Learning-based Body / Arm Velocity Control for a Mobile Manipulation Robot

Julia Adamczyk and Fangshuo Li

***Abstract*— Recent research in mobile manipulation shows that deep reinforcement learning methods can be very successful in training autonomous robots to perform various tasks. However, choosing appropriate approach and training parameters that will train the autonomous robot successfully is a challenging venture. Factors such as the robot's internal structure, its motion control system, or just the sheer complexity of the assigned task need to be taken into account while designing a Reinforcement Learning pipeline. This paper examines the Proximal Policy Optimization algorithm and its effectiveness in training complex robots. Then, this algorithm is applied to a custom robot which needs to learn whole body control to solve the assigned task. Simulation results show that the pipeline still needs to be adjusted in order to run the training system successfully.**

I. INTRODUCTION

Autonomous robots remain as important as ever in practical applications. In achieving robotic autonomy, two choices arise: using a deterministic algorithmic approach or using a learning-based stochastic approach. While deterministic approaches are often highly efficient, the need to anticipate and manually program all possible tasks becomes extremely costly as the environment becomes more complex. Thus, many contemporary mobile manipulators rely instead on deep learning algorithms. In our experiments, we use the Proximal Policy Optimization (PPO) algorithm with an actor-critic framework to train our mobile manipulator. The mobile manipulator consists of a six-degree-of-freedom UR5 arm with a Robotiq 2-fingered end-effector mounted on a wheeled base. The training stage is conducted in Gazebo in order to avoid physical harm to the mobile manipulator. The task of the mobile manipulator is to bring its end-effector to within a specific distance of a randomly spawned goal object. Specifically, the mobile manipulator must be able to reach the goal within a specific time frame without reaching singularity or exceeding its joint limits. Our algorithm takes in the current environment as input and outputs velocity commands for the mobile manipulator.

II. PROBLEM DEFINITION

Within the last couple of decades, the field of automation has been experiencing tremendous progress. Yet autonomous robot manipulation remains one of the most demanding subjects of robotic research. While traditional control techniques can perform well in fixed environments, learning-based techniques show an advantage in dynamic environments where extreme environment conditions may arise. Deep Reinforcement Learning methods give promising results in solving challenging problems in autonomous control.

A. Reinforcement Learning Overview

Deep Reinforcement Learning has shown great results in learning complex tasks in unstructured environments. We define our mobile manipulation task as a reinforcement learning problem that can be solved using a trial-and-error approach in which the agent learns by interacting with the environment and trying to maximize the reward that it receives. The robot's interaction with the environment can be described as a Markov Decision Process (MDP). Markov Decision Process is defined as a sequence of events in which the probability of each event depends only on the previous outcome and is represented by a tuple $(\mathbf{S}, \mathbf{A_S}, \mathbf{P_a}, \mathbf{R_a})$ where $\mathbf{S}$ represents state space, $\mathbf{A_S}$ is the set of actions available from state $\mathbf{S}$, $\mathbf{P_a}(\mathbf{s}, \mathbf{s}')$ is the probability of taking action $\mathbf{a}$ in state $\mathbf{s}$ that will advance to state $\mathbf{s}'$, and $\mathbf{R_a}$ is the reward received after transitioning to state $\mathbf{s}'$ from state $\mathbf{s}$ by choosing action $\mathbf{a}$. The Reinforcement Learning process is shown in Figure 1. In real-world environments

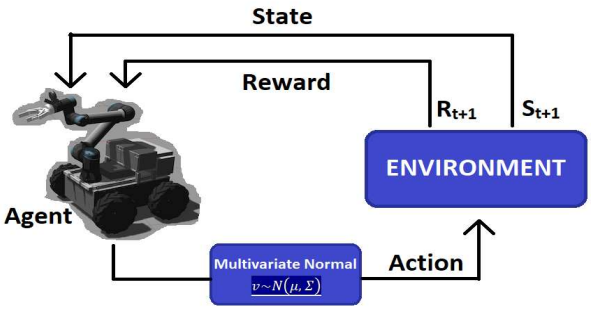

Fig. 1. Basic Reinforcement Learning Framework: the agent advances from state s into state s' by taking an action sampled from Multivariate Normal Distribution. This process is repeated until the agent learns the optimum policy parameters.

the probability distribution of actions can be approximated by Gaussian Distribution. Sampling from Multivariate Normal Distribution with mean μ and covariance $\mathbf{\Sigma}$ allows for better exploration of the environment without frequent policy updates during training.

