IMAV2018-45
http://www.imavs.org/pdf/imav.2018.45

Towards Intelligent Aircraft Through Deep Reinforcement Learning

S. Morrison*, A. Fisher, and F. Zambetta
RMIT University, 414-418 Swanston St, Melbourne, VIC, 3000

ABSTRACT

Deep reinforcement learning has achieved recent successes in solving games and learning robotics tasks from scratch, and has shown early promise for the guidance, navigation, and control of MAVs. Though MAV control is well-established, many complex tasks still require human oversight, and techniques for reducing the level of human involvement are still nascent. In this paper, we present ongoing work in applying continuous-action deep reinforcement learning to autonomous aircraft in simulation, in order to learn such complex tasks autonomously. We provide a brief overview of our simulation environment and tasks of interest, and present preliminary results using model-free methods to learn simple flight tasks. We conclude with remarks on potential directions of research that we believe will have an impact on the future of unmanned systems.

1 INTRODUCTION

Deep reinforcement learning is a framework for training controllers in a manner that mimics learning in biological organisms [1,2]. It has recently achieved critical successes in playing computer Go [3,4], computer games [5,6], and learning nonlinear controllers for tasks such as robot locomotion and grasping [7,8]. Preliminary results have also shown that reinforcement learning can be used to train MAV flight controllers capable of complex flight tasks such as aircraft recovery [9] and collision avoidance [10]. Given RL's ability to learn complex we might hope to use it to learn robust flight controllers from scratch that enable greater levels of autonomy.

One of the primary impediments to this goal is the lack of a framework for training aircraft controllers. Deep reinforcement learning relies on the existence of simulated environments to train policies; applications of RL to the MAV domain in the literature are split on the use of the Robot Operating System (ROS) with Gazebo (typically RotorS [11]), and custom flight simulations. This is at odds with much of RL research, for which the primary benchmark is the OpenAI Gym framework. For our research, we are interested in learning individual flight tasks, and utilizing existing RL implementations. Whilst frameworks like RotorS are excellent for standard robotics research, they are unsuitable for RL, for which we might want to trial many thousands of episodes across a range of environments, in order to learn a policy.

In this paper, we present on-going work in building such a framework for learning nonlinear quadrotor flight controllers. Our framework extends OpenAI Gym, and includes flight control tasks such hovering, random waypoint navigation, landing, and target following. We hope that by sharing it, we can spur further innovation and development of intelligent flight controllers for small unmanned systems.

2 BACKGROUND

2.1 Deep Learning

Deep learning is a function approximation method loosely based on neural networks in the brains of biological organisms [1,2]. An artificial neuron takes input from surrounding neurons, performs a nonlinear transformation (known as an activation function), and then passes this output as input to a group of connected neurons [2]. The typical structure of a neural net is given in Figure 1, and

*contact email: address@domain.com

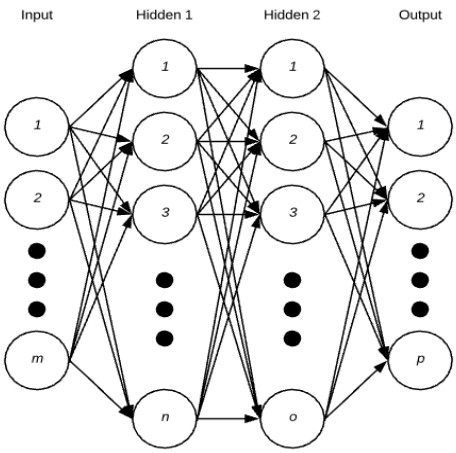

Figure 1 – Typical Multi-layer Perceptron

consists of a series of sequentially connected layers known as a Multilayer Perceptron (MLP). Neurons are connected to one another via a group of weights $\theta \in \Theta$, with the process of learning strengthening important weight connections and weakening detrimental ones. At a simple level, a neural network can approximate basic functions such as polynomials, but their real strength lies in being able to approximate functions that cannot be represented in any other way – for example, a function that detects cats in images, or estimates the value of an action – by learning them. The network is trained by minimizing a cost function or maximizing a score function. Gradients for each weight are pushed backwards through the network using the backpropagation algorithm – a variant of the chain rule from differential calculus – and an optimizer is used to step the weights in the given direction.

