Reproducibility of RL in Real World Robotics Task Training

Team 6: George Muhn, Kylel Scott, Jacob Sindorf

Abstract—**Robotics are becoming increasingly usable in commercial and industrial settings to perform monotonous/repetitive tasks. However, the controls architecture for a system developed for basic actuation and minimal environmental cognition feedback is very different than one that is aware of its environment and uses this intuition to base current and future decisions off of. As this trend continues, there is demand in the space for improvement of autonomy and robustness of these systems. Our team investigated the practical implementation of different Reinforcement Learning (RL) algorithms and approaches on an UR5 robotic manipulator. The team found that benchmarking different solutions provided a sense of practical implementation of our studies, and we believe this work also serves as a resource with tangible juxtaposition of different RL approaches; analytically explaining the pros and cons of optimal agent creation, training, and implementation.The team also evaluated the replication of some of the most state-of-the-art algorithms and development environments for real-world robotic implementation, that being SenseAct. As replication is one of the most difficult things to accomplish in this novel space, the successful implementation of our project supports the viability and usability of SenseAct as a launching platform for Reinforcement learning deployment on their supported robotic platforms.**

I. Introduction

Utilizing RL algorithms and fine tuned hyperparameters, robotic reaching tasks can be simplified and expanded for multiple use cases.The team sought to investigate the usability of common online Reinforcement Learning software packages, while replicating the results shared from more novel real-world implementations. The team first began by investigating a commonly used library, OpenAI; and their two degree-of-freedom environment, Reacher-2D. This environment was a useful introduction into the current benchmarked solutions within the field of online robotic manipulator training. However, the team was specifically looking to pursue a tangential adaptation of this approach by implementing the task offline to a real-world robotic arm. This entailed working with an environment that did not use a simulator in order to train a robot in real time.

While this approach can be seen as counter-intuitive to solving the training-time issues that online simulations were created to mitigate, the team found this strategy to be one of value given its simplicity. It was observed that researchers often show reluctance towards non-simulation-based training because the acquisition of physical robots can be expensive, or the maintenance that running the robot through different training sessions can drastically increase the cost of ownership and depreciate the robot's value. Additionally, the inconvenience entailed in training real agents, pertaining to communication errors, physical constraints, and time-based limitations, further encourage the use of online simulations. The team decided to investigate this stigma given that a UR5 robot in working condition was available. While offline training does have its challenges, the team understood that running online simulators can also carry complications; those being in dependency errors, version conflicts, and limited software support being available. This research seeks to find out not only how feasible it would be to deploy a real-world training task to a UR5 robot over using a simulation, but also how usable were the current resources available for the integration.

Following the research conducted on online training frameworks, the team pursued real-world training of a 2-Degree of Freedom (DOF) UR5 arm using a supported fork of Sensact from Kindredresearch [1]. The team was able to successfully recreate the work in the paper, assisted by the Sense Act Experiments [2], by deploying the project onto the UR5 arm for real-world training. Using convergence dynamics for the TRPO (Trust Region Policy Optimization) and PPO (Proximal Policy Optimization) Reinforcement Algorithms with the hyperparameters that were used in the paper. Overall, implementation was successful, even though the team experienced a number of complications with initial communication with the robot and the ROS architecture. These findings are shown in the sections to follow.

II. Methods

A. 2DOF (UR-Reacher-2) Setup

The team proposed a simplified method to replicate the results from Lynnerup et al [2] using their findings on hyper parameters and more streamlined versions of the SenseAct code from Mahmood et al [1]. Using RL in real time applications has become a growing application; however the barriers involving software, hardware, communication and documentation hinder new results. The paper by Mahmood et al [1] has provided a way to fully train a UR5 Robot with reinforcement learning without the need for a virtual environment. In order to encourage other researchers, Mahmood et al have provided their code in a SenseAct Github, but the documentation brings up many challenges that Lynnerup et al address in their modified SenseActExperiments Github. Those challenges mainly include the github and required inputs that are not provided or made clear by Mahmood et al. However, even with a streamlined approach, there are still gaps in the software setup that are not made clear which can lead to unnecessary debugging. This work describes the necessary set up of the RL problem on a UR5 robot with descriptions of the process and RL algorithms also provided.

