Deep Reinforcement Learning for Autonomous Model-Free Navigation with Partial Observability

1st Daniel Tapia
Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
Madrid, Spain
d.tapiam@alumnos.upm.es

2nd Juan Parras
Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
Madrid, Spain
j.parras@upm.es

3rd Santiago Zazo
Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
Madrid, Spain
santiago.zazo@upm.es

***Abstract*—Navigation is known to be a hard Sequential Decision-Making problem that attracts the attention of a large number of fields like Artificial Intelligence or Robotics. In this work, we approach the problem of partially observable navigation with a reactive system trained by model-free Reinforcement Learning. The advantages of this learned approach include reducing the engineering effort at the cost of more computing power during training. We designed an agent and an environment with a focus on being able to navigate independently of the map. We use well-tested general Reinforcement Learning algorithms without any hyper-parameter tuning and achieve promising results. Our results show that several general purpose Reinforcement Learning algorithms can reach the target in our navigation setup more than 85% of the episodes. Hence, these algorithms may provide a significant step forward towards autonomous navigation systems.**

***Index Terms*—Navigation, Reinforcement Learning, Robotics, Artificial Intelligence, Partially Observable Markov Decision Processes**

I. Introduction

Navigation is in the intersection between different fields like Artificial Intelligence, Robotics or Sequential Decision Making. It has been a challenge for many years to create systems that can traverse an environment without having previously explored it and doing it consistently as humans or animals do.

Classical Navigation approaches rely on internal representations or mappings of the state. The two most used ones in robotics are planning and mapping and sometimes the integration of these two [1]. The basics of planning are well covered in sources like [2]. More advanced methods can be found in [3] or [4]. However, these methods face the additional problem of accurately estimating the representation of the environment with Simultaneous Localization and Mapping (SLAM) algorithms [5], [6] apart from navigating in it.

Mapping approaches include memory-less methods that only act on instantaneous sensing data [7], like in our approach, but then using planning methods to create trajectories. Similar approaches called *reactive* use a memory-less representation of the environment and choose its trajectory from closed form solutions [8]. This last type of methods is the ones we are trying to learn with Reinforcement Learning.

Also, there is another disadvantage with the computation cost and memory footprint of these systems, which makes them harder to fit in small and low-powered embedded systems such as the processor unit in a small Unmanned Aerial Vehicle.

There has been recent interesting work on trying to learn to navigate with different representations of the map [9], [10] that pose simple tasks of navigating a grid-like maze environment and solve it with different policy architectures that use external memory. Or learning to navigate with implicit representations using general Reinforcement Learning (RL) algorithms [11], [12], [13], [14] however some of these works over-fit heavily on the the environment [12], [11] or use perceptually complex environments where the underlying structure is a simple graph [15], [16].

In this work we explore the approach of a reactive system, biologically inspired, that does not face problems related to internal map representations because it acts reacting to the observations. We attempt to solve this problem with different out of the box model-free Reinforcement Learning algorithms [17] and compare how well they perform in a simple but challenging environment. These algorithms have shown promising success in different tasks like superhuman performance in games like Atari [18] or Go [19]. And the advantage over handcrafted systems is that given the appropriate task definition and problem statement, it is possible to leverage computing power to replace a human effort of writing a very complex program with all the rules for a learned system that solves the task.

Model-free algorithms are promising in tasks where it is difficult to hard-code all the rules of the system and it is hard to gather lots of expert data for supervised algorithms, but where simulation and computing power are cheap and available. However aside from these advantages they also face different challenges [20] that have to be taken into account in order to succeed in our task. In this work, we will explain the strategies and design decisions used to deal with problems such as

The work is supported by the Universidad Politécnica de Madrid CAIT with a student grant to the first author. It is also supported by the Universidad Politécnica de Madrid by a PhD grant given to the second author, as well as by the Spanish Ministry of Science and Innovation under the grant TEC2016-76038-C3-1-R (HERAKLES). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

Sample Inefficiency, Reward Sparsity or Environmental Overfitting.

The rest of the work goes as follows: in section II we define the problem and set the theoretical background we are following. Section III describes our system and the design decisions involved. In section IV we analyze and comment the results and finally section V draws some conclusions.

