and hence, adapt to it. In our case, it means that the UAV could be able to infer which are the local effects of the disturbance, and adapt to it. Even though this idea may lead to better performance, it also means increasing the state space, which beings a more difficult training.

	PPO (Optimal)	Robust PPO with memory	Robust PPO without memory	Baseline
S	-64.93 ± 23.4 (-71.04 ± 26.3)	-69.19 ± 25.0 (-69.67 ± 25.2)	-75.84 ± 25.5 (-82.36 ± 24.4)	-79.11 ± 25.2
V	-62.30 ± 22.5 (-67.24 ± 25.4)	-70.44 ± 24.9 (-71.24 ± 24.9)	-66.08 ± 23.7 (-76.41 ± 26.0)	-68.90 ± 23.6
H	-61.61 ± 23.2 (-64.37 ± 24.4)	-66.41 ± 25.0 (-67.35 ± 25.1)	-71.07 ± 25.1 (-78.30 ± 25.6)	-69.04 ± 24.9
C	-58.09 ± 18.6 (-73.79 ± 25.9)	-62.37 ± 21.8 (-63.56 ± 22.3)	-61.81 ± 20.3 (-74.12 ± 25.5)	-59.07 ± 17.6

Table 1. Total cumulative reward obtained for each method, using the same 100 initial states, where S stands for swirl, V for vertical current, H for horizontal current, and C for constant field. The numbers are mean $\pm$ the standard deviation: the best seed results come first, and the best three seeds results are in brackets. Very remarkably, robust PPO with memory consistently provides good results, even though it was trained without any knowledge of the disturbances.

4.2. Simulation results

We test out ideas training the PPO agent, as described before, in four environments, each with a different disturbance: a swirl (3), whose parameters are $(b, x_0, y_0) = (1/2, 5, 1)$; a constant field (2), with parameters $(a, \alpha) = (0.1, \pi/4)$; an horizontal current (4), with parameters $(c, y_0, w) = (1, 5, 3)$; and a vertical current (5), with parameters $(c, x_0, w) = (1, 5, 3)$. The PPO agent trained in these environments will learn the different optimal policies for each case, but as mentioned before, a change in the disturbance parameters or the type of disturbance may cause that the policy yields low rewards. The PPO UAV is trained using 10 different seeds, as DRL methods convergence is seriously affected by initial conditions [32]. We also train the robust PPO agents described, with and without memory: note that robust PPO is trained without knowledge of any disturbance, as their training is against the nature. Again, we train both PPO robust agents using 10 different seeds.

After training all agents, we compare their performance using the same 100 initial conditions s_0. For each disturbance, we compare the performance of PPO, robust PPO with and without memory, and the baseline 10. The results obtained can be observed in Table 1 and Figure 1. In Table 1, we represent the cumulative rewards obtained by averaging the results obtained both using the best and the three best seeds of each algorithm. If we focus on the best seed, in all cases PPO returns the best value, and this is to be expected, as PPO was trained in the environment with disturbance. Very remarkably, the results of robust PPO closely match the results of PPO, but note that robust PPO was trained without knowing the disturbance: the same robust PPO polices are used for the four disturbances and obtain good results, specially in the case of robust PPO with memory. This is the great advantage of using robust methods: they are able to provide very good results, even though the training conditions and test conditions differ. If we focus on the best three seeds average in Table 1, note that robust PPO with memory is the algorithm that provides a more stable training (i.e., small difference between the best seed results and the three best seeds), and thus, it provides the smallest variability on the results. By comparing both robust PPO methods, we observe that having memory is an advantage, not only in terms of less variability on the results, but also in the capacity to approximate the optimal policy results. Finally, note that the simple baseline provides in all cases worse or very similar results to robust methods: this, together with the fact that robust methods are trained once and may adapt to any disturbance, provides a strong reason to use them in UAV navigation problems.

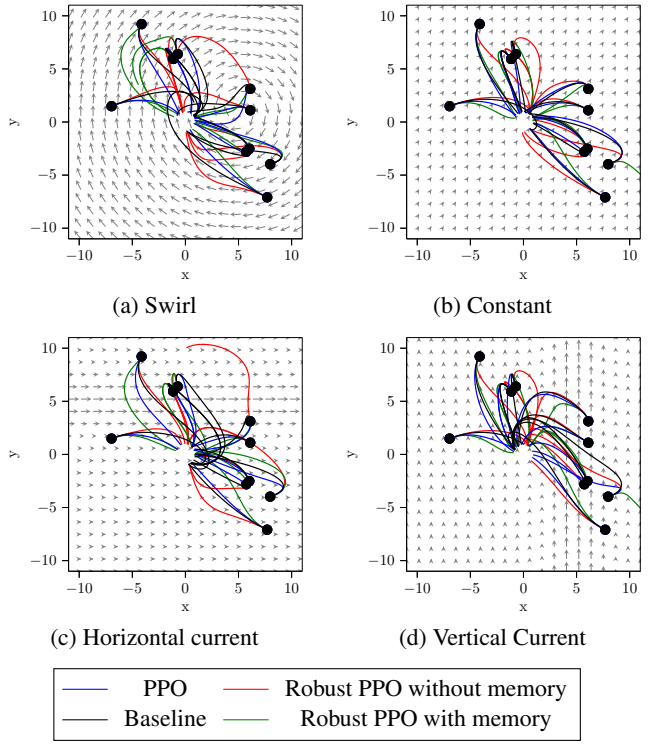

Fig. 1. Trajectory examples for each method for different disturbances, where the vector field in gray represent the disturbance effect and the black dots the initial positions. In all cases, the objective is to reach the origin as fast as possible, where the disturbance affects the trajectories. The initial conditions are randomly chosen and are the same for all disturbances.