Specialist architectures such as Convolutional Neural Networks (CNNs) can learn filters that aid in the detection and classification of objects, and Recurrent Neural Networks (RNNs) are able to model dependencies through time [2]. CNNs see use in image classification, and have been applied to reinforcement learning for games. RNNs are used in cases where information is relevant over long time scales, and maintain a short term memory that aids in learning temporal dependencies [2]. Recent work has seen the development of generative models that learn to synthesize new examples of their training data [12,13], and memory networks that extend the functionality of RNNs through the use of an external long-term memory module [14,15]. These newer techniques are now finding their way into natural language processing and deep reinforcement learning, where they are learned to 'imagine' new scenarios and remember information over long very time scales [16,17].

2.2 Reinforcement Learning

Reinforcement learning (RL) deals with the problem of training an agent to act intelligently in an environment using an external reward signal [18]. The agent begins in an initial state s_0, and takes an action a_o according to a policy π that determines how actions should be selected. The agent receives a reward r_0 from the environment, and arrives in a new state s_1 that is governed by a transition probability $P(s_{t+1}|s_t, a_t)$. We refer to a sequence of such events as a trajectory (denoted τ) where $\tau = \{s_0, a_0, r_0, s_1, \dots, s_{t-1}, a_{t-1}, r_{t-1}, s_T\}$. The goal of an agent acting in an environment is take a trajectory that maximizes the total expected reward [18]:

$$R_t = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \right] \quad (1)$$

Where $\gamma \in [0,1]$ is a discount factor that more heavily weights immediate rewards, and ensures that the sum converges for the horizon $T = \infty$.

RL assumes a Markov Decision Process (MDP), meaning that the past and present are conditionally independent of one another given the present. More concretely, all information necessary for taking the best action is assumed to be included in the present state of the agent. As an example, a robot with a battery can take two trajectories, and end up in the same state s_t at the same time t, with different levels of remaining

charge. If the battery's charge is not included in the robot's state, the Markov property is violated, and learning becomes more difficult (for example, the robot may not have enough remaining charge to reach the goal). In this case, the Markov property can be preserved by ensuring the robot can "see" its current level of charge.

RL problems are typically solved by learning a value function $V: S \to \mathbb{R}$, that maps the relative value of each state (known as the state value function for π, $V^{\pi}(s_t)$), or each state-action pair $Q: S \times A \to \mathbb{R}$ (known as the state-action value function for π, $Q^{\pi}(s_t, a_t)$). Actions are selected according to the policy π; for example, our policy might be to take the highest value action with probability P, and choose a random action with probability $1 - P$. Similarly, if we know the true value of every state, we can always choose the highest value actions (known as the greedy action). Our policy can also be a function $\pi: S \to A$ that maps states directly to actions. This function is often a probability distribution over actions $\pi_{\theta}(a_t|s_t)$ with arbitrary parameters θ, with learning being the task of optimizing θ. This can be done by sampling actions from $\pi_{\theta}(a_t|s_t)$, evaluating their quality, and then adjusting θ accordingly.

Since – in continuous state-and-action spaces – there are infinite possible values, modern applications use function approximation to learn $V^{\pi}(s_t)$ and $\pi_{\theta}(a_t|s_t)$. For $V^{\pi}(s_t)$, this is done using supervised learning, by first rolling out an episode, and then calculating the return at each state using Equation 1. A cost function is minimised using these values as a target. The policy can be trained in multiple ways, but one of the more common techniques – known as Monte-Carlo Policy Gradient – uses a score function estimator of the form:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t|s_t)\, \Phi_t\right] \qquad (2)$$