II. Problem Description

A. Navigation

Navigation is the field that studies the skills and techniques needed by a mobile system to traverse an environment. From the different tasks involved in Navigation, we choose to tackle PointGoal [5] where the agent will have to move in the environment to reach the goal point $\|\vec{d}\| < \varepsilon$ with a controlled speed $\|\vec{v}\| < \epsilon$, with ε being a distance threshold and ϵ a speed threshold. This task is trivial if there are no obstacles in the environment but navigating in a cluttered environment is an NP-hard problem [2]. Reaching the objective depends on all the previous actions taken, thus making Navigation a sequential decision problem where the sensors provide the input signal and after we process them our system outputs an action. We will use a mathematical model that suits this problem called Partially Observable Markov Decision Processes (POMDP).

B. POMDP

We can formulate the problem as a POMDP, which is an extension of Markov Decision Processes where the agent is unable to observe the current state. This means that for each time step t the agent only has an observation o which may not contain all the information of the state s.

A partially observable Markov decision process is formulated [21] as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, \Omega, O \rangle$, where:

- $\mathcal{S}$ is a set of states of the world. In our case, it is infinite because the space is continuous.
- $\mathcal{A}$ is a finite set of actions.
- $T\colon \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$ is the state-transition function, giving for each world state and action, a probability distribution over world states. We write $T(s, a, s')$ for the probability of ending in state s' , given that the agent starts in state s and takes action a.
- $R\colon \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, giving the expected immediate reward gained by the agent for taking each action in each state. We write $r(s, a)$ for the instantaneous reward for taking action a in state s.
- Ω is an infinite set of observations the agent can experience of its world.
- $O\colon \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\Omega)$ is the observation function, which gives, for each action and resulting state, a probability distribution over possible observations. We write $O(s', a, o)$ for the probability of obtaining observation o given that the agent took action a and landed in state s'.

We define in (1) a performance metric R^T as the expected sum of discounted rewards, using a discount factor of $0 < \gamma < 1$ for a sequence of rewards in an episode of length T.

$$R^T \ = \ \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right] \tag{1}$$

C. Reinforcement Learning

We define the policy π as the function that maps from observations to actions. If the observation space is finite we could use a look-up table. But when we have a continuous observation space and we do not approximate by discretizing it, we have to use a more powerful function approximator. In our case we use a Multi-Layer Perceptron [22], mapping $a_t = f(o_t; \theta)$, where θ are the Multi-Layer Perceptron parameters .

The Reinforcement Learning problem is to find or to approximate the optimal policy π^* that maximizes the performance metric defined in (1).

$$\pi^* = \arg_\pi \max R^T \tag{2}$$

It is approximated with different algorithms. It can be done mainly by two methods:

- Policy Optimization where the performance objective $R^T(\pi_\theta)$ is optimized by gradient ascent of the policy function or by maximizing local approximations of the policy function. As in the methods A2C [23], ACKTR [24], ACER [25].
- Q-Learning that learns an approximator $Q_\theta(s, a)$ for the optimal action-value function $Q^*(s, a)$, that tells how good is to take an action in a certain state and is used to derive the optimal policy π^*. As in DQN [18].

In this work, we compare different approaches as there are certain trade-offs between sample-efficiency and reliability [26].

III. Experiment Details

A. System Description

The model studied in the experiments consists of an agent, which represents the mobile that navigates, and an environment, which represents the world in which the agent is located. The interactions between agent and environment happen at discrete time steps. For each time step t the agent receives an observation o_t and a reward r_t, and interacts with the environment by taking an action a_t, which may affect the state s_{t+1} of the environment and the future observations. This feedback loop allows us to define episodes of a maximum length T where the RL algorithm will try to maximize the performance score $R^T(\pi_\theta)$ enough to solve the task of reaching the goal point without collisions.

The agent has no prior knowledge of the environment and only sees what it is received by the observations. In our system, o_t are the raw sensory input and they do not contain all the information of the state of the environment. We define the observation vector as the concatenation of several elements:

- The first two elements are the Cartesian coordinates of the clipped distance vector (3)

$$\vec{d}_{clipped} = \frac{d_{max}(\vec{p}_{target} - \vec{p}_{agent})}{\max\left(d_{max}, \|\vec{p}_{target} - \vec{p}_{agent}\|\right)} \tag{3}$$

  Where $\vec{p}_{target}$ and $\vec{p}_{agent}$ are the positions of the target and the agent. Where d_{max} is a threshold distance that limits the size of the distance vector, making the agent generalize better over maps of different sizes. In practical terms d_{max} can be seen as the limited range of the sensors.
- The third and fourth elements are the components of the velocity vector $\vec{v}$.
- And the rest of the input elements are a radial collision lattice around the agent that simulates proximity sensors like a lidar device. The value of the lattice elements take values of 1 when they collide with an obstacle and 0 when they do not.