This material is based upon work supported by the Nevada Governors Office of Economic Development under the Construction Robotics Award. The presented content and ideas are solely those of the authors.

The authors are with the Robotic Workers (RoboWork) Lab, University of Nevada, Reno, 1664 N. Virginia, 89557, Reno, NV, USA cpapachristos@unr.edu

B. Task Setup

In our project, the mobile manipulator must learn to bring its end-effector to close proximity of the timber board, the target object, in one smooth motion. At the beginning of each episode, the target object is initially spawned out of the robot's reach. The task is considered completed when the the gripper is within appropriate distance to grasp the timber board. Specifically, the task is completed when the distance and orientation from the timber board is less than 0.076 meters and 10 degrees (0.174 rad) respectively. Grasping the target object and avoiding obstacles are not part of the task. The three task completion conditions are:

1) The robot's gripper reaches the desired location,
2) During episode, the robot's arm does not reach its joint limits,
3) During episode, the robot's arm does not reach singularity.

C. Singularity

The robot must learn to avoid reaching singularity. Singularity is formally defined using the equation:

$$\dot{q} = J^{-1}v \tag{1}$$

where v is defined as

$$v = [\dot{x}, \dot{y}, \dot{z}, \omega_x, \omega_y, \omega_z]^T$$

and the Jacobian matrix is a $6 \times n$ matrix of the joint arm positions and robot geometry where n is the number of degrees of freedom. The configuration reaches singularity when the Jacobian matrix loses rank and equation (1) becomes undefined. Intuitively, singularity can be thought of as when a certain configuration causes the robot to lose a degree of freedom and thus becomes blocked from moving in certain Cartesian directions. Singularity must be avoided due to the impossibility of crossing points of singularity and the high resulting joint velocities when approaching singularity. When the Jacobian matrix nearly loses rank and its determinant approaches zero, the joint velocities must become very large to move the end-effector by a certain amount.

III. PROPOSED APPROACH

A. Simulation

Due to the task complexity, training is first performed in a simulated environment using Gazebo. The autonomous-mobile-manipulation repository provides a motion planning pipeline for navigating the mobile robot [1]. The robot's components include a wheeled base, UR5 Arm Manipulator and Robotiq 2-finger gripper. The robot's base and arm are actuated using joint velocity control. The simulation runs at the rate of 5Hz with 256 time steps per episode. At each time step, the action, in the form of base and arm joint velocities, is chosen by the RL agent based on the current state. Then, the action is executed by the motion planning pipeline in order to advance the robot to the new state. After every episode, the simulation condition is "reset". The goal, a timber board and a cinder block as a support, is randomly spawned 5 meters away from the robot's current location with a random yaw orientation. The arm is returned back to its original "home position" via forward kinematics, but the base remains in the same orientation as prior to the reset. Resetting the world this way is supposed to increase the exploration of the environment. It can possibly teach the robot to reach the actual goal rather than reach the certain position in the environment. The Gazebo world used in this experiment does not contain any obstacles which simplifies the task definition significantly. The only objects launched in Gazebo world are the robot itself and the goal object. By doing so, we are aiming to show proof of concept rather than develop a real world application for the robot.

B. Observation and Action Space

The *bvr SIM* mobile manipulator used in this experiment is shown on Figure 2. It is an 8 degrees of freedom custom

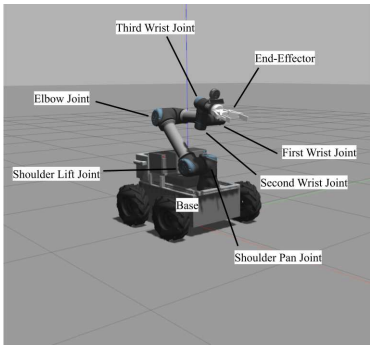

Fig. 2. $BvrSIM$ robot is a 8DoF robot that is controlled using joint velocity commands that correspond to each joint.