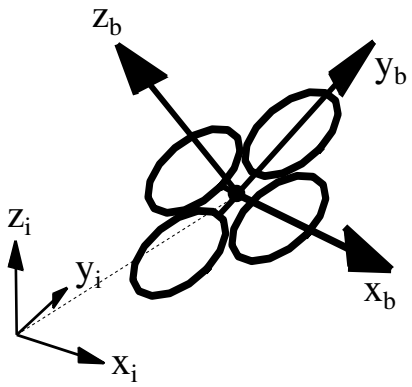

Figure 2 – Aircraft axis system

Where π is parameterized by weights θ, and Φ_t can take multiple forms, but is typically some variant of Equation 1 (see [18,19,20] for derivation and exposition of the Policy Gradient Theorem). By stepping the policy weights in the direction that maximizes the value function, it can progressively be trained to take better actions. The expressive power of neural networks makes them a popular choice for representing both the value function and the policy; when deep learning is combined with reinforcement learning, the combination is known as deep reinforcement learning.

3 SIMULATION AND ENVIRONMENTS

We model a plus-configuration aircraft in an East-North-Up axis system, and assume a flat Earth with constant air density. A diagram of our aircraft's axis system is given in Figure 2. Our equations of motion are:

$$\begin{bmatrix} 0 \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} 0 \\ \dfrac{\boldsymbol{F}_T + \boldsymbol{F}_A}{m} \end{bmatrix} + \boldsymbol{q}\begin{bmatrix} 0 \\ \boldsymbol{G}_i \end{bmatrix}\boldsymbol{q}^{-1} - \begin{bmatrix} 0 \\ \boldsymbol{\omega} \times \boldsymbol{v} \end{bmatrix} \qquad (3)$$

$$\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}(\boldsymbol{M}_T + \boldsymbol{M}_A - \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega}) \qquad (4)$$

$$\dot{\boldsymbol{x}} = \boldsymbol{q}^{-1}\begin{bmatrix} 0 \\ \boldsymbol{v} \end{bmatrix}\boldsymbol{q} \qquad (5)$$

$$\dot{\boldsymbol{q}} = -\frac{1}{2}\begin{bmatrix} 0 \\ \boldsymbol{\omega} \end{bmatrix}\boldsymbol{q} \qquad (6)$$

Where $\boldsymbol{x}$ is the position vector in the inertial frame, $\boldsymbol{v}$ and $\boldsymbol{\omega}$ are the linear and angular velocity vectors in the body frame, $\boldsymbol{F_T}$ and $\boldsymbol{F_A}$ denote the thrust and aerodynamic forces in the body frame, $\boldsymbol{M_T}$ and $\boldsymbol{M_A}$ denote the thrust and aerodynamic moments in the body frame, $\boldsymbol{G_i}$ is the gravity vector in the inertial frame, m is the mass of the aircraft, $\boldsymbol{J}$ is the inertia tensor, and $\boldsymbol{q}$ is a quaternion encoding the attitude of the aircraft. $\boldsymbol{q}$ is converted to Euler angles $\boldsymbol{\zeta}$, which are used by the RL environments. We use dot notation to denote the time derivative.

Thrust and torques are modelled as:

$$\begin{bmatrix} F_T \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} k_T & k_T & k_T & k_T \\ 0 & lk_T & 0 & -lk_T \\ -lk_T & 0 & lk_T & 0 \\ -k_Q & k_Q & -k_Q & k_Q \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} \quad (7)$$

For scalar thrust and moment coefficients k_T and k_Q, and arm length l. Aerodynamic forces and moments are:

$$\boldsymbol{F_A} = -k_D \boldsymbol{v^T v \hat{v}} \quad (8)$$

$$\boldsymbol{M_A} = -k_M \boldsymbol{\omega^T \omega \hat{\omega}} \quad (9)$$

For scalar drag and aerodynamic moment coefficients k_D and k_M. Rotor speeds are modelled as a first order linear differential equation:

$$\dot{\boldsymbol{\Omega}} = -k_c(\boldsymbol{\Omega} - \boldsymbol{\Omega_c}) \quad (10)$$

that ensures the RPM adjusts smoothly over time with policy commands. We step the simulation forward using a standard RK4 integrator. We do not currently include more advanced effects such as ground effect, blade flapping, vortex ring state, or wind, though some of these are planned for future work.