For each time step t the output of the agent after processing signal o_t is action a_t . This action is an integer representing the choice between 4 possible actions. These choice of actions are increasing or decreasing one of the components of the velocity, thus accelerating in one of four possible directions. We chose to model the actions as a discrete choice because it conditions the available algorithms and limits the computational complexity of the problem.

We found that other environments [27], [28], [16] used to benchmark this type of task were either discrete grids, where the obstacle avoidance problem is trivialized or lacked the diversity in environments making the agent memorize the map and not being able to generalize across different scenarios.

Our environment follows simple yet realistic dynamic rules with continuous space, which follows a double integrator equations (4) in which inertia is taken into account. We address the diversity problem by randomizing the generation of the environment which we later explain in subsection III-C.

$$\begin{aligned} \vec{p}_{t+1} &= \vec{p}_t + \vec{v}_t \Delta t \\ \vec{v}_{t+1} &= (F\vec{a}_t - k\vec{v}_t)\Delta t + \vec{v}_t \end{aligned} \tag{4}$$

Where F is a constant used to regulate the force of the acceleration and $k = \frac{F}{|v_{max}|}$ is a term used to add some friction. The term $\vec{a}_t$ represents the action as a direction vector in which the force is applied. The vector $\vec{p}$ is the position vector and $\vec{v}$ is the velocity vector. The constant Δt is the time step.

B. The Agent

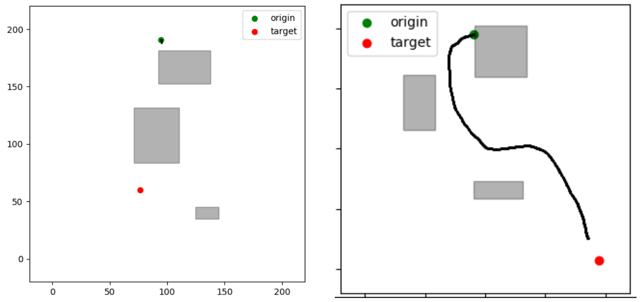
We defined the Agent as a system where the output are the actions a_t and the inputs are the observations o_t and the reward r_t that we use in the performance function we try to optimize.

As the scope of this work is not focused on the algorithmic side of RL, we chose to use a well tested and documented implementation of the algorithms taken from [17] and used without any hyper-parameter tuning.

However, RL has some problems [20], [29] which, if not taken care of, can lead to failure in our task. The first one is sample inefficiency, meaning that these algorithms take millions of steps to converge. The second problem is reliability: due to the stochastic nature of the algorithms used, these will sometimes fail to converge. Another problem we face is that sparse rewards lead to slower training, so we should use a denser reward than +1 if you reach the target, else 0. But the definition of the reward is the definition of the problem, so we should make sure it captures the task we want to achieve and the correct way to do it. Another problem is the Environment over-fitting, meaning that if there is not enough diversity in the spaces explored by the agent, it may just be memorizing how to act in that specific environment which does not generalize well to unseen ones. These problems are tackled with the design decisions and strategies in the Environment we created.

C. The Environment

The environment consists of a 2D rectangular map with 3 rectangular obstacles placed randomly inside, the starting point and the goal location. To mitigate the problem of sample efficiency we did the environment very simple so that it could run fast and the agents could be trained in a couple of hours. We can see an example of a generated environment in Fig. 1a where the gray rectangles are the obstacles, the green dot is the starting point and the red dot is the goal.



(a) Example of a generated map with 3 obstacles.

(b) Example of a successful trajectory.

Fig. 1: Example of the map and a sample trajectory

The definition of the problem in the environment is done through reward shaping. We want the agent to arrive at the spot and not crash, so we include those factors into the reward function with the first two cases of (6). Then we want the reward to be dense, meaning that it takes nonzero values in places other than end-states because it speeds up training. To solve the reward sparsity problem we designed a function (5) for the third case of (6) that took into account three factors that captured the problem definition.