5. CONCLUSIONS

In this work, we show that a robust PPO agent successfully copes with an underwater navigation problem with unknown disturbances, being able to match the performance of the PPO agent which knows the disturbance and offering better performance than a trivial solution, which consists in accelerating in the direction of the target. Thus, if we do not have a precise knowledge of the disturbances, robust DRL offers a valid tool to deal with this situation. Also, as we have shown, the robust agent strongly benefits from having a memory: a future line to this work could use more complex memory structures, such as LSTM neural networks [33]. Note that this may result in even better results, as these networks may be not only able of coping with unknown disturbances by using a robust method, but also with uncertainty about the state of the UAV: this situation is known as partial observability and has used LSTM to address this problem, as in [4]. We strongly believe that using LSTM not only helps managing partial observability situations, but also unknown disturbances.

6. REFERENCES

[1] Liam Paull, Sajad Saeedi, Mae Seto, and Howard Li, "Auv navigation and localization: A review," *IEEE Journal of Oceanic Engineering*, vol. 39, no. 1, pp. 131–149, 2013.

[2] Hiroshi Kawano and Tamaki Ura, "Motion planning algorithm for nonholonomic autonomous underwater vehicle in disturbance using reinforcement learning and teaching method," in *Proceedings 2002 IEEE International Conference on Robotics and Automation*. IEEE, 2002, vol. 4, pp. 4032–4038.

[3] A El-Fakdi, M Carreras, N Palomeras, and P Ridao, "Autonomous underwater vehicle control using reinforcement learning policy search methods," in *Europe Oceans 2005*. IEEE, 2005, vol. 2, pp. 793–798.

[4] Xiang Cao, Changyin Sun, and Mingzhong Yan, "Target search control of auv in underwater environment with deep reinforcement learning," *IEEE Access*, vol. 7, pp. 96549–96559, 2019.

[5] Yushan Sun, Junhan Cheng, Guocheng Zhang, and Hao Xu, "Mapless motion planning system for an autonomous underwater vehicle using policy gradient-based deep reinforcement learning," *Journal of Intelligent & Robotic Systems*, vol. 96, no. 3-4, pp. 591–601, 2019.

[6] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, Aviv Tamar, et al., "Bayesian reinforcement learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 8, no. 5-6, pp. 359–483, 2015.

[7] Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor, and Pascal Poupart, "Bayesian reinforcement learning," in *Reinforcement learning*, pp. 359–386. Springer, 2012.

[8] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein, "Deep neural networks as gaussian processes," *arXiv preprint arXiv:1711.00165*, 2017.

[9] Grani Adiwena Hanasusanto and Daniel Kuhn, "Robust data-driven dynamic programming," in *Advances in Neural Information Processing Systems*, 2013, pp. 827–835.

[10] Shie Mannor, Ofir Mebel, and Huan Xu, "Lightning does not strike twice: Robust mdps with coupled uncertainty," *arXiv preprint arXiv:1206.4643*, 2012.

[11] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem, "Robust markov decision processes," *Mathematics of Operations Research*, vol. 38, no. 1, pp. 153–183, 2013.

[12] Ding Wang, Haibo He, and Derong Liu, "Adaptive critic nonlinear robust control: A survey," *IEEE transactions on cybernetics*, vol. 47, no. 10, pp. 3429–3451, 2017.

[13] Paul Christiano, Zain Shah, Igor Mordatch, Jonas Schneider, Trevor Blackwell, Joshua Tobin, Pieter Abbeel, and Wojciech Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[14] Julia P Owen and William S Ryu, "The effects of linear and quadratic drag on falling spheres: an undergraduate laboratory," *European Journal of Physics*, vol. 26, no. 6, pp. 1085, 2005.

[15] Rufus Isaacs, *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*, Courier Corporation, 1999.

[16] Dimitri P Bertsekas, *Dynamic programming and optimal control*, vol. 1, Athena Scientific, 2005.

[17] Sebastian Thrun, Wolfram Burgard, and Dieter Fox, *Probabilistic robotics*, MIT press, 2005.

[18] Terry L Friesz, *Dynamic optimization and differential games*, vol. 135, Springer Science & Business Media, 2010.

[19] Dimitri P Bertsekas, *Dynamic programming and optimal control*, vol. 2, Athena Scientific, 2007.

[20] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press Cambridge, 1998.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[22] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[23] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[24] Yaakov Engel, Shie Mannor, and Ron Meir, "Reinforcement learning with gaussian processes," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 201–208.

[25] Yunpeng Pan, George I Boutselis, and Evangelos A Theodorou, "Efficient reinforcement learning via probabilistic trajectory optimization," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5459–5474, 2018.

[26] Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, 2019.

[27] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta, "Robust adversarial reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2817–2826.

[28] Esther Derman, Daniel Mankowitz, Timothy Mann, and Shie Mannor, "A bayesian approach to robust reinforcement learning," *arXiv preprint arXiv:1905.08188*, 2019.

[29] Joshua Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[30] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016,*, 2016.

[31] Diederik P Kingma and Jimmy Lei Ba, "Adam: A method for stochastic gradient descent," in *ICLR: International Conference on Learning Representations*, 2015.

[32] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger, "Deep reinforcement learning that matters," *arXiv preprint arXiv:1709.06560*, 2017.

[33] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.