robot designed in the University of Nevada, Reno $Robowork$ lab. The action space includes 2 subcomponents: base action space and arm action space. The base action space consists of linear velocity (x) and angular velocity (yaw). The arm action space consists of 6 joint velocities that move corresponding joints of the robot: elbow joint, shoulder lift joint, shoulder pan joint, first wrist joint, second wrist joint, and third wrist joint. The observation space is 32 items large:

- Base state in the world frame (x_b, y_b, ϕ_b)
- End effector state in the world frame $(x_e, y_e, z_e, \psi_e, \theta_e, \phi_e)$
- Arm joint angles $(x, y, z, \psi, \theta, \phi)$
- Arm joint velocities $(\dot{x}, \dot{y}, \dot{z}, \dot{\psi}, \dot{\theta}, \dot{\phi})$
- Goal position and orientation relative to the base $(\delta_{x_b}, \delta_{y_b}, \delta_{\psi_b}, \delta_{\theta_b}, \delta_{\phi_b})$
- Goal position and orientation relative to the end effector $(\delta_{x_e}, \delta_{y_e}, \delta_{z_e}, \delta_{\psi_e}, \delta_{\theta_e}, \delta_{\phi_e})$

where ψ is roll, θ is pitch, and ϕ is yaw.

C. Reinforcement Learning Algorithm

1) Proximal Policy Optimization: The *bvr SIM* robot is trained using the Proximal Policy Optimization, a policy gradient Reinforcement Learning method. Even though this algorithm is quite new, it is already broadly used in training mobile manipulators to solve various tasks due to its simplicity of implementation and great performance. [2–4] In addition, PPO performs better than other RL algorithms in

solving continuous control tasks and is comparably easier to tune. [5] PPO can be implemented with the actor-critic framework. In this work, the policy network is estimated using the Clipped Surrogate Objective function to ensure that policy gradient updates are not changing the policy excessively. It smoothens the training process and discourages extreme, possibly unfavorable, updates. The Clipped Surrogate Objective is defined as:

$$J^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \quad (2)$$

where r_t is the ratio of the probability under the new and old policies, respectively, A_t is the advantage estimated at time t, and ϵ is the clip value. Advantage is calculated using Generalized Advantage Estimation (GAE) which, by tuning hyper-parameters δ and γ, allows adjustment of the bias-variance trade-off[6]. The Generalized Advantage Estimator function is defined as:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V \quad (3)$$

Proximal Policy Optimization with Clipped Objective provides a clever approach to finding an optimal solution for policy gradient. When the advantage at the particular time step is positive (the action was better than expected) then the probability of taking that particular action should increase. Therefore the gradient moves along the positive direction, however, is clipped by certain range it can move forward in order to prevent too large of an update. On the other hand, if the advantage is negative (the action was worse than expected), the probability of taking that action should decrease. The gradient moves in the negative direction allowing the policy to undo the latest gradient step. The value function estimation of the critic network is estimated using the mean squared error between the observed value and the value predicted by the critic network [6].

2) Actor-Critic Architecture: As Proximal Policy Optimization follows the actor-critic framework, we use two Neural Networks, one for the actor, and one for the critic to train our model. The architecture of both networks is shown in Figure 3. Both actor and critic networks receive the same observation as input. They are also structured similarly. Both networks have two hidden layers with 256 neurons each applied with *ReLU* activation function. A hyperbolic tangent $(tanh)$ activation function is used in the output layer of the actor network while the critic network uses a *linear* activation function on its output layer. The *tanh* activation function is applied to actor network output in order to restrict the chosen velocity mean value to within the rage of acceptable input for motion planning pipeline ([-1, 1] m/s or rad/s). The control velocity action is then sampled from a Multivariate Normal Distribution to provide balance between exploration and exploitation.

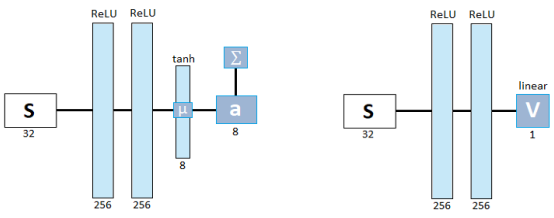

Fig. 3. Actor-Critic framework uses two separate fully connected Neural Networks in order to approximate the parameters of the policy and value function.

