Reinforcement Learning for Short Range Path Planning in Indoor Environment

Prasanna Natu
MS Robotics Engineering
pvnatu@wpi.edu

Shreya Bang
MS Robotics Engineering
srbang@wpi.edu

Shambhuraj Mane
MS Robotics Engineering
samane@wpi.edu

Vaibhav Kadam
MS Robotics Engineering
vkadam@wpi.edu

I. Introduction

Path planning of robot in an indoor environment is a crucial task of autonomous navigation, where the robot subsequently plans path from its initial state to a target state in a grid or map. Many of the path planning algorithms are not actually used on real-time robots due to its computational demands and high dimensional problems. As the advancement in Robotics, Deep Reinforcement Learning (DRL) provides a powerful tool to provide optimal path planning in deterministic indoor environments. In comparison of classical path planning algorithms, reinforcement learning (RL) based approaches have received significant attention in the recent past due to the success of the deep learning.

We provide a report for the RL based path planning for short range given a target state (pose). The robot is equipped with a 2D Lidar sensor, RL-based path planner provides optimal paths to given target pose independent of map environmennt. Our contribution is to plan paths using Proximal Policy Optimization (PPO) and Actor Critic (A2C) implementation.

Fig. 1. Gazebo Hospital Environment

II. Literature Review

Reinforcement learning (RL) has emerged as a promising approach for robot path planning and navigation in recent years. Compared to traditional path planning methods like A*, RRTs, etc., RL does not require an explicit map of the environment and can learn policies that generalize well to new environments.

Q-learning is one of the most commonly used RL algorithms. Faust et al. (2018)[1] proposed a hierarchical planning method for long-range navigation tasks, that combines

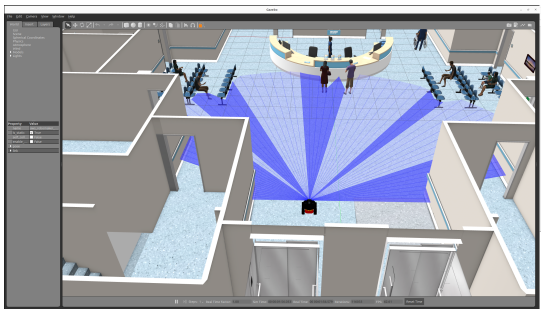

Fig. 2. Environment with Pioneer 3AT, a 4-wheel differential drive agent

sampling- based path planning(PRM) with RL agents to complete tasks in very large environments. Which showed that PRM-RL expands the capabilities of both RL agents and sampling-based planners. The benefit here is that, although the RL agent depends on the robot and the goal, it is not dependent on the environment and can be used to create roadmaps for a variety of contexts.

Policy gradient methods like PPO can handle continuous action spaces and have also been applied for path planning. Tai et al. (2017)[2] used PPO to train a policy for navigation in simple 2D environments with continuous control of velocity and steering angle. The policy was trained using only laser scan data as input.

Imitation learning from expert demonstrations can speed up training of navigation policies. Pfeiffer et al. (2017)[3] used behavior cloning with CNNs to imitate an A* planner and then fine-tuned the policy with RL. Demonstration was done that the learned navigation model is directly transferable to previously unseen virtual and, real-world environments. The method outperformed learning from scratch.

The question of whether policies developed in simulation using deep reinforcement learning could be used in the actual world was explored by Tzeng E. (2017)[4]. As compared to policies trained using sparse real-world data, he shows in his research that vision systems trained on simulated data and adapted using our technique may be utilized to initiate deep visuomotor policies that achieve greater performance on real-world tasks.

Key challenges for RL navigation include sample efficiency,

sim-to-real transfer, and integration with traditional methods for complementary capabilities. Hybrid methods that combine learning and classical planning are an active area of research.