- We want to take distance into account, the closer to the target the better. This is represented in the first element of the expression$(-\frac{d^2}{100})$.

- We want to include the speed in the reward, but only close to the target, as our goal is to reach the target with a limited speed. This is captured in the second element of the expression $(-\min((3v^2-80),0)\frac{(d^2+1)}{(3d^2+1)})$ that is quadratic with the speed but only in a region close to the target.
- We also want the agent to reach the target fast so we should take time into consideration by making all the reward negative except for the successful ending. This is captured in the third element of the expression (-80) which offsets all the expression under 0.

The plot of the function of (5) along the distance and speed axis can be seen in Fig. 2:

$$f(\vec{d},\vec{v}) = -\frac{\|\vec{d}\|^2}{100} - \min((3\|\vec{v}\|^2-80),0)\frac{(\|\vec{d}\|^2+1)}{(3\|\vec{d}\|^2+1)} - 80 \tag{5}$$

$$r(s,a) = \begin{cases} -10 & \text{crash} \\ +10 & \|\vec{d}\| < \varepsilon, \|\vec{v}\| < \epsilon \\ f(\vec{d},\vec{v}) & \text{else} \end{cases} \tag{6}$$

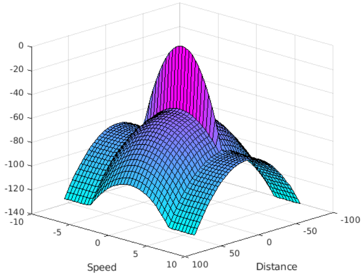


Fig. 2: The reward function over the distance and speed axis.

Every episode the environment will change the size and location of the obstacles, starting point and goal location making the agent unable to memorize the environment and forcing it to develop a strategy that enables it to reach the objective independently of the disposition. The idea of randomizing the environment in training was a key point in [20] and proved to be an effective yet simple solution to overfitting the environment.

IV. Results

The training for each algorithm was run 10 times with different seeds to make sure the less reliable methods converged. At the end of 10^6 time-steps of training the agents were evaluated for 100 episodes and its performance was measured. We evaluate the performance of the algorithms by four metrics.

- The first one, reliability refers to the percentage of the 10 seeds that did not collapse the policy to bad local maxima, meaning the percentage of seeds in which from the 100 evaluation runs at least 10 runs reached the objective.

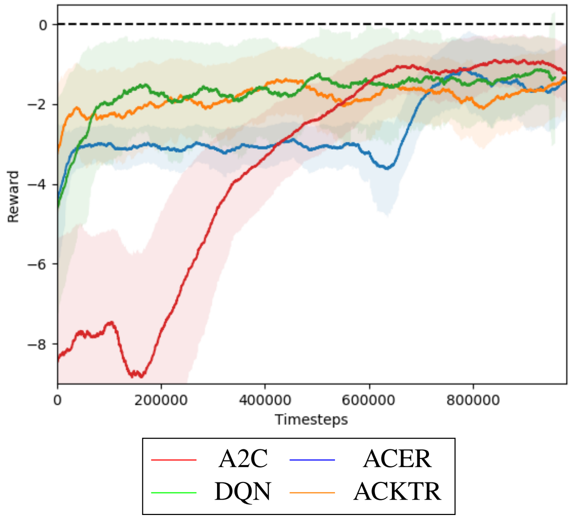


Fig. 3: Learning curves of the best seed for each algorithm smoothed over 400 time steps. Shadowed area represents $\pm0.5\sigma$ over the smoothed average.

- The second one is time. Measuring the average time it took the algorithms to train for 1M timesteps.
- The third metric, average score refers to the mean reward in the last 100 evaluation time-steps. The evolution of this metric during the training is shown in Fig. 3. By design decisions of the reward function, it will be always negative except for the success-case where the agent reaches the goal, this means that the performance metric will mostly be negative. So to understand better this metric, when the average score goes above -1.5, it means that the agent starts completing episodes. This is why we use the fourth metric, to clarify and explain in a more qualitative way how the algorithm is performing.
- The fourth metric is the percentage of successfully finished episodes in the 100 evaluation runs, which is much more descriptive than the average score. For the third and fourth metric, we chose the best performing seed of each algorithm.