D. Reward Function

Defining reward functions for Reinforcement Learning problems is a challenging task. In high-dimensional continuous control tasks, a sparse reward function might not be sufficient for learning. Therefore, in this project, the reward is defined by continuous reward function:

$$r_t = -w_d * d_t + e^{-d_t^2} \quad (4)$$

where w_d is the weighting factor (0.5), and d_t is the distance from a goal at time t. Using a continuous reward function in the task of reaching the goal facilitates and speeds up the process of learning by giving the agent gradual feedback. This way, the agent knows when it gets closer to the goal and progressively becomes better at solving the task. In addition to the continuous reward received at each time step, the agent is also rewarded positively by adding an extra 100 points when the gripper reaches the desired distance and orientation from the goal, and penalized by subtracting 100 points when the goal was not reached or if the arm reached singularity or joint limits during the episode.

IV. IMPLEMENTATION

Implementation of the project can be found in the GitHub repository [7]. The project involves combining the Reinforcement Learning training pipeline with the robot motion planning pipeline. The robot control pipeline is defined in the *autonomous-mobile-manipulation* repository in GitHub [1]. While the training pipeline determines what actions are chosen, the motion planning pipeline is responsible for executing them. The Reinforcement Learning framework is placed in the ros-gym-mobile-manipulation package. This package follows the Open AI framework for training ROS-based robots in Gazebo and uses some of the provided libraries. According to this framework, there are three sub-environments that help in implementing and organizing the code. The Gazebo Environment creates a connection between the robot and the simulation. The Robot Environment defines the basic robot control functionalities, i.e. how the robot moves or interacts with the environment. Lastly, the Task Environment defines functions related to completing the specific task at hand, in this case, grasping the goal. This framework is very useful in making the ROS project user-friendly and well-organized. One crucial functionality, the arm reset, is implemented using Action Client Server. Moving the arm

TABLE I

PROXIMAL POLICY OPTIMIZATION HYPERPARAMETERS.

Hyper-parameter	Value
PPO Clip Value	0.2
PPO epochs	16
Minibatch	32
Optimizer	Adam
Learning Rate	0.0003
Discount Factor	0.99
GAE Coefficient	0.95
Sample Batch Size	512

home is performed by executing a joint trajectory goal action.

The agent is trained on data generated by Gazebo simulation. The observation, which is fed to the Neural Network, is taken directly from either subscribing to the certain topics or calculated using frame transformations. The training is done using an on-policy algorithm in which the policy is evaluated using the experience collected from the current policy. Therefore, the agent uses real-time generated samples to update the policy. In this project, we use batch sampling to train the robot. The hyperparameters used on the PPO algorithm for training are shown in Table 1. The RL agent takes 512 $(batchsize)$ steps before updating the policy. The samples collected from the environment are first stored in the buffer, and then used to update the policy. The advantage and return for each time step are calculated using the Generalized Advantage Estimation. The policy is updated with 16 epochs using mini-batches of size 32.

V. EXPERIMENTS AND DISCUSSION

This project's entire pipeline consists of two main components: the Reinforcement Learning algorithm and the custom OpenAI Gym Environment for the *bvr SIM* robot. Therefore, we split this complex project into smaller parts in order to assess each component separately.

A. PPO Evaluation

Evaluation of the PPO algorithm can be performed using one of the environments already provided by Gym. Doing so, we check if the implementation of PPO is correct. Gym is a toolkit that allows for comparing the performance of different Reinforcement Learning algorithms. The Gym Environment that is chosen for evaluation is Continuous Lunar Lander. Its description can be found on the OpenAI Gym official website. The PPO agent is initialized with different parameters in order to train the model for solving the landing task. Adjusting parameters is essential as part of the parameter tuning and motivated by observation and action spaces being comparably smaller than the ones defined in the main experiment. The RL agent was able to achieve sufficient results during training and the agent was able to land in the target location as shown in Figure 4. Its performance is evaluated based on the test run where the agent uses trained policy to evaluate an action. Rewards received by the agent in the test run exceed 200 points which, in this particular environment, is the threshold indicating that the