B. Algorithms

In order to train the UR5 arm to reach random points in space, it requires the use of different RL algorithms to test performance. Mahmood et al reference four main algorithms including TRPO, PPO, Soft-Q, and Deep Deterministic Policy Gradient (DDPG). Using DDPG and Soft-Q are not possible without hard coding them as they are not provided by Mahmood et al. Thus the focus of these experiments, as well as Lynnerup et al, are on the performance of TRPO and PPO. The TRPO and PPO algorithms are set up easily through OpenAI Baselines, using their packages allows the research community a standard baseline to replicate, refine and identify new ideas. Baselines makes it easy to change, adapt and experiment with different hyper parameters to explore new and interesting results. [3].

1) Trust Region Policy Optimization: (TRPO) was developed in order to optimize large nonlinear policies. This allows for optimization of neural networks used in controlling high-dimensional state space used in robotic locomotion [4].This RL algorithm is a great option for controlling the motion of a 6 degree of freedom robot arm. As described by Mahmood et al, the policy is optimized through iteratively solving this optimization problem. [1]

$$\underset{\theta}{\text{maximize}}\, \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{\text{old}}}(s,a)\right] \qquad \text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s) \,\|\, \pi_\theta(\cdot|s))\right] \leq \delta.$$

2) Proximal Policy Optimization: (PPO) is very similar to TRPO but replaces the KL-divergence constraint with a penalty term. PPO simplifies TRPO by using a first order derivative solution. PPO uses an actor-critic network to maintain two policies, the current policy and outdated sample collection policy [5]. Then using the difference ratio from the two policies creates a new objective function as shown. [1]

$$L_\theta^{CLIP} = \mathbb{E}_{s,a\sim\pi_{\theta_{\text{old}}}}\left[\min(r_\theta(a|s)A_{\theta_{\text{old}}}(a,s), \text{clip}(r_\theta(a|s), 1-\varepsilon, 1+\varepsilon)A_{\theta_{\text{old}}}(s,a))\right],$$

C. Software Requirements

Overcoming the software requirements was the biggest challenge in replicating the RL experiments. Documentation on software acts as a barrier to those looking to run RL on their own as it requires a lot of specific steps to complete the proper compatibility. To start, it is best to have a version of Ubuntu 64 bit installed (16.04 to 20.04). If your current machine is not Linux compatible, it is best to download a virtual machine (VM) such as VirtualBox. Getting Ubuntu on a VM is well documented online and resources can be found for assistance. Within the Ubuntu interface, python version 3.7 is required for many of the packages used. The most notable one being tensorflow version 1.15. OpenAI Baselines [3] must then be installed from the openai/baselines github by following the Ubuntu step under prerequisites. The readme on kindredresearch/SenseAct [1] github can be followed; however this is where the work done by Lynnerup et al is used. Install a compatible version of Docker onto the Ubuntu VM then under dti-research/SenseActExperiments [2] follow the steps on how to run it. Now under their code/experiments/ur5 section, RL can be ran if the UR5 is set up properly as mentioned in the next section below.

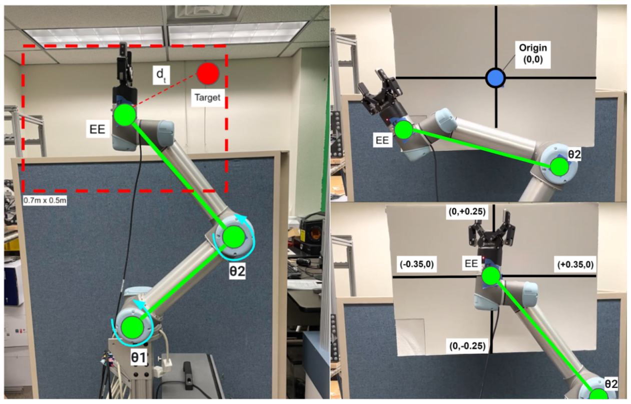

Fig. 1. UR5 2DOF Setup and depiction of boundary limitations and labels of components.

D. UR5 Requirements

Once the Software requirements have been met, the UR5 arm can be used. Calibration of the UR5 as well as factory setup are outside the scope of this paper. It is assumed the UR5 is fully functional. The first most important thing is to make sure the UR5 is running UR Software v. 3.3.4.310. Previous software such as this can be obtained from the UR website and installed easily onto the UR5. The device running Ubuntu and the UR5 must both have a separate Ethernet cord plugged into them and connected to a network switch that connects to the main Ethernet port for internet use. Troubleshooting and testing connection to the UR5 is well documented online. If a successful connection is established then on the UR5 tablet, under the internet section, an IP address should be shown. That IP address is required for robot connection and needs to be manually put into the code downloaded from the dti-research/SenseActExperiments whenever the IP address is seen.

