RL-based generalized safe manipulator controller for human-robot interaction

Arjun Rajeev Warrier
Department of Electrical and Computer Engineering
warrier.arj@northeeastern.edu

Divyansh Gupta
Department of Mechanical and Industrial Engineering
gupta.divy@northeastern.edu

I. Introduction

In the evolving field of robotics, the integration of Reinforcement Learning (RL) for enhancing human-robot interaction, especially in industrial environments, is becoming increasingly vital. Our project focuses on the development of an RL-based controller to navigate and place objects within environments containing both multiple obstacles and disturbances in target/goal object positions. Manipulator control models have the characteristics of high-order, nonlinear and multivariable which makes it hard to have good adaptability. Aiming at these problems, we utilize multiple Actor-Critic algorithms to implement continuous control with a custom environment model and well defined reward schemes. This project primarily addresses the challenge of safety and generalizability in manipulator controllers, ensuring their adaptability to new environments that may be populated with humans and other impact sensitive machinery.

II. Methodology

In our project, the primary focus is on simulating a manipulator within a human-surrounded environment. The objective is to utilize this simulated environment to train a reinforcement learning (RL)-based controller capable of ensuring human safety when interacting with manipulators. Another key goal of the project is to achieve task generalizability, allowing adaptability to disturbances in object and goal positions. We leverage the MuJoCo physics engine and extend the previously available Pusher-v4 environment from OpenAI to create our custom simulation environment.

A. Environment

*1) **Model**:* The Pusher-v4 environment employs a Pusher XML file to generate the physics model of the manipulator under consideration. This is a two link manipulator with 7 degrees of freedom. For our work, we have modified this model to include obstacles that mimic humans within the robot workspace. Additionally, we have adjusted the joint ranges, and contact dynamics to simulate desired safety margins.

Obstacles: Spherical objects, possessing mass and elastic coefficients similar to the human arm, are employed as obstacles to replicate human body characteristics. The added obstacles have 3 DOF - can move along X, Y and Z axis, with added damping.

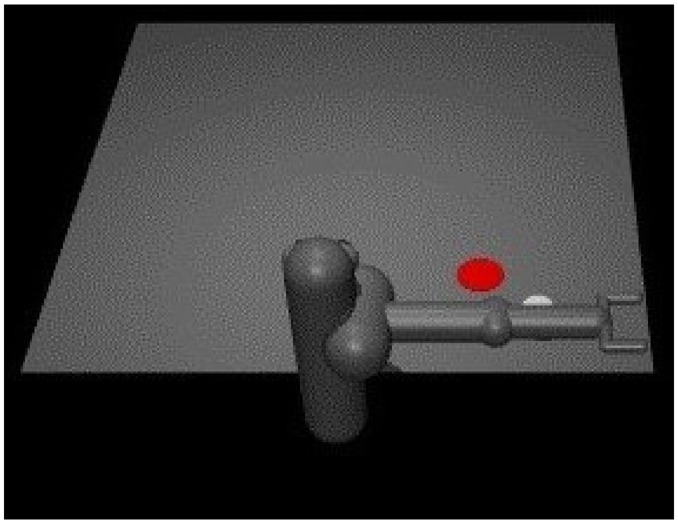


Fig. 1: Default Pusher Environment

Contact Dynamics: Contacts within the geometries of the model are modified to simplify the model for computational feasibility. Additional contact pairs are generated for obstacles and the manipulator body.

*2) **State Space**:* The state space serves as an abstract representation of the environment, which is passed to the RL algorithms for decision-making. In this case, we append the default state space *reference* of the Pusher-v4 environment to incorporate additional states corresponding to obstacles in the surroundings. Appended states includes:

$$\begin{aligned}\text{State Space} = [&\text{Default state space,}\\ &\text{Obstacle Position,}\\ &\text{Safety Margin of each obstacle,}\\ &\text{Contact Boolean of each obstacle}]\end{aligned}$$

*3) **Action Space**:* The action space represents set of possible actions agent can act. In our environment action space encompasses 7 actuators - 3 at Shoulder Joint, 2 at Elbow Joint and 2 at Wrist Joint.

