ENABLING SAFE EXPLORATION OF ACTION SPACE IN REAL–WORLD ROBOTS

Shivam Garg[*1] **, Homayoon Farrahi**[*1]**, and A. Rupam Mahmood**[1,2]
[1]Reinforcement Learning and Artificial Intelligence Lab, University of Alberta, Canada
[2]Canada CIFAR AI Chair, Amii, Canada
{sgarg2,farrahi,armahmood}@ualberta.ca

ABSTRACT

The use of robots has been steadily increasing, and reinforcement learning (RL) has shown increasingly promising results in controlling physical robots. However, due to physical safety constraints, RL methods are often unable to exploit the full action space of the robots, which can be crucial in achieving good control performance. In this work, we experimentally demonstrate that using higher magnitude action spaces leads to better policy performance on a physical reaching task. Based on this result, we propose a curriculum technique for safe exploration of the full action space. Our experiments on a physical robot show that this technique applied to a policy gradient method allows RL agents to explore safely while utilizing the full range of the action space.

1 INTRODUCTION

Robots are important in automating manual tasks, and have already been deployed extensively in manufacturing to reduce costs and improve efficiency. They have been instrumental in manufacturing automation (automobile industry), collaborating with humans in high precision environments (medical robotics), or working in dangerous situations (mine detection robots). Robots have greatly minimized the human physical effort. However, developing robot control algorithms is a painstaking task requiring expert knowledge. Apart from being time consuming to develop, the traditional approaches don't scale across tasks, i.e. a human expert needs to separately devise an algorithm for each different robotic task. The need to minimize human intellectual effort provides the motivation to develop methods, which can scale across tasks and help robots learn effective control without human supervision.

Reinforcement Learning (RL) (Sutton & Barto, 2018) promises to alleviate these problems by providing general purpose control algorithms which can automate the learning process of the robot with minimal human intervention. However, the application of RL methods to real world robots has been limited due to the need for high amount of experience required to learn a policy from scratch and the complexity associated with physical hardware systems.

Recent work has shown some promising results in applying RL to real world robots (Mahmood et al., 2018a; 2018b). RL methods learn control by directly interacting with the robot in real–time. They require minimal human intervention (except while setting up the task) and have shown to consistently learn good control policies. However, in experiments, RL methods were not able to learn control policies better than those obtained from traditional control engineering (scripted agents like the PID controller). We hypothesize that there are two major reasons for this observation:

1. The range of action space (such as motor torques) was restricted during the experiments while learning, in order to prevent aggressive exploratory behavior (such as jerkiness, motors getting too hot, moving violently causing harm to surroundings and its internal components); and
2. The training time on robots was limited to short durations.

[*]Equal contribution.

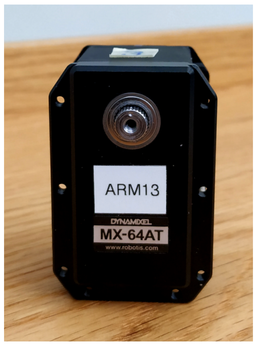

Figure 1: Dynamixel MX–64AT motor.

Consequently, increasing the range of actions while learning robot control, and training the robots for long durations maybe necessary to learn superior control. The focus of this work is on mitigating the effects arising from the former reason. This area of study, where RL algorithms are applied to critical systems with safety constraints is known as Safe RL (Gardia & Fernández, 2015).

We propose to solve this problem by developing a method which enables the robots to safely explore their complete action space. In this project, we will focus on controlling the Dynamixel MX–64AT motor shown in Figure 1. We try to answer the following questions:

1. Does providing the motor access to a larger action space actually result in superior performance; and
2. How do we safely train a policy on the complete action space of the robot?

For answering the first question, we train the robot on identical control tasks with a sequentially increasing action space (e.g. torque range increase from $[-100, 100]$ to $[-200, 200]$). This will help us evaluate if the policies learned for tasks with a larger action space are superior to those learned on a smaller action space. For each new task, we can either learn the policy from scratch or utilize the policy learned from the previous task, leading to two separate approaches.

To answer the second question, we will train a policy successively on identical tasks with increasing action space based on a fixed schedule. We expect the smoothness of the actions generated by the policy, learned on the previous task, to carry over to subsequent tasks with larger action spaces, effectively enabling safe exploration.