E. UR5 Setup (2DOF, UR-Reacher-2 Task)

With the necessary software and UR5 requirements met, RL experiments can now be done. For the UR-reacher-2 task, the experiments follow the setup seen in fig. 1. There are two moving joints, the shoulder labeled θ_1, and the elbow labeled θ_2, and an end effector labeled EE. The EE is limited to a 0.7m by 0.5m rectangle and is always initialized to the center of the rectangle to an initial position labeled the origin with a value (0,0) as seen in fig. 1. This helps avoid singularities and ensures the system runs safely. Within the given boundary, a random point, labeled target, is formed and the robot learns to reach towards that target each episode. The euclidean distance between the EE and target is expressed as d_t and is the main value used in calculating reward.

In order to run RL it is imperative to describe the state dynamics, actions, states, and rewards. Here the action space,

$At \in A$, is a continuous value between (-0.3, 0.3) rad/s [1]. States, $St \in S$, are not directly observable, which is why the system receives observations. This is because the states are continuous and complex and are not directly known by the system. So the observations can be used to estimate the state with an observation vector consisting of joint angles, joint velocities, the previous action, and the vector difference between the target and the fingertip coordinates [1]. Rewards, $Rt \in R$, follow the equation $R_t = -d_t + exp(-100d_t^2)$, with the second term being the precision reward that encourages more precise movements to the target [1]. State dynamics follow the equations of motion from Lagrangian equations. Episodes for each run are 4 seconds long, where an episode consists of a new randomly generated target [1]. Prior to each episode, the UR5 returns to the origin of its initial configuration. All targets are generated within the boundary.

Now that the setup has been described, RL experiments using PPO and TRPO can be performed using the UR5 robot arm.

F. UR5 Setup (6DOF, UR-Reacher-6 Task)

Set up for the 6DOF expands upon the 2DOF problem. There are now 6 moving joints, labeled θ_1 to θ_6, and an end effector labeled EE in fig. 2. The EE is limited to a 0.7m by 0.5m by 0.4m box and is always initialized to the center with position (0,0,0). Both systems share the same reward function, which only depends on d_t, or the euclidean distance between the end effector and the target. The action space remains the same as each $\dot{\theta}$ value can take a continuous value between (-0.3, 0.3) rad/s. However, the increase in joints increases the state and observation spaces to include every joint.

III. Implementation and Simulation

A. 2DOF

Once the setup is complete and the correct software is installed on both the computer and UR5 arm, training and experimentation is ready to begin. Running the simulation begins with running a python script linked to a specific yaml file. The YAML is an added feature from the original SenseAct research, it provides an easier repeatable method of running experiments with the UR5 arm. The file organizes all the training hyperparameters, IP address, joint configurations and limits, and desired end effector position limits. This allows for experiments to easily be modified and tested; changing the degree of freedom, policies, learning rate, neural network size or steps per episode.

Each training iteration is evaluated as follows. The robot arm trains for a total of 150k time steps and each iteration is composed of 41 episodes of which take 4s of exploratory actions and reward observation. The reward is then averaged at the end of the 41 episodes. Hyper parameters are configured to match those found in [2] as they have identified the top five best hyper parameter configurations. These were identified through extensive testing and were made available for public use. This work covers the specific configurations, labeled 1 and

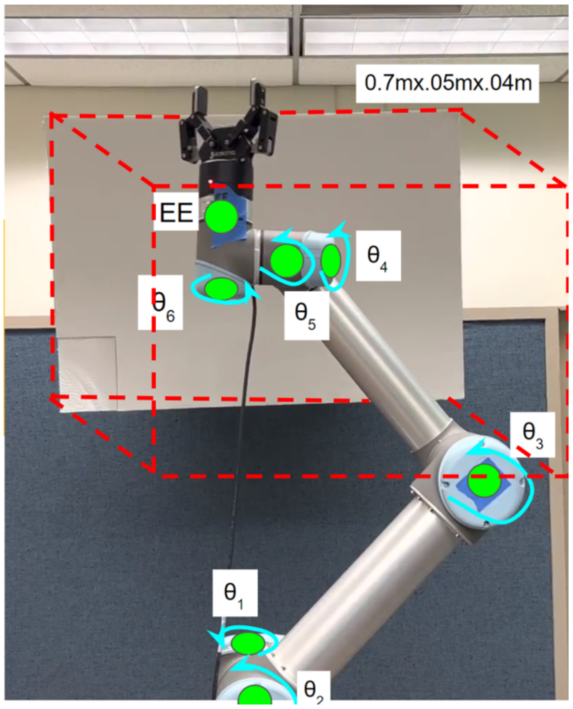

Fig. 2. UR5 6DOF Setup and depiction of boundary limitations and labels of components.