III. Methodology

A. Environment Setup

In our study, we focus on training a reinforcement learning (RL) agent to navigate short distances within a simulated environment, specifically a hospital setting, without prior knowledge of the space's large-scale topology. This integrates the RL agent into existing path planning algorithms as a local planner. The environment is a 3D representation of a hospital, complete with various objects and human figures, and is simulated using Gazebo, which integrates seamlessly with ROS2 Humble. Our chosen robotic platform is the Pioneer 3AT, a four-wheeled robot equipped with a 180° laser rangefinder providing 61 distance measurements ranging from 0.08 to 10 meters. This setup allows the robot to determine its position in space relative to a fixed Cartesian system and execute movement commands based on specified angular and linear velocities.

The RL agent's objective is to navigate to short-range goals (up to 10 meters) within this simulated environment, avoiding obstacles and fulfilling task constraints. The agent operates under specific conditions: it limits its angular speed to [-1,1] rad/s and linear speed to [0,1] m/s, with no backward motion. Collision detection is crucial and is determined when any laser measurement falls below 0.25 meters, considering the robot's geometry in relation to the laser's position. The target is defined as a point in the (x, y) space, and task completion is acknowledged when the robot comes within 0.40 meters of the target, a distance that accounts for the laser's position ahead of the robot's center and an additional safety margin.

*1) **OpenAI-Gym Setup**:* The RL agent is developed and trained using OpenAI Gym, an open-source toolkit that provides the necessary framework for creating a custom RL environment. The 3D models of the hospital and the Pioneer 3AT, slightly modified to include the 180° laser. This setup forms the foundation of our study, where the RL agent learns to navigate effectively in a static environment, representative of real-world conditions in a hospital setting.

Observation Space: Defined as a dictionary, it includes:

- Polar coordinates of the goal (r, θ), with $r \in [0, 60]$ meters and $\theta \in [-\pi, \pi]$ radians.
- LIDAR sensor data, o_i for $i = 1, \ldots, 61$, from the 180° LIDAR sensor with a depth of up to 10 meters.

Action Space: A 2-dimensional continuous array:

- First element: Linear speed, within [0, 1] m/s.
- Second element: Angular speed, within [-1, 1] rad/s.

Reward Function: Implemented as a sparse reward strategy, the reward r_t at time t is defined as: x'

$$r_t = \begin{cases} 1 & \text{if the goal is reached} \quad (r < 0.4), \\ -1 & \text{if a collision occurs} \quad (\min_i o_i < 0.25), \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

This design penalizes collisions and rewards goal attainment, fostering a risk-neutral policy.

Step Method:

1) De-normalize the action to send the proper command to the robot
2) Send velocity command to the robot
3) Spin the node to allow sensor updates via callbacks
4) Transform coordinates to get polar coordinates of goal relative to robot
5) Get updated observation with laser readings and robot location
6) Get additional environment info
7) Compute reward
8) Check if episode is finished based on:
   - Reaching goal (distance < 0.4 m)
   - Collision detected (laser reading < 0.25 m)
9) Return observation, reward, done flag, and info dict

Reset Method:

- Sample new start state:
  - Randomly generate the initial state, including robot pose and target location, for the new episode based on a distribution. This introduces variety across episodes.
- Reset simulation:
  - Move the robot and update other entities in the simulation to the new start state. This is done by calling services and allowing time for the reset to fully propagate.
- Initialize observation:
  - Spin for sensor updates, transform coordinates, and construct the observation using the reset state. This creates the starting observation the agent sees.
- Return observation:
  - The observation is returned to the RL training algorithm so that the next episode can begin from the sampled start state.

B. Learning Algorithms

We implement two methods, Proximal Policy Optimization (PPO) and Actor Critic (A2C), to see which one works better for making a robot move in a fixed environment. We're using LiDAR data to help the robot navigate in real-time.

Proximal Policy Optimization (PPO): PPO has been designed to ensure that when updating the policy, the new policy doesn't deviate too far from the previous one. This is beneficial because drastic changes in the current policy parameters can lead to a significant drop in model performance, requiring many timesteps to recover the previous state. PPO is an on-policy algorithm applicable to both discrete and continuous action spaces. It comes in two variants: PPO-clip and PPO-penalty. We are concentrating on the first variant.