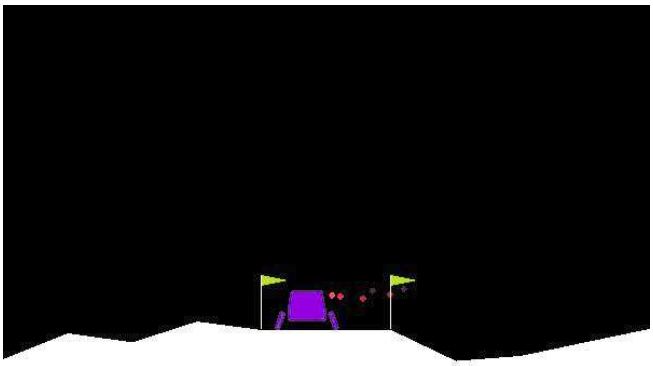

Fig. 4. The task is completed when the Lunar Lander lands in the landing zone, preferably with zero speed.

task is solved. One of the test runs is recorded as a .gif file and can be found here: Lunar-Lander-Run. The performance and effectiveness of the RL algorithm can be evaluated based on the rewards that the agent is receiving in the training process. As the agent learns how to solve the particular task, the reward should gradually increase. Figure 5 illustrates the reward collected by the RL agent during training in Lunar Lander Continuous Environment. It can be seen that during

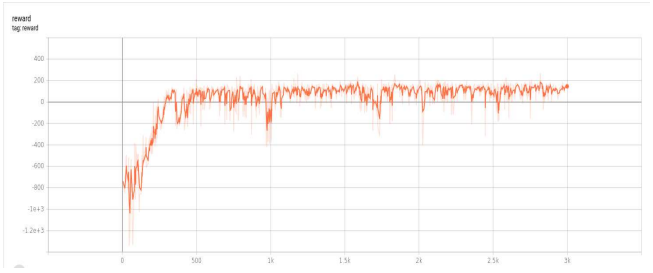

Fig. 5. Reward collected by the RL Agent in training Lunar Lander Continuous is gradually rising and reaches desired threshold after around 1000 episodes.

first 500 episodes the reward increases slowly. This part of training is when the initial exploration of the environment is performed by the agent. Once it starts obtaining higher rewards, it learns the task and is able to stay on the desired reward threshold.

B. $BvrSIM$ Environment

The $bvrSIM$ environment is also tested separately from the RL algorithm to see if all necessary functionalities work properly. Such functionalities include: publishing actions, watching the singularity topic, watching the joint limit topic, and resetting properly. In order to test the environment implementation, we set up a pipeline where random velocity commands would be published and the robot would need to detect singularity and reset itself. The issue with the motion pipeline we have noticed was that compliant commands are constantly paused. There are three reasons for this occurrence: (1) the velocity commands are too large to be executed, (2) the robot's arm reached the joint limit,

or (3) the robot's arm is near singularity. While the joint limit and singularity issues are taken care of by subscribing to message topics explicitly, the combinations of velocity commands need to be learned by the RL agent itself and cannot be accounted for in the motion planning pipeline. This is because even if each individual joint velocity command is between -1 and 1, the resulting transformed Cartesian velocity might exceed the joint velocity limit. The resetting functionality involved two processes: resetting the goal and resetting the arm. The goal is able to be properly moved to a random orientation 5 meters from the robot base after each episode. The arm reset is tested against multiple invalid configurations, and the arm seems to be able to recover from most positions. However, there seems to be no pattern detected among the configurations that the robot can not recover from.