In summary, our method will enable robots to safely explore the complete action space. By utilizing the full motor capabilities of robots, RL methods would ideally be able to find control policies superior to those obtained from classical techniques, which would help increase the adoption of robots in different domains.

2 Background and Methodology

2.1 Task

We run our experiments on the Dynamixel MX–64AT servo motor manufactured by ROBOTIS. It features a stall torque of 6 N.m and a maximum no load speed of 63 rpm. We define our task similar to the DXL–Reacher task defined by Mahmood et al. (2018b). In this task, the agent can control the motor through position, speed, or torque. We use torque control for all of our experiments. The agent's goal is to move the motor to the target position as fast as possible.

2.2 No Scheduling: Fixed Torque Range

To answer the first research question, we compare the learning curves for four different constant torque ranges. For these experiments, the network's mean output is fed into a `tanh()` function to

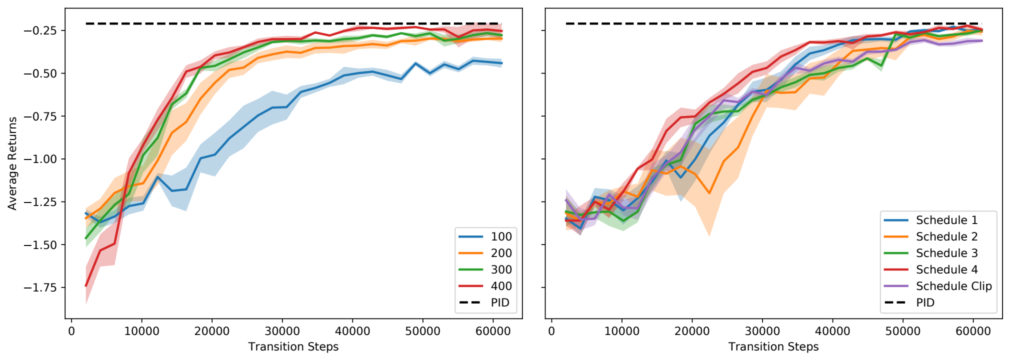

Figure 2: Average returns (across batch size of 40 episodes) against the transition steps for different variants of learning methods on the Dynamixel Reacher task. The graphs show mean and standard error for five independent runs of each method. The solid horizontal line corresponds to the average performance of a simple PID controller on the reacher task.

map it between -1 and 1, then multiplied by a scaling factor τ. Runs with different scaling factors are then compared in terms of the final return performance and the speed of learning.

2.3 Scheduling Methods

Next, we investigate the effect of two scheduling techniques for gradually increasing the action space.

Scaling Schedule Method: The first method is scaling. The network's output is mapped to the range [-1, 1] using `tanh()` and then multiplied by a constant scaling factor. This scaling factor can then be increased using different schedules. Various scheduling methods differ in their type e.g.: linear or quadratic and how often the scaling factor is increased during training (such as gradually increasing the scaling factor). Using the scaling technique results in sudden jumps in the action magnitudes whenever the scaling factor is increased. Howerver, the output of the policy network (mean of the policy distribution) remains bounded via `tanh()`.

Clipping Schedule Method: Another technique is to clip the policy action. In this case, the network's output is multiplied by a constant factor and clipped to be within the range $[\texttt{-clip}, \texttt{clip}]$. This range can then be increased using the same scheduling methods described above. In the clipping technique, the output mean is not bounded and can go well beyond the clipping range. This makes it harder to learn policies, since small changes to the policy can result in the same output action if the action is beyond the clipping range and thereby gets clipped to the threshold. However, unlike the scaling method, there are no longer any sudden jumps in the actions.

3 Experimental Setup

3.1 Environment and the Learning Algorithm

We use our own implementations of the Reacher task and the PPO algorithm (Schulman et al., 2017). For the PPO algorithm, we use neural networks as function approximators for both our policy and value functions and train them using the Adam optimizier (Kingma & Ba, 2014). Our observation space consists of $[\texttt{motor\_angle}, \texttt{motor\_speed}, \texttt{target\_angle}]$ for the motor. Each observation is normalized to lie between 0 and 1. Both neural networks have two fully–connected hidden layers of size 64. The output of the value network is a single number and the output of the policy network consists of two numbers μ and σ defining the Normal distribution from which we then draw our agent's actions: $A_t \sim \mathcal{N}(\mu, \sigma)$; this way both μ and σ are state dependent. We apply a `softplus()` function to σ to ensure that it stays positive. For fixed torque schedules and scaling schedules, we constrain the mean output of the policy network between -1 and 1 by applying

Table 1: The average values (along with standard deviations) of various metrics across the learning phase of different policy variants on the reacher task. *T–value* policies refer to fixed torque policies with $\tau = value$, *S–value* policies refer to scaling schedule policy with schedule type equal to *value*, and *CLIP* policy refers to clipping schedule policy. The details are given in Section 3.2. Note the extremely high variance of the values.