5 for both TRPO and PPO in the appendix of [2]. Specifically, the hyper parameters are as follows:

TRPO

#	Hidden Layer	Hidden Size	Batch Size	Step size	γ	λ	δ_{KL}
1	2	64	4096	0.00472	0.96833	0.99874	0.02437
5	4	32	2048	0.00163	0.96801	0.96893	0.00510

PPO

#	Hidden Layer	Hidden Size	Batch Size	Step size	γ	λ	Opt. Batch Size
1	3	64	512	0.00005	0.96836	0.99944	16
5	1	128	2048	0.00280	0.99924	0.99003	32

TRPO was trained 5 times with TRPO hyper parameter set 1; 2 runs finished completely while the other 3 failed part-way through training. One successful run was completed using TRPO hyper parameter set 5. The PPO was trained 4 times with PPO hyper parameter set 1; 2 complete runs and 2 failed part-way. One successful run was completed using PPO hyper parameter set 5. The failed runs for hyper parameter configuration 5 are not included to prevent crowding. Failures from configuration 1 are included for comparison as it is important to visualize when runs fail, and the amount of time consumed waiting on an unsuccessful run.

Each run is roughly 180min if fully completed. Some observations and adjustments were made throughout the training time. First, observation made was the difference in motion between different runs. There was a clear difference in the smoothness of motion as reward increased, the motion of the arm was immensely less jerky. This motion observation led to a few other observations; the more jolting and lurching in the motion the logging information resulted in higher packet hiccups between communications. Packet hiccups are delays (milliseconds) in communication between the host computer and UR5 controller, the higher the hiccup, the higher the likelihood of lost packet information or communication error. The reasoning behind this will be discussed more in the results and discussion sections. In order to improve this communication latency and smooth the robots motion, we changed the logging info that was printed on the command prompt. Printing any sort of debugging info to a serial monitor can be computationally expensive. When changing the logging configuration from DEGUG to INFO there was significant improvement in the motion of the arm or loss in communication. However, this did not solve the mid-run failure as it still occurred.

B. 6DOF

Running 6DOF remains very similar to running the 2DOF system. The biggest difference, and the one that hindered the results in this work came from the hyper parameters. Documentation on the 2DOF hyper parameters, as mentioned in the top 5 configurations, was not performed for 6DOF. This would result in having to figure out the ideal values to maximize reward through trial and error. Given that each trial requires use of the hardware, this would not be a feasible process. This remains one of the downsides found in a physical environment, as the UR5 arm has to be watched closely during its lengthy runtime, and must remain in the lab environment. Some results were found with the 6DOF mentioned in the following section.

IV. Results

Overall, data was collected for multiple runs of TRPO and PPO. Not every run was successful and many failed prior to the 150,000 timestep limit. A failed run occurred when the robot had a miscommunication with the computer causing a 'box out-of-bounds' message. This also causes the robot to stop as it is unable to receive any code from the computer, terminating the run. Fig. 3 displays the results for TRPO (top) and PPO (bottom) for all runs. With TRPO it can be seen that all runs follow around the same trend and begin to level out at around 250 reward in the end. The difference in runs are due to the generated randomness in each trial as they won't always be exactly the same. Noise could also be a factor in slight variations, however the same seed was used which is why the runs look about the same. With PPO, it can be seen that a large variation in the two main runs with run 2 (blue) failing around the 800,000 timestep mark. Its trend was following the same trend as TRPO, meaning it was the more successful run. Run 3 had a smaller final reward and a slower more linear trend which deviated from the expected result. Overall, hyper parameter configuration 1 gains a higher overall reward than configuration 5, which verifies the importance of hyper parameter selection.