*4) **Reward Function**:* In reinforcement learning reward functions are defined to evaluate agents interaction with the environment. This reward function guides the RL-algorithm to learn favorable vs unfavorable actions for corresponding state and through training it learns to choose optimal actions within

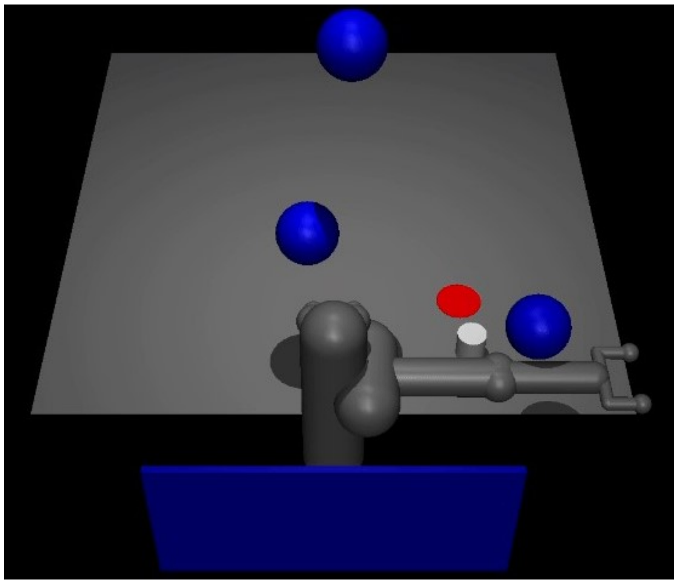


Fig. 2: Modified Pusher Environment

each state. To model safety and generalizability, a new reward function is used, incorporating penalties to enforce scenarios with no impact, which is crucial for human-safe environments.

Default Penalty: The agent receives a penalty proportional to its distance from the target object and the target object's distance from the goal position. An additional penalty is applied for effort.

$$\text{Reward} = \text{Object Dist} + 0.5 \times \text{Goal Dist} + 0.1 \times \text{Effort}$$

Safety Margin Breach Penalty: If the agent (Manipulator) moves within the safety margin of the obstacle (where the safety margin is the minimum distance between the manipulator and the obstacle), it receives an additional penalty based on the margin:

$$\text{Reward} = \begin{cases} 1.0 \times \text{Default Penalty} & \text{if margin} \geq 15 \text{ cm} \\ 1.1 \times \text{Default Penalty} & \text{if } 10 \text{ cm} < \text{margin} < 15 \text{ cm} \\ 1.3 \times \text{Default Penalty} & \text{if margin} \leq 10 \text{ cm} \end{cases}$$

Impact Penalty: If the agent touches the obstacle, a penalty proportional to the impact magnitude received by the agent is applied:

$$\text{Reward} = 1.5 \times \|\mathbf{Acc}\|_2$$

B. Algorithms

For the purpose of dealing with continuous states, three popular Actor-Critic algorithms were selected, namely, Proximal Policy Optimization, Deep Deterministic Policy Gradient and Soft Actor-Critic. With various implementations available, the Stable Baselines versions were tuned and utilized in this project. Parameters were tuned with a focus on increasing agent exploration during the training phase.

1) Proximal Policy Optimization (PPO): PPO, a policy gradient method, focuses on balancing exploration and exploitation by using a clipped objective function. This function restricts the updates to the policy, ensuring small, incremental changes and improving training stability. The parameters modified here were the clip range values to [0.2, 0.5] and the usage of generalized State Dependent Exploration (SDE) instead of the regular action noise inclusion.