Policy	Current	Temperature	Load	$\lvert A_{t+1} - A_t \rvert$	$\lvert \tau A_{t+1} - \tau A_t \rvert$
T–100	-4.02 ± 57.0	38.5 ± 1.57	-48.7 ± 50.1	0.66 ± 0.86	65.6 ± 85.7
T–200	-3.96 ± 102	43.0 ± 2.26	-45.5 ± 50.1	0.63 ± 0.84	125 ± 168
T–300	$+2.59 \pm 150$	47.1 ± 3.09	-41.5 ± 50.2	0.68 ± 0.90	204 ± 269
T–400	-4.26 ± 170	48.1 ± 3.74	-42.6 ± 50.2	0.58 ± 0.78	234 ± 310
S–1	-6.14 ± 130	42.8 ± 4.06	-46.5 ± 50.2	0.73 ± 0.95	176 ± 249
S–2	-6.18 ± 135	45.7 ± 3.72	-47.0 ± 50.3	0.76 ± 0.99	176 ± 260
S–3	-1.96 ± 97.5	40.8 ± 1.87	-46.1 ± 50.2	0.67 ± 0.85	114 ± 183
S–4	-1.23 ± 129	41.1 ± 5.36	-43.7 ± 50.2	0.70 ± 0.90	177 ± 245
CLIP	-1.51 ± 65.5	40.3 ± 2.62	-46.9 ± 50.2	0.77 ± 0.99	76.5 ± 99.0

`tanh()`; for clipping schedule method we don't constrain the mean output. Once we obtain the action A_t, we scale it with a constant scale factor τ, clip it to be within the range [`-clip`, `clip`] and then pass the result as the action to the environment.

We implement the Reacher task using the drivers provided by SenseAct library (Mahmood et al., 2018b). At the beginning of each episode, the motor's position is reset to zero and the target angle is uniformly sampled from -1.31 radians to +1.31 radians. We maintain an action cycle time of 40ms and keep the episode length to 2s; this corresponds to each episode having exactly 51 transition steps. The constant cycle time is achieved by applying an action, suspending the process for some milliseconds and reading the next state from the motor. The sleep duration is adjusted adaptively based on the elapsed time in each transition. This also ensures that the next observed states are correlated with the applied actions. The reward function is the negative of the absolute difference between the present motor angle and the target angle, scaled by the action cycle time.

While learning, we use a batch size of 2040 corresponding to 40 episodes, i.e. we sample 40 episodes using the current policy and then update the PPO agent on these collected transitions. The training occurs for 10 epochs over this batch of transitions with randomly shuffled mini–batches of 32 transitions. The value function is trained in a similar fashion using λ–returns. All our experiments are performed for 61200 transition steps (equivalent to 1200 episodes) corresponding to about 1 hour of robot wall time.

3.2 Training on Fixed Torque Ranges and Different Scheduling Techniques

As discussed in the previous section, we train our networks on the Dynamixel Reacher task, for 1200 episodes, using different approaches (for brevity we represent the torque scaling factor used during training from episode number a to b as $\tau_{a \to b}$):

Fixed Max Torque Range: We train the PPO agent on the Reacher task for four different values of the max torque range $\tau_{1 \to 1200} \in \{100, 200, 300, 400\}$, corresponding to method names *T*–100, *T*–200, *T*–300, and *T*–400.

Scaling Schedule Method: For the scaling type of scheduling method, we experiment with four different types of schedules

1. Schedule 1 (*S*–1): This is a linear schedule in which we increase τ by 100 after every 320 episodes, i.e. $\tau_{1 \to 320} = 100$, $\tau_{321 \to 640} = 200$, $\tau_{641 \to 960} = 300$, and $\tau_{961 \to 1200} = 400$.
2. Schedule 2 (*S*–2): This is a quadratic schedule in which τ is doubled after every 400 episodes, i.e. $\tau_{1 \to 400} = 100$, $\tau_{401 \to 800} = 200$, and $\tau_{801 \to 1200} = 400$.
3. Schedule 3 (*S*–3): In this schedule, we keep τ at 100 for most of the time and then change it to 400 for a short time during the end of the training. Specifically, $\tau_{1 \to 920} = 100$ and $\tau_{921 \to 1200} = 400$.