Note that the expectation is calculated over the state-action pair (a, s), where "a" represents the action sampled from the previous policy. A singular update is executed using

Stochastic Gradient Descent (SGD), offering an alternative to traditional policy gradient descent. SGD significantly reduces computational effort by taking multiple smaller steps to adjust policy parameters. In each of these smaller steps, the gradient is computed using a mini-batch, a smaller sample, rather than considering all the sampled trajectories.

Let $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$. The the objective of PPO is the following equation:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(\text{r}_\text{t}(\theta), 1-\epsilon, 1+\epsilon)\Big)\hat{A}_t)\right] \tag{2}$$

1) Positive Advantage ($A^{\pi_{\theta_k}}(s,a) > 0$): If the advantage of a state-action pair (a, s) is positive with respect to the old policy π_{θ_k}, it is desirable to increase the probability of that action when in states. In doing so, the probability ratio $\dfrac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ will be greater than one.

   The second part of the minimum function sets a limit on how much the probability ratio for a given state-action pair can increase. If this ratio exceeds $1+\epsilon$, the clipped objective is chosen because it is lower than the unclipped one. This ensures that the growth in the likelihood of the action is constrained. Conversely, if the new parameters θ result in a policy that decreases the probability ratio even below $1-\epsilon$, the unclipped term is selected. This decision is logical as it leads to a lower value of L, opposing the maximization goal of the algorithm. In such cases, the reduction in the probability ratio is not restricted.

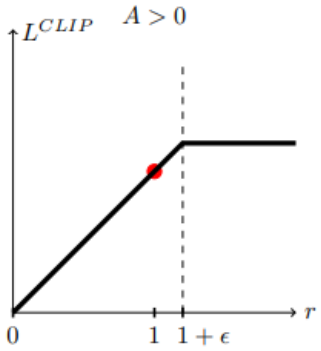

Fig. 3. L-function with respect to the probability ratio when the advantage A is positive

2) Negative Advantage ($A^{\pi_{\theta_k}}(s,a) < 0$): It is convenient to reduce the probability of the action a in state s when the relative advantage is negative.
   Similarly, in this scenario, the clipped objective constrains the reduction in the likelihood of the state-action pair. When the probability ratio of the new policy falls below $1-\epsilon$, the second term of the minimum function is chosen, ensuring a controlled decrease in the action likelihood. Conversely, if the ratio is positive, the unclipped term is selected, allowing the penalization of such a policy to be unbounded. It's important to note that, given the negative nature of the advantage, increasing the likelihood of the action is undesirable.

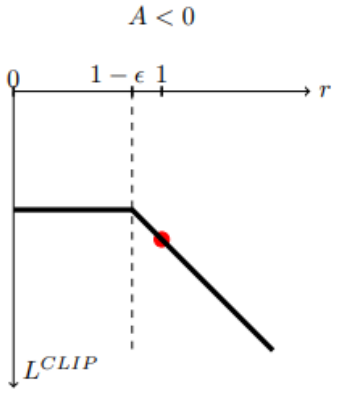

Fig. 4. L-function with respect to the probability ratio when the advantage A is negative

To ensure that the agent explores new states of the environment, an entropy bonus is incorporated into the loss function L. The influence of this component can be controlled by tuning the entropy coefficient. In essence, a higher coefficient increases the likelihood that the agent favors a random action rather than sampling from the current policy. As training progresses, the entropy bonus diminishes, ensuring that the agent's behavior gradually converges towards its existing policy over time.

Further, to update the policy, we need information about the on-policy value function. Since it involves an expectation, we can calculate it using a suitable estimator. In the case of PPO, an estimator with parameters is utilized, and these parameters θ are adjusted at each step of the algorithm. The pseudocode for the algorithm is provided below:

Algorithm 1 Q Actor Critic

Initialize parameters s, θ, w and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1 \dots T$: **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$
  Then sample the next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_\theta \log \pi_\theta(a|s)$; Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$
  and use it to update the parameters of Q function:
    $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s,a)$
  Move to a $\leftarrow a'$ and s $\leftarrow s'$
end for