TABLE I: Algorithm Performance

Alg. Name	Performance			
	Reliability	***Time***	***Avg.Score***	***Completed***
DQN	100%	3380s ± 12s	-1.156	85%
A2C	100%	2450s ± 6s	-0.957	88%
ACER	40%	2859s ± 36s	-0.990	87%
ACKTR	100%	2097s ± 28s	-1.190	86%

The results are displayed in Table I and the training curves are shown in Fig. 3. Looking at them we see that the algorithms DQN and ACKTR rapidly converged to an average score close to the final performance, and steadily increased from there. The A2C algorithm is the slowest one to converge but steadily rises to higher performance than the other three.

The ACER algorithm quickly converges to an average score of -3 and until later in training, it does not leave the bad local maxima. About the reliability of ACER, only four of the ten seeds managed to converge to a policy that reached the target. These results could be explained because ACER is trained off-policy and these methods are supposed to be more sample efficient but less stable [26]. Looking at the *time* category we can see that DQN is the slowest, and ACKTR is the fastest computationally.

Regarding the *Completed* score of the four algorithms, we can note that they successfully completed the task more than 85% of the times which is impressive for a general-purpose reinforcement learning algorithm that started without any previous knowledge about the task it is solving.

On qualitative analysis, while watching the agent during training, we can point out what several stages of the training looked like. At the beginning with the randomly initialized policy, the agent moved erratically shaking or directly going outside the map. When it reaches the plateau of -3 of *Average Score*, it learned to move in opposite directions each time-step, resulting in standing still and avoiding crashes and bad outcomes. But it was also unable to reach the target. Finally, when the agent is trained we can see it avoids certain obstacles like in Fig. 1b, but failed to evade others when it moves fast. So we should take into account that an 85% does not represent an industry-ready navigation algorithm, but provides a strong result for model-free RL algorithms and settles a baseline to compare other methods to it.

V. Conclusion

In this work, we approach the problem of partially observable navigation with a reactive system trained by model-free Reinforcement Learning. This approach is attractive because it can reduce engineering effort at the cost of more computing power during training. We designed an agent with a focus on it being able to navigate independently of the map. And we developed an environment with a reward function that captures the problem we are trying to solve. We used well-tested RL algorithms without any hyper-parameter tuning and achieved promising results. Going from agents moving aimlessly, to reaching the target more than 85% of the episodes. Despite having promising results, there is still work to be done with this approach and other ones to achieve industry-ready navigation systems.

References

[1] J. Tordesillas, B. T. Lopez, J. Carter, J. Ware, and J. P. How, "Real-time planning with multi-fidelity models for agile flights in unknown environments," *arXiv preprint arXiv:1810.01035*, 2018.

[2] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[3] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2520–2525.

[4] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in se (3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[5] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva *et al.*, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.

[6] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[7] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Eriksen, S. Daftry, M. Hebert, and J. A. Bagnell, "Vision and learning for deliberative monocular cluttered flight," in *Field and Service Robotics*. Springer, 2016, pp. 391–409.

[8] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5759–5765.

[9] J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in minecraft," *arXiv preprint arXiv:1605.09128*, 2016.

[10] E. Parisotto and R. Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," *arXiv preprint arXiv:1702.08360*, 2017.

[11] A. Dosovitskiy and V. Koltun, "Learning to act by predicting the future," *arXiv preprint arXiv:1611.01779*, 2016.

[12] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[13] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[14] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," *arXiv preprint arXiv:1611.04201*, 2016.

[15] S. Brahmbhatt and J. Hays, "Deepnav: Learning to navigate large cities," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5193–5202.

[16] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," *arXiv preprint arXiv:1709.06158*, 2017.

[17] A. Hill, A. Raffin, M. Ernestus, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

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

[19] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[20] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[21] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[23] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[24] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in neural information processing systems*, 2017, pp. 5279–5288.

[25] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 2011.

[27] M. Savva, A. X. Chang, A. Dosovitskiy, T. Funkhouser, and V. Koltun, "Minos: Multimodal indoor simulator for navigation in complex environments," *arXiv preprint arXiv:1712.03931*, 2017.

[28] M. Chevalier-Boisvert and L. Willems, "Minimalistic gridworld environment for openai gym," https://github.com/maximecb/gym-minigrid, 2018.

[29] A. Irpan, "Deep reinforcement learning doesn't work yet," https://www.alexirpan.com/2018/02/14/rl-hard.html, 2018.