In order to validate these results it is best to compare them to the findings in Lynnerup et al [2], and Mahmood et al [1]. Figure 2 from Mahmood et al, shown in Fig. 4, for the Ur-Reacher-2 tasks shows 2 seeds being run 4 times, then averaged for TRPO. Both runs have variations however they

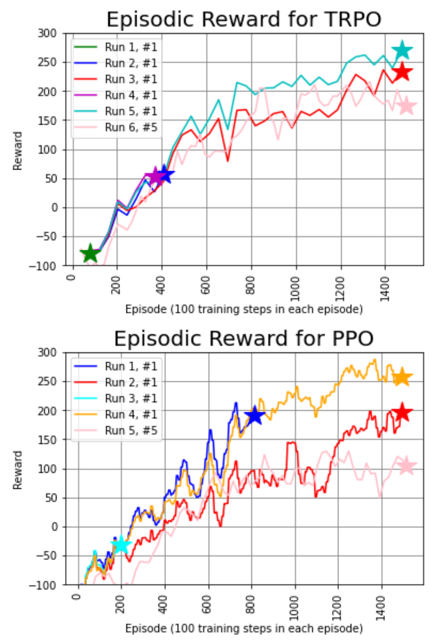

Fig. 3. Episodic rewards for TRPO and PPO for 150,000 timestep runs using hyperparameter configuration 1 and 5 from Appendix A3, Tables 3 and 4, in the paper by Lynnerup et al. [2].

follow about the same trend meaning the chosen seed should result in around the same trend every time. Figure 2 from the lynnerup et al paper, shown in Fig. 4, shows 10 runs for the same seed and hyperparameters for TRPO. The first 7 runs are almost identical, and match up to run 5 and run 6 from the TRPO results in Fig. 3. This indicates successful replication of TRPO on a UR5 robot reaching task. The last 3 runs in Figure 2 from lynnerup diverge and all follow a similar slower and more linear trend. This matches the PPO run where run 3 diverged and followed a similar trend. This error is caused from communication delays in the UR5 and the computer as mentioned in Lynnerup et al. The same error occurred in the PPO experiments with run 3 as after it completed the 150,000 timesteps it immediately needed to restart and recalibrate.

To fully validate the results, they can be compared with Figure 6 from Mahmood et al, shown in Fig. 5, where they display the average results from their 4 RL algorithms. Fig. 5 contains the best TRPO and PPO runs (blue and orange respectively). Comparing the TRPO and PPO with Mahmood et al, the trend is almost the same, with the variation due to slight changes in hyperparameters and setup.

After verifying 2DOF, the 6DOF configuration results showed no improvement. This could mainly be due to the lack of well defined hyper parameters, but run displayed in Fig. 6 uses hyperparameters similar to the configuration 1 for PPO. Overall the trend remains almost constant and sees no improvement using PPO. Future work would be required to evaluate hyper parameters and successfully train the system on 6DOF.

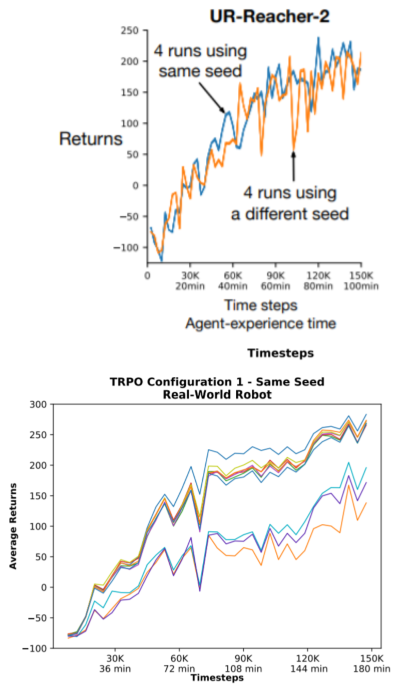

Fig. 4. Part of Figure 2 from Mahmood et al (top) [1] and part of Figure 2 from Lynnerup et al (bottom) [2]

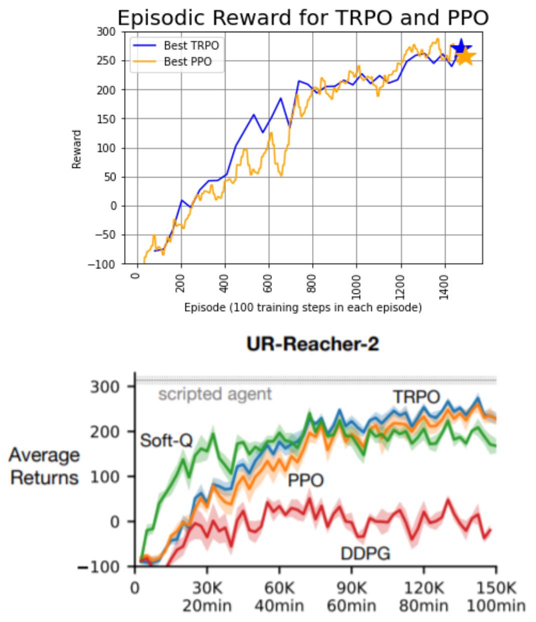

Fig. 5. Comparison of the most successful TRPO and PPO runs with configuration 1 (top) and part of Figure 6 from Mahmood et al (bottom) [1]