Algorithm 1 PPO-Clip

1: Define initial policy parameters θ_0
2: **for** k=0,1,2,... **do**
3: Collect a set of trajectories $D_k = \tau_i$
4: Compute the rewards $R(\tau_i)$ for each episode i
5: Compute advantage estimate $\hat{A}^{\pi_{\theta_\kappa}}$ with the current value function V_{ϕ_κ} (using any advantage estimation method)
6: Update policy parameter by maximizing using the Gradient Descent
7: Update the value function parameters ϕ by regression on mean-squared error (using some gradient descent algorithm)
8: **end for**

Advantage Actor-Critic Network (A2C):

We implement actor critic network as it has following advantages:

- Sequential Decision Making: Mobile robots often operate in dynamic environments where they need to make a sequence of decisions over time to achieve a goal. A2C, being a sequential decision-making algorithm, is designed to handle such scenarios.
- Combining Value and Policy Learning: A2C combines the advantages of both value-based methods (e.g., Q-learning) and policy-based methods. It maintains both a value function (critic) and a policy (actor), allowing for a more stable and efficient learning process
- Efficient Exploration-Exploitation Trade-off: A2C includes a mechanism to balance exploration and exploitation. This is crucial for a mobile robot in an environment where it needs to explore to discover optimal paths but also exploit known information to achieve its goals efficiently.
- Adaptability: A2C is adaptable to different types of problems and can handle continuous action spaces, making it suitable for motion planning tasks where the robot's actions may be continuous.

C. Training of the Agent

The training process of the agent is illustrated below.

1) Increasing simulation speed: In general, RL is very sample inefficient. Many time steps are needed to get the ideal course of action. It therefore seems prudent to accelerate the simulation time as much as feasible. The training algorithm can handle more time steps in the same amount of time in this way. Two world file parameters can be changed in Gazebo to modify the simulation time:

- Max step size (default value: 0.001 seconds), determining the time in the simulation to be simulated in one step
- Real time update rate (default value: 1000 Hz), which determines the minimum time period after which the next simulation step is triggered.

But you can't keep raising this parameter indefinitely. It's obvious that the algorithm's computational effort is limited by the amount of processing power the computer can handle. Our initial attempt at parameter adjustment resulted in a 15x quicker training pace for the first agent, with the maximum step size increased to 0.05 seconds and the real-time update rate to 3000 Hz. Moreover, the graphics output was disabled during training using a special Gazebo start file in order to lower the computational load on the system.

Ubuntu 22.04 LTS is installed on the machine that runs the training algorithm, a 12th Gen Intel® CoreTM i9-12900H × 20 with 16 GB of RAM.

2) Training on a simplified environment: To assess the efficacy of the chosen reward function, a simplified environment with no LIDAR samples was analyzed at the start of the training phase. The goal of such a simplified setup is to debug undesirable behaviors in the Gym environment and ensure that the chosen reward function results in the desired policy. Later, more complexity can be added to the problem so that more sophisticated agents can be trained. The changes made for the simplified environment are listed below.

- The observation space contains only the polar coordinates of the target (r, θ^T, FA) relative to the robot.
- An episode ends if the robot reaches the target $(r > 0.4)$ or if it goes 3.5 meters farther than the target $(r > 3.5)$, creating a navigation limit.
  The reward function only returns 1 when the robot reaches the target, in all the other cases it returns 0.

The maximum number of steps is set to 1000 timesteps before beginning the training process. Given that the agent can travel up to 0.1 meters in a single step and that the agent is placed 3 meters away from the obstacle, the robot could complete the task in as few as 30 steps. As a result, 1000 steps per episode are more than enough to complete the task successfully.

It has been observed that the trained robot's behavior corresponds to intuition. In fact, the agent points directly to the target and arrives there without changing direction. All of this is evidence that the reward function selected can effectively lead to the desired policy. The following step is to train an agent by incorporating laser readings into the state.

3) Training for the standard task: The objective is to train an agent that can do the earlier described task, in which laser readings are additionally included in the state and a collision results in a negative reward. At this point, the procedure is broken down into multiple phases where the task's complexity gradually rises. By doing this, it is ensured that the training process is better understood and that any potential problems are easier to find.