C. Main Experiment

Ideally, to provide sufficient data in the $bvrSIM$ environment, the training pipeline should run for at least $1,000,000$ time steps. In high-dimensional continuous spaces, training the RL agent is time-consuming because the agent needs to collect vast amounts of data before it is able to learn. In our case, the pipeline is only able to run for 1135 time steps before the arm falls into an unrecoverable configuration. Using velocity control as opposed to position control allows

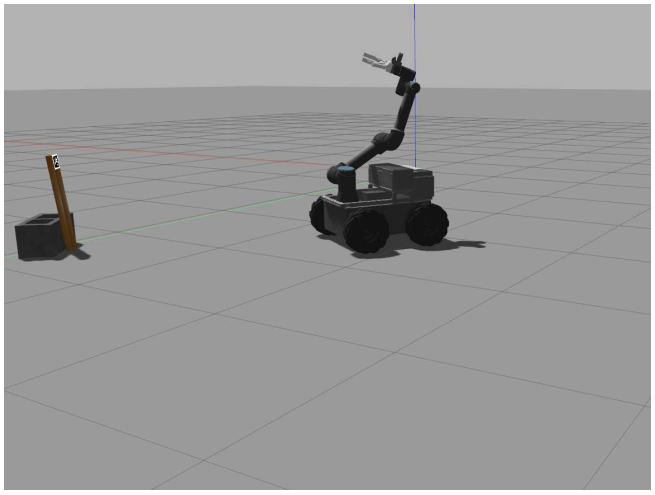

Fig. 6. The arm configuration which allows the robot to recover even though it calls 'reset' function multiple times before the arm is brought to the proper position.

for a faster training process as the actions are executed more frequently, however, velocity control is also more prone to incorrigible behavior. In our case, this behavior includes reaching singularity or exceeding joint limits. Over multiple episodes, this behavior eventually leads to failure in resetting the arm. There are multiple arm configurations in which the trajectory goal is either pre-empted or aborted due to the path tolerance violation. Interestingly, the arm is able to recover from the position shown in Figure 6 by repeatedly calling and failing the arm reset function. However, the configuration shown in Figure 7 never seems to be handled correctly. Another issue is that the robot tends to reach

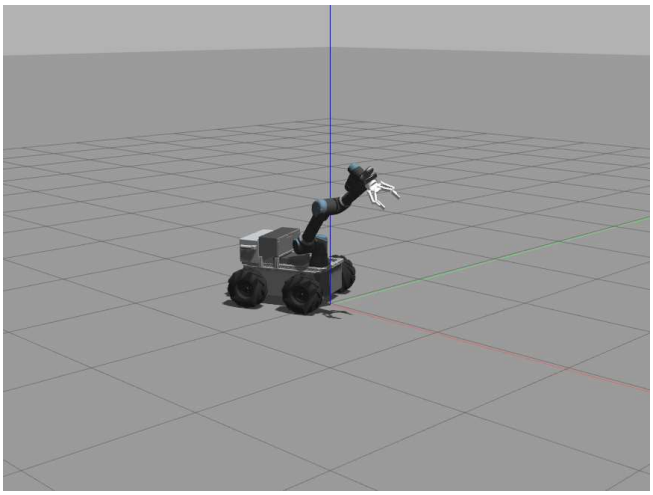

Fig. 7. The arm configuration from which the robot cannot recover. Due to the gravity acting on the fully extended arm, the robot motion planning pipeline is unable to execute the home trajectory.

singularity early on resulting in episodes being much shorter than expected. Because we use the clipped PPO objective function and initialize the policy parameters at random, the process of finding proper policy parameters might take a longer time. Thus, during initial part of training the robot will get stuck more often as the actions are sampled from a poor policy. The reward function we obtained from one of the trial runs is shown in Figure 8. The shape of the reward function indicates that the agent is in the early stage of training process. The reward is noisy due to the extensive initial exploration. The training lasted for 52 episodes and

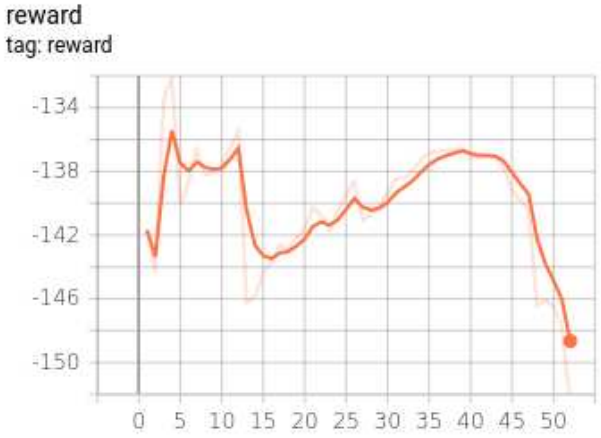

Fig. 8. The reward function obtained from one of the runs.