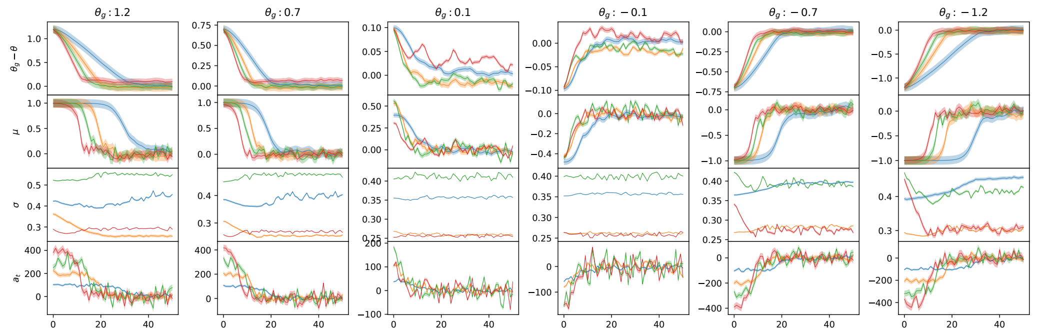

Figure 3: The distance from goal, mean, standard deviation, and final (torque scaled) action of fixed torque policies against the episode timesteps for six different goal positions. This plot shows the performance of the converged policy for each of the variants. θ refers to the present motor position, and θ_g refers to the goal (target) position. **Color Scheme:** **blue** corresponds to T–100, **orange** corresponds to T–200, **green** corresponds to T–300, and **red** corresponds to T–400.

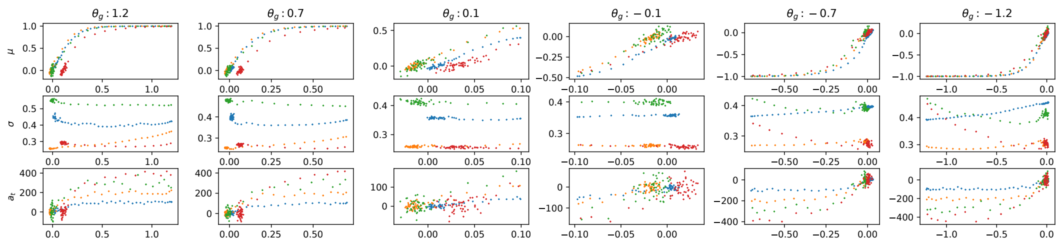

Figure 4: The mean, standard deviation, and final (torque scaled) action of the fixed policies against the distance from the goal states for six different goal positions. The plot shows the performance of the converged policy. Color scheme is same as used in Figure 3.

4. Schedule 4 (S–4): This schedule gradually increases the τ from 100 in the first episode upto 408 in the last episodes. τ is increased by 22 after every two episodes. This way, there is a small jump in the action values gradually over time.

Clipping Schedule Method: We experiment with a single type of schedule for this method. We increase the action clipping range from 100 to 400 by incrementing it by 100 units after every 320 episodes, i.e. $\texttt{clip}_{1\to320} = 100, \texttt{clip}_{321\to640} = 200, \texttt{clip}_{641\to960} = 300$, and $\texttt{clip}_{961\to1200} = 400$.

4 Results and Discussions

4.1 Experimental Results

We show the average returns (over single batch size $\equiv$ 40 episodes) against the transition steps for the policies learned by fixed torque range and different scheduling techniques in Figure 2. The plots show mean performance along with shaded standard error across five independent runs for each variant. The graph on the left shows the results for the fixed torque ranges. The graph on the right shows the results for four different scaling schedule methods and one clipping schedule method as described previously in Section 3.2.