The results and graphs mentioned in the report are from the first generation agent.Second-generation agent: We are planning to train agent on a more heterogeneous set of episodes that recognizes different patterns of obstacles through its laser readings. Many different locations will be defined during the Gym environment's initialization by specifying the coordinates of the initial robot position and the target. Risk-seeker agent: We are planning to make the robot less sensitive to far away obstacles, a different

Step	Error	default
learning rate	0.0001177	0.0003
n steps	2279	2048
gamma	0.9880615	0.99
clip range	0.1482	0.02
gae lambda	0.9435888	0.95
ent coef	0.0000968	0.0
vf coef	0.6330533	0.5

TABLE I
HYPER-PARAMETERS FOR FIRST GENERATION AGENT

reward function will be defined for the Risk-seeker agent.

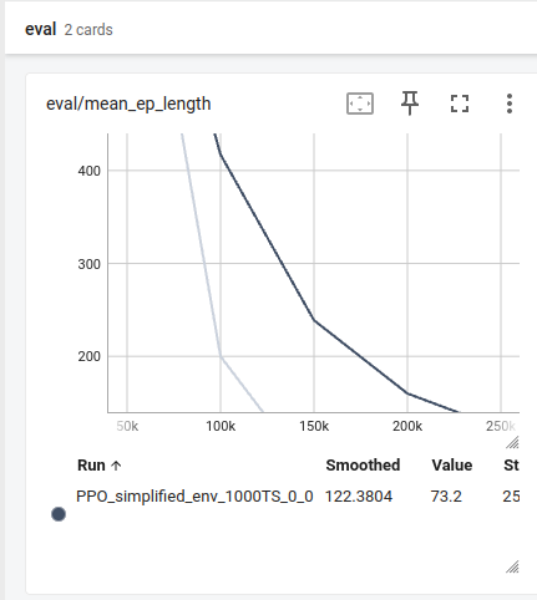

Fig. 5. Mean Episode Length for Simplified Environment

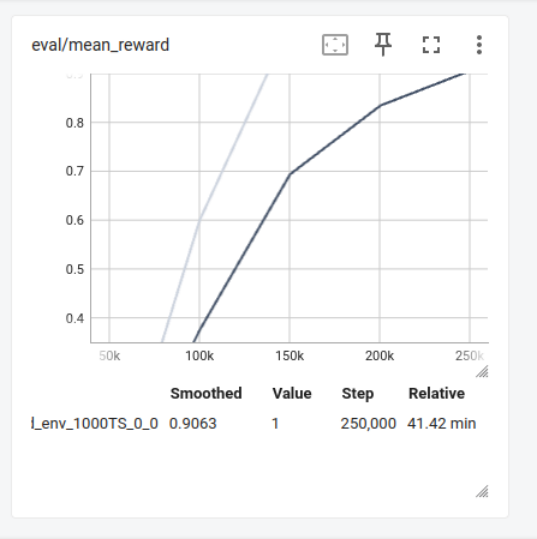

Fig. 6. Mean Reward for Simplified Environment

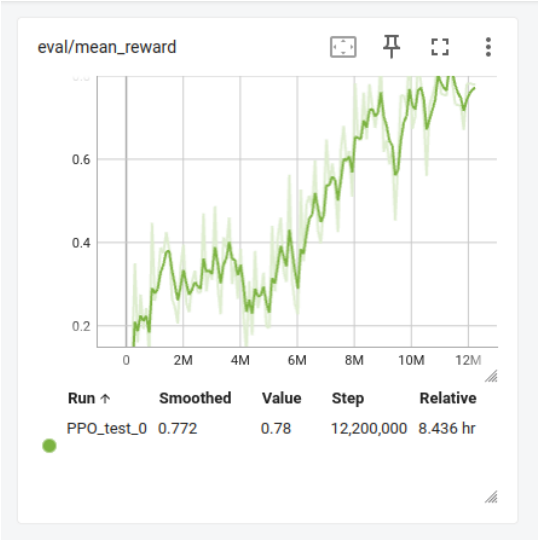

Fig. 7. Mean Reward for Complex Environment - PPO

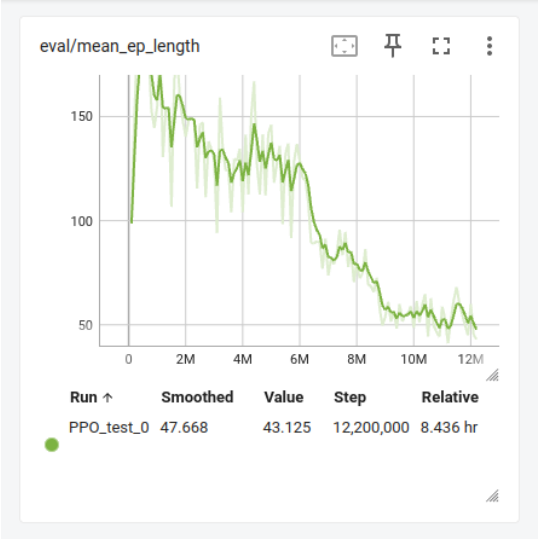

Fig. 8. Mean Episode Length for complex Environment - PPO

IV. APPLICATIONS

1) Mobile Robots in Healthcare: Robots used in healthcare settings for tasks like medication delivery and patient assistance can benefit from RL-based navigation to ensure safe and precise movement in clinical environments.
2) Warehouse Automation: Autonomous robots in warehouses and fulfillment centers use RL to efficiently navigate and pick items from shelves, optimizing the logistics process.
3) Autonomous Vehicles: Self-driving cars and autonomous drones use reinforcement learning for navigation. These vehicles learn to make decisions like lane changes, speed adjustments, and avoiding obstacles based on sensory data from cameras, LIDAR, and other sensors.

Fig. 9. Mean Episode Length for complex Environment - A2C

Fig. 10. Mean Episode Length for complex Environment - A2C

V. Result

We discuss the short ranger path planning results here for PPO and A2C. In-case of PPO in training performance we got mean reward more than 0.8 as shown in the Fig. 7 for the simplified environment and the mean episode length converged to 73.2, which states that agent was successful in not only planning the path from spawn location to designated goal location but also finding the optimum path eventually.