Current environments include:

1. Hover, for which the aim is to hover on a static waypoint for the duration of an episode;
2. Static waypoint navigation, for which the goal is to sequentially navigate to a waypoint and then hover there for the remainder of the episode;
3. Random waypoint navigation, for which the goal is to navigate to a randomly generated waypoint within a given distance to the aircraft;
4. Flying straight and level, for which the aircraft must fly in a constant direction for as long as possible, whilst maintaining constant altitude;
5. Landing, for which the aircraft must smoothly land without crashing, and without descending through its own rotor wash; and.
6. Target following, for which the aircraft must keep to a constant distance from a target as it moves through the environment.

Our reward function provides the aircraft with a positive reward for getting closer to a given goal, and a negative reward for moving away from it.

Our environments are episodic and terminate when the time limit has been reached or a termination condition has been met. These typically include flying beyond a given distance from the goal, though in the case of landing we also check for crashes. We ensure that the aircraft doesn't descend through its own rotor wash by checking the velocity in the aircraft's z-axis, and terminating the episode if it goes below -2m/s.

The observation space that is provided as input to the policy is often task dependent, but at a minimum it includes the position vector $\boldsymbol{x}$, $\sin(\boldsymbol{\zeta})$ and $\cos(\boldsymbol{\zeta})$, linear and angular velocities in the body frame $\boldsymbol{v}$ and $\boldsymbol{\omega}$, and – in goal-conditioned cases – the vector pointing towards the goal, $\boldsymbol{g}$. The target following case also includes the

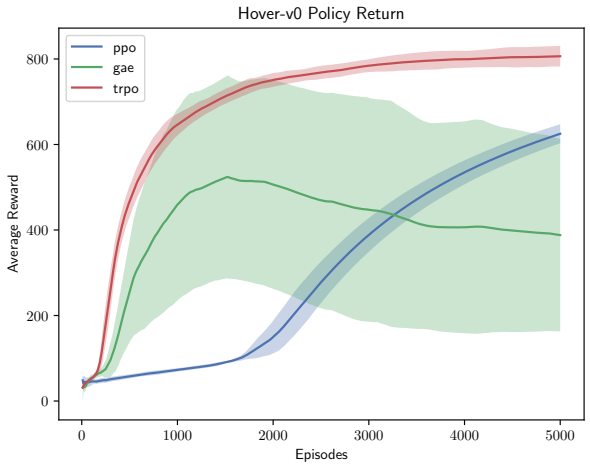

Figure 3 – Learning curves for the Hover task

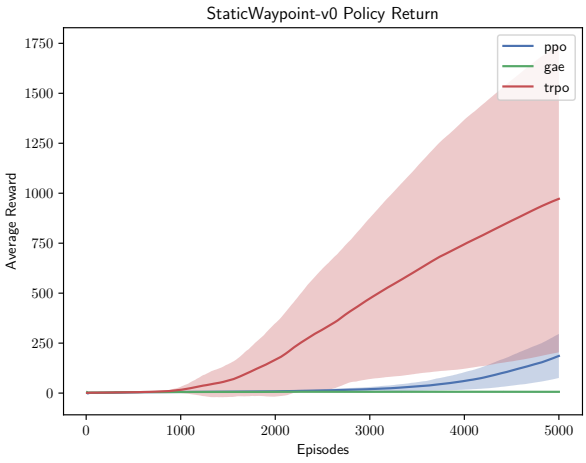

Figure 4 – Learning curves for the Static Waypoint task

velocity vector of the target. Per the MDP assumption, this information needs to be included in the observation space, and is realistically attainable for an aircraft. As our simulation does not currently include wind, we don't differentiate between the ground speed and airspeed of the aircraft. This is expected to change in future updates.

4 EXPERIMENTS

We applied three state-of-the-art Monte-Carlo policy gradient algorithms to learning hover, static waypoint navigation, and landing tasks. The algorithms we use are Generalized Advantage Estimation (GAE), Proximal Policy Optimization (PPO) and Trust Region Policy Optimization (TRPO), respectively. For further insight into these methods, we refer the reader to [21,22,23].