the policy and value function parameters were only updated twice. The actor loss and critic loss are shown in Figure 9. The training run is too short to make any conclusions about the PPO performance in this environment. In order to obtain sufficient results the pipeline needs to be debugged so the training process can complete at least $1,000,000$ time steps. Nevertheless, we are able to identify the areas of possible improvement. In terms of the motion planning pipeline,

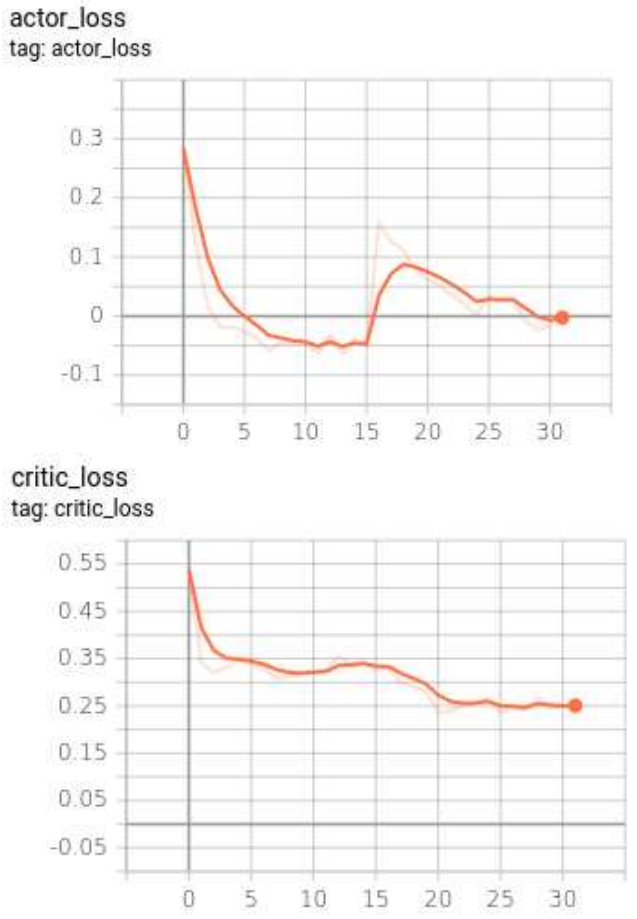

Fig. 9. The results obtained are not sufficient to make any conclusions about PPO performance.

potential areas of improvement would be fixing the arm reset functionality and reducing the reliance on the Gazebo simulated environment by reading from the sensor plugins. In terms of the training pipeline, potential areas of improvement would include further parameter tuning, penalizing actions that exceed acceptable velocities, and testing various RL algorithms against PPO.

VI. CONCLUSION

The training process is not completed due to the arm reset failure which breaks the system flow. The performance of Proximal Policy Optimization cannot be estimated due to insufficient training process. Proximal Policy Optimization algorithm is successfully tested on the Gym environment, however, for the algorithm to succeed on the $bvrSIM$ environment the pipeline must be debugged.

REFERENCES

[1] Robowork, "Autonomous robot manipulation," August 2021. [Online]. Available: https://github.com/robowork/autonomous_mobile_manipulation

[2] J. Kindle, F. Furrer, T. Novkovic, J. Chung, R. Siegwart, and J. Nieto, "Whole-body control of a mobile manipulator using end-to-end reinforcement learning," 02 2020.

[3] C. Wang, Q. Zhang, Q. Tian, S. Li, X. Wang, D. Lane, Y. Petillot, and S. Wang, "Learning mobile manipulation through deep reinforcement learning," *Sensors*, vol. 20, p. 939, 02 2020.

[4] K. Wang and Y. Li, "Comparison of control methods: Learning robotics manipulation with contact dynamics," in *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2018, pp. 1–7.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[6] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," 06 2015.

[7] J. Adamczyk and F. Li, "Ros gym mobile manipulation," August 2021. [Online]. Available: https://github.com/robowork/ros-gym-mobile-manipulation/tree/summer_project