We show the mean values (along with standard deviation) for various metrics across the learning phase of different scheduling methods and fixed torque variants in Table 1. The mean and standard deviation were calculated by taking average across 62k transition steps and five independent runs for each variant. The metric "$|A_{t+1} - A_t|$" refers to difference between the action of the policy (before scaling by a torque factor) at two consecutive timesteps. The metric "$|\tau A_{t+1} - \tau A_t|$" refers to the difference between actions (scaled by the torque factor) at two consecutive timesteps in an episode. We use these two metrics as a proxy for the jittering of the learned policy (a higher value corresponds

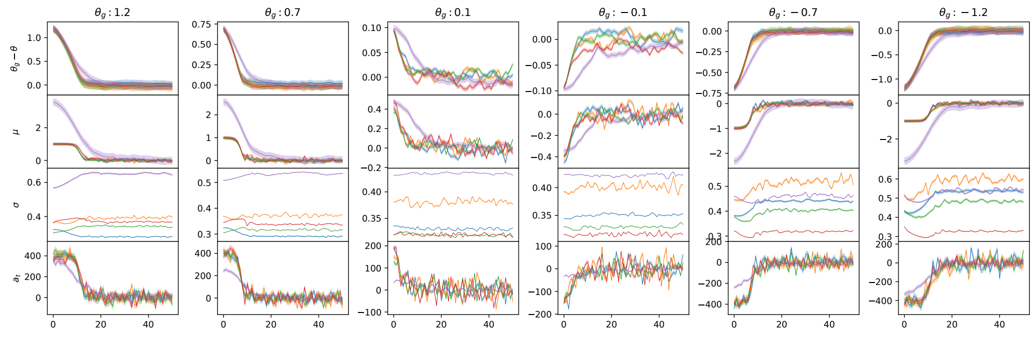

Figure 5: The distance from goal state, policy mean and standard deviation, and final (torque scaled) action of scheduled policies against the episode timesteps for six different goal positions. **Color Scheme:** **blue** corresponds to *S*–1, **orange** corresponds to *S*–2, **green** corresponds to *S*–3, **red** corresponds to *S*–4, and **purple** corresponds to *CLIP* scheduling technique.

to a higher jittering). Note that all the scheduled policies have a final τ that converges to 400 (408 for the gradual increasing schedule 4).

We show the average final performance of all the methods on the reacher task with six different target angles in Figure 3 (for four fixed torque ranges) and in Figure 5 (for five different scheduling methods). In the plots, each of the six column represents the different goal position (in radians) and the rows correspond to $\theta_g - \theta$: the difference between the target and present location, μ: the network output for mean of the policy distribution, σ: the network output for standard deviation of the policy distribution, and a_t: the final (torque scaled) action of the policy. These plots were generated by sampling five different episodes for each of the five policies learned corresponding to five independent runs[1] of a single policy type. As a result, the graphs show mean and standard error over 25 runs, for each target position.

We show the μ, σ, and the final (torque scaled) action for the fixed torque policies against the distance from the goal state in Figure 4. The horizontal axis in all the sub–graphs correspond to the distance of the motor's current angle to its target angle. The graphs were generated similarly by averaging over 25 runs as for Figures 3 and 5.

4.2 Discussion

We observe that amongst the policies with fixed torque, policies with higher torque range obtained better final returns than the policies with smaller range, as shown in Figure 2. We believe the main reason for this is that policies with higher torque range are able to get to the goal faster as evident from Figure 3. Interestingly, *T*–400 had higher returns than *T*–300, but from Figure 3, we can see that *T*–400 was unable to stabilize on the goal (i.e. it swiftly goes near the goal state and jitters there). This leads us to conclude that maximizing our chosen reward function may not necessarily lead to the best performing practical control policy. Further, from Figure 2, we can see that in general higher torque range policies are also able to converge faster.

Again from Figure 2, we find that scheduling methods are able to reach a final performance which is atleast as good as the policies directly learned with high fixed torque ranges (compare the policies on the right graph with *T*–400 on left). This along with the fact that scheduled policies spend less time in higher torque ranges (e.g. *S*–3 spends just a quarter of its time exploring 400 torque range), enable them to avoid aggressive exploration during the initial phase of learning (as seen from the $|\tau A_{t+1} - \tau A_t|$ column in Table 1: *S*–3 has a value of 114 compared to a value of 234 for *T*–400). Further, from Figure 5, we see that the final actions taken by each of the scheduled policies are quite similar; so are the final returns accrued by them. Thus it is useful to select schedules which spend smaller amounts of time learning in higher torque ranges.