Our policies are standard MLPs with 128 neurons in a single hidden layer, and learn a delta from the hover RPM. The networks output the mean and log-variance of the action, and we use this to construct and sample from a normal distribution over actions.

5 RESULTS

We show learning curves in Figures 3, 4 and 5, and trajectory plots are given in Figures 6 and 7.

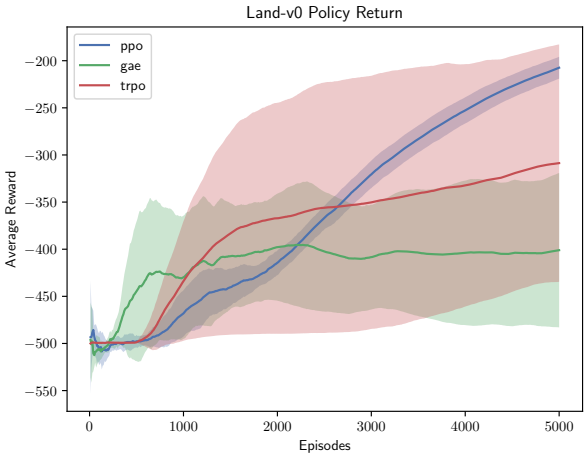

Figure 5 – Learning curves for the Landing task

We found that PPO and TRPO produced the most consistent overall performance, and provide video footage of our controllers at [address withheld]. In general, stochastic policies don't produce analogous performance to a modern controller, since they sample from a distribution. A stochastic policy will always produce a slight wobble due to its probabilistic nature. An advantage of this is that such policies should be robust to additional disturbances such as wind, and can produce smoother behaviour by taking the mean action, rather than sampling from the distribution.

In practice, we found that we were able to learn policies capable of achieving most goals. We

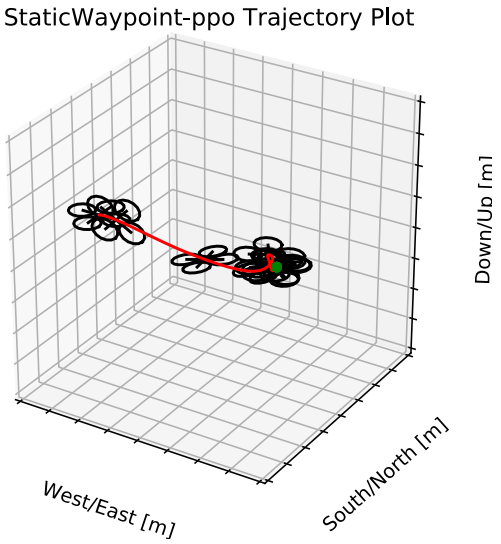

Figure 6 – Static Waypoint trajectory plot

found PPO to be the most consistent algorithm, though TRPO learned to flight behaviour that was visually smoother and more controlled. This is likely due to the difference in optimization methods – TRPO is a second order algorithm, whereas PPO is a first order method that approximates TRPO. PPO uses an optimizer known as ADAM that has been shown to find sharper local minima than other methods. We believe this is a potential underlying cause for the difference in flight behaviour.

Whilst GAE was able to learn most tasks, we found its performance to be higher variance than the other algorithms, and prone to collapse during training. This indicates that GAE might be less suitable for MAV flight control applications.

6 FUTURE WORK

Future work will involve expanding on the current environments to include navigation through unknown domains, perching, and the inclusion of stochastic wind. We aim to standardize the observation space across tasks so that we can use our current framework for multi-task learning (that is, a single controller that is capable of performing all tasks given some task parameter).