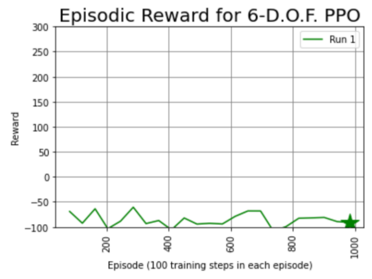

Fig. 6. 6DOF results using PPO

V. Discussions and Conclusions

Through updated and detailed documentation of setting up a Reinforcement Learning problem on a UR5 robot, it can be replicated to produce promising results. This in turn helps open the door for future research regarding physical systems and reinforcement learning. Accessibility allows for more research teams to use past work, making it easier to replicate and build on. It also helps future researchers interested in the field get a head start that focuses on using reinforcement learning rather than spending more time on the extensive software and hardware dependencies required to run it. Future research on this project could start with making DDPG and SoftQ algorithms readily available to the public for use so all four can be compared and tested. Perhaps a larger area of interest would be the actual application of the fully trained robot. RL allows for the robot to compete with complex kinematics and control loop control, so future comparisons of the two and their outcome would help compare the methods. It would also be worth documenting how long development of each would take. Such as getting the robot to train with RL versus creating the control system.

Difficulty in this research starts with the extensive software dependencies. A large majority of time was spent scouring the internet and trying to put pieces together to get the system to run. A more detailed documentation to guide through the process would make the entire project more accessible to others. Description of the software and hardware setup was done in hope to help future groups avoid unnecessary time spent on software setup.

Running the training in a physical environment was also a challenge. In a physical environment there are three main issues that stick out. The first being no visualization of the training. A custom but not as accurate border was put behind the robot to see the boundary restrictions, however it was too difficult to place a target onto the board as it changed every 4 seconds. 6DOF would be almost impossible to accurately visualize unless an extensive physical box is made to contain the end effector. This is where an accompanying virtual environment would be useful to see where in space the target is with respect to the end effector. The second main issue involves communication. Many runs failed due to a box out of bounds error meaning some part of the communication was disrupted, causing the entire training session to fail. Hiccups and delays in signals sent also cause issues in the results. As seen in the results section, communication issues can also lead to incorrect results if the test isn't stopped and the robot recalibrated. The last main issue is the time to train. With the entire system being a physical UR5 robot, it requires close monitoring during every step of the learning process as if a major hiccup is encountered, it could harm the hardware. Being a large robot arm, the system was also tethered to a lab and could only be run during lab hours, where virtual environments can be run over night. This restricted us to only doing a few runs, with only a few successful runs due to communication errors. Time restrictions and wear on the hardware also prevented us from further exploring the 6DOF problem. Without a head start on hyper parameters, it would take a long time to output good results.

Even with the few issues, this work displays how it is possible to replicate the real world RL problem on a UR5 robot. With clear documentation, and advanced knowledge of potential communication errors, future research groups, even those with little to no experience in RL and software, can replicate the results.

VI. Contributions

George Muhn - Set up the software requirements and virtual machine to run the different RL algorithms on the UR5 robot arm. Did literature review and research on TRPO and PPO algorithms. Helped Jacob on the preparation and set up of the UR5 arm to communicate with the host computer.

Jacob Sindorf - Worked on writing final report and powerpoint. Assisted in software requirements and set up with VM. Prepared the UR5 robot for experiments and communication. Organized and formatted the report and ppt. Aided in UR5 running of experiments.

Kylel Scott - Aided in the installment and debugging of software suites to facilitate robot communications from host laptop (ROS, SenseAct, etc.). Did literature research on TRPO and PPO. Wrote python scripts to plot CSV data from TRPO and PPO runs for post-experiment analysis. Contributed to various sections of the report.

References

[1] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," *CoRR*, vol. abs/1809.07731, 2018. [Online]. Available: http://arxiv.org/abs/1809.07731

[2] N. A. Lynnerup, L. Nolling, R. Hasle, and J. Hallam, "A survey on reproducibility by evaluating deep reinforcement learning algorithms on real-world robots," *CoRR*, vol. abs/1909.03772, 2019. [Online]. Available: http://arxiv.org/abs/1909.03772

[3] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," *GitHub repository*, 2017.

[4] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347