For complex environment the PPO algorithm, initially for 5 million training steps gave unsatisfactory results of 0.4 rewards but after around 6 million training steps the agent showed good rise in the reward and after 12 million training steps final reward was of 0.78 and mean episode length converged to 43, which was better than the result from simplified environment. This demonstrates that with more number of training steps this algorithm can perform better with finding optimum path between start to goal. Which illustrates that PPO is suitable for short range path planning application.

On the other hand, A2C has not shown promising results in terms of planning path to goal location though the agent learned to prolong the episode length and navigate in the environment avoiding obstacles. Fig. 9 shows that the mean episode length value was 300 (the maximum episode length) after the 8.4 million training steps and the reward which was initially negative till 3 million steps converged to 0 reward. The potential reason can be the sparse reward of attaining goal. The inference is supported by simulation as well.

VI. Conclusion

We designed two environments - a simplified and a complex one, with the later featuring more obstacles than the former. In our experiments, the Proximal Policy Optimization with Clip (PPO-Clip) algorithm demonstrated superior performance compared to the Advantage Actor-Critic (A2C) algorithm in both environments.

VII. Future Scope

Enhancing the agent's performance could be achieved through refining the reward function to be more specific or by incorporating a more diverse range of episodes in the training setting. However, owing to the constraints of the present study, additional research is necessary to pinpoint an optimal learning environment configuration.

References

[1] Faust, A., Ramirez, O., and Fiser, M., "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning," *IEEE International Conference on Robotics and Automation (ICRA), 2018*, arXiv:1710.03937, Nov 2017.

[2] Tai, L., Paolo, G., and Liu, M., "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2017.

[3] Pfeiffer, M., Schaeuble, M., and Nieto, J., "From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017.

[4] Tzeng, E., Devin, C., and Hoffman, J., "Adapting Deep Visuomotor Representations with Weak Pairwise Constraints," *under submission*, arXiv:1511.07111, May 2017.

[5] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O., "Proximal Policy Optimization Algorithms", https://arxiv.org/abs/1707.06347v2.

[6] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783