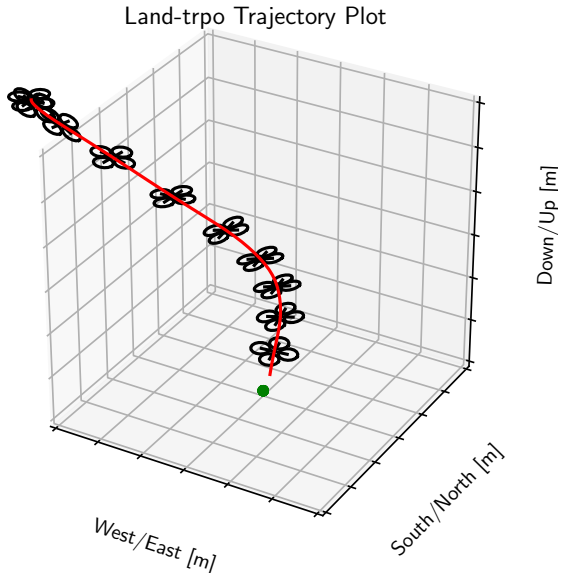

Figure 7 – Landing trajectory plot

Our goal is to fly a learned policy on an aircraft sometime towards the end of 2018.

6 CONCLUSION

The future of autonomous systems likely involves creating context-aware AI that is able to operate intelligently across a broad range of situations and tasks. The tools to build such systems might include recurrent neural networks that are able to make use of short term memory, memory-augmented networks that are able to write-to and read-from an external memory module, variational autoencoders that automatically learn a compressed representation of their input, and generative adversarial networks that can “imagine” new scenarios. Though such techniques are still difficult to train, research is ongoing and much preliminary work has already been done.

Training such agents is contingent on the existence of adequate simulation environments. We have presented ongoing work on such a framework that allows us to train up simple flight control policies, and we hope that our efforts may spur further effort in developing more intelligent flight controllers for MAVs.

ACKNOWLEDGEMENTS

The research was undertaken by the RMIT Unmanned Aircraft Systems Research Team, within the Sir Lawrence Wackett Aerospace Research Centre at RMIT University.

REFERENCES

[1] Russell, S.J. and Norvig, P., 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.

[2] Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y., 2016. *Deep learning* (Vol. 1). Cambridge: MIT press.

[3] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, *529*(7587), p.484.

[4] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and Chen, Y., 2017. Mastering the game of Go without human knowledge. *Nature*, *550*(7676), p.354.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M., 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. and Petersen, S., 2015. Human-level control through deep reinforcement learning. *Nature*, *518*(7540), p.529.

[7] Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M. and Silver, D., 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

[8] Arulkumaran, K., Deisenroth, M.P., Brundage, M. and Bharath, A.A., 2017. A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866.

[9] Hwangbo, J., Sa, I., Siegwart, R. and Hutter, M., 2017. Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters, 2(4), pp.2096-2103.

[10] Andersson, O., Wzorek, M. and Doherty, P., 2017, February. Deep Learning Quadcopter Control via Risk-Aware Active Learning. In *AAAI* (pp. 3812-3818).

[11] Furrer, F., Burri, M., Achtelik, M. and Siegwart, R., 2016. Rotors—A modular gazebo mav simulator framework. In *Robot Operating System (ROS)* (pp. 595-625). Springer, Cham.

[12] Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[13] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial nets. In Advances in neural information processing systems (pp. 2672-2680).

[14] Graves, A., Wayne, G. and Danihelka, I., 2014. Neural turing machines. arXiv preprint arXiv:1410.5401.

[15] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S.G., Grefenstette, E., Ramalho, T., Agapiou, J. and Badia, A.P., 2016. Hybrid computing using a neural network with dynamic external memory. Nature, 538(7626), p.471.

[16] Zhang, J., Tai, L., Boedecker, J., Burgard, W. and Liu, M., 2017. Neural SLAM. *arXiv preprint arXiv:1706.09520*.

[17] Ha, D. and Schmidhuber, J., 2018. World Models. *arXiv preprint arXiv:1803.10122*.

[18] Sutton, R.S. and Barto, A.G., 1998. *Reinforcement learning: An introduction* (Vol. 1, No. 1). Cambridge: MIT press.

[19] Peters, J. and Schaal, S., 2006, October. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* (pp. 2219-2225). IEEE.

[20] Peters, J. and Schaal, S., 2008. Reinforcement learning of motor skills with policy gradients. *Neural networks*, *21*(4), pp.682-697.

[21] Schulman, J., Moritz, P., Levine, S., Jordan, M. and Abbeel, P., 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[22] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[23] Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015, June. Trust region policy optimization. In *International Conference on Machine Learning* (pp. 1889-1897).