[1]For 200 and 400 torque ranges, we plot the results for 4 runs out of the 5 we performed. Each of them had one run which wasn't able to reach near to the goal state that was skewing the performance of all the runs in Figure 3.

Qualitatively, from Figure 5, we see that clipping policy is able to increase its μ (due to not having the `tanh()` function), and thus produce a higher final action; whereas scaling schedule policies have a constrained μ and are able to produce higher final actions with the help of the torque scaling factor.

We now provide explanation for why the scaling technique works. From Figure 4, we see that for different states (represented by different distances from the goal position on the horizontal axes), all the policies (corresponding to different torque ranges) give a very similar output for μ. This similar μ combined with their different torque scaling factors τ gives rise to their actions being scaled differently. Now assume that we have learned a good policy with $\tau = 100$, and we increase τ to 200 while keeping the weights of this policy same. Then from Figure 4, we can argue that the resulting policy (having weights corresponding to 100 torque factor but with the current scaling factor τ = 200) is quite similar to the policy learned directly with 200 torque scaling factor. As a result, when we scale already learned policies in a smaller torque range, they get transformed into correspondingly good policies in the higher torque range.

From our experiments we see that the policies have more difficulty in reaching nearby locations (exercising fine grained control) as shown by excessive oscillations in μ values for goal states near 0 radians in Figure 4.

We also implemented the clipping method using a different PPO implementation (with a state independent σ); however that was unable to learn well. Although other methods (fixed torque ranges and scaling schedule methods) worked fine with that same implementation. We obtained very similar results to what are shown here with that implementation as well. Interestingly, despite running a single motor for about three–four hours with a $\tau = 1000$ we were unable to get it to overheat and shut down, contrary to what was mentioned in (Mahmood et al., 2018b); we found that the temperature plateaued at around 70 degrees Celsius while the maximum allowable temperature is 80 degrees Celsius.

5 Conclusions

Our main motivation in pursuing this work was that RL methods were unable to outperform classical control methods in a recent study on physical robots. We hypothesized that the reason for this was that RL policies were not allowed to act on the full action space because of their aggressive exploratory behavior which can be physically damaging to the robotic components. Our goal was then to ascertain the truth of this hypothesis and come up with a method which is able to circumvent this problem on the Dynamixel Reacher task.

We were able to experimentally verify that higher torque ranges did lead to better performance. After that we proposed two different types of scheduling methods that sequentially increase the action space of the agent allowing it to utilize its full motor capabilities. We found that scheduling approaches (both scaling and clipping) were able to obtain performance similar to what we could directly learn using high torque range actions. Further, since scheduling approaches spent less time acting in higher torque ranges while learning, they exhibited better behavior (in terms of motor temperature, action jittering). However, we could not conclusively show that our proposed scheduling methods would be able to alltogether avoid dangerous exploratory behavior by transferring prior learned behavior to larger action spaces (infact we suspect that they don't).

Finally, we concluded that scaling is an empirically sound method for increasing the action space of a PPO policy on the reacher task, which results in reasonable policies, i.e. the resultant policy is well behaved and doesn't significantly degrade in performance. As a result, we have a method of transferring a good policy (learned on a smaller action range) to a higher action range where it can tune itself in a small amount of time. The reacher task for a single motor is an important problem because many complicated systems (like a robotic arm) can be viewed as a composition of multiple reacher tasks.

Future work can include learning to automatically schedule a sequence of tasks using a teacher agent which optimally tunes the complexity of the tasks. Apart from this, of particular interest would be experiments that combine the scaling schedule method with functions other than `tanh()`, like linear or sigmoid functions. We also believe that action cycle time and the kind of policy gradient algorithm used can have an impact on the scheduling approach as well.

Acknowledgments

The authors would like to acknowledge funding from the Canada CIFAR AI Chairs Program. Shivam and Homayoon would like to thank Craig Sherstan and their wonderful fellow students for useful discussions and suggestions.

References

Garcia, J., Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research, 16*(1), 1437-1480.

Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Mahmood, A. R., Korenkevych, D., Komer, B. J., Bergstra, J. (2018a). Setting up a reinforcement learning task with a real-world robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems.*

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., Bergstra, J. (2018b). Benchmarking reinforcement learning algorithms on real-world robots. *Annual Conference on Robot Learning.*

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347.*

Sutton, R. S., Barto, A. G. (2018). *Reinforcement Learning: An Introduction,* Second Edition. MIT Press.