2) Deep Deterministic Policy Gradient (DDPG): DDPG combines policy-based and value-based approaches. It utilizes a deterministic policy to optimize the expected return and employs a separate critic network to estimate the value of state-action pairs, enabling it to handle continuous action spaces efficiently. The parameters modified here were action noise distribution parameters and the usage of SDE over action noise induced exploration.

3) Soft Actor-Critic (SAC): SAC is an off-policy algorithm that optimizes a stochastic policy in an entropy-augmented objective framework. It encourages exploration by adding an entropy term to the reward, helping the algorithm to find robust policies in complex environments.The parameters modified here were the entropy coefficients, action noise distribution parameters and the usage of SDE over action noise induced exploration.

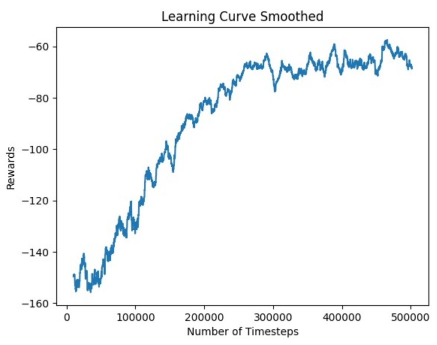
III. Results

Our experiments with the algorithms initially yielded minimum success and required further tuning of the algorithm parameters to improve exploration during training. The training parameters are mentioned in Table.I. These yielded the learning curves (Rewards vs Number of Timesteps Executed) shown in Fig.3. The following observations were obtained from training:

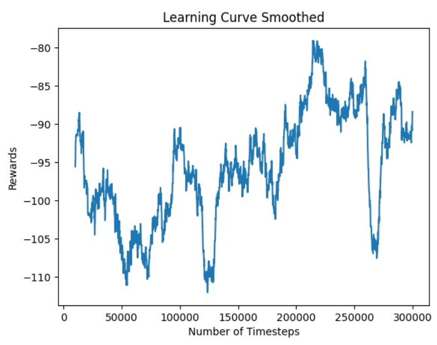
1) PPO shows more success in converging with a stable learning curve. It is faster in training with just 35 minutes required to run half a million time steps of training. It also shows more robustness to parameter changes in the algoriothm prior to training.
2) DDPG was also observed to be the worst in terms of stability of learning. It shows high sensitivity to parameter tuning. The DDPG training curve does not converge in 300,000 steps and shows constant instabilities in the plot, where rewards plummet after a large number of steps.
3) SAC has been observed to converge in the least amount of training time steps. It also outputs a fairly stable plot. DDPG required 104 minutes to train for 300,000 steps, while SAC was slowest and required 180 minutes for the same amount of steps.

The trained models were then tested for their performances.

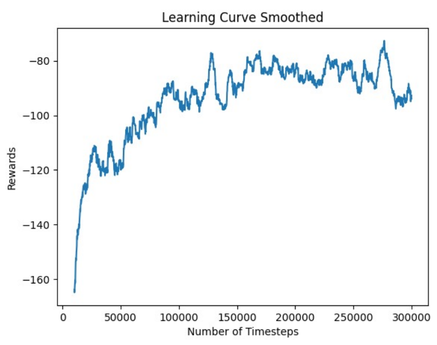
1) The models were run for 100 episodes and their success rate was noted as Fig.4. PPO showed 12 successes, while SAC showed 7 and DDPG showed 3.



(a) PPO



(b) DDPG



(c) SAC

Fig. 3: Smoothed Learning Curves with PPO, DDPG and SAC

Algorithm Used	PPO	DDPG	SAC
Training Steps	500,000	300,000	300,000
Training Time (min)	35	104	180

TABLE I: Training Parameters

2) The models were also run continuously, till they achieved 20 successful episodes, where the episode terminated by the agent successfully pushing the object to the target location. The average power expenditure per episode has been computed from the recorded states(angular displacements),time and actions(torques). As seen from Fig.5,PPO showed the least power required because of its better convergence after more training. SAC showed slightly more, while DDPG shows the worst performance in lieu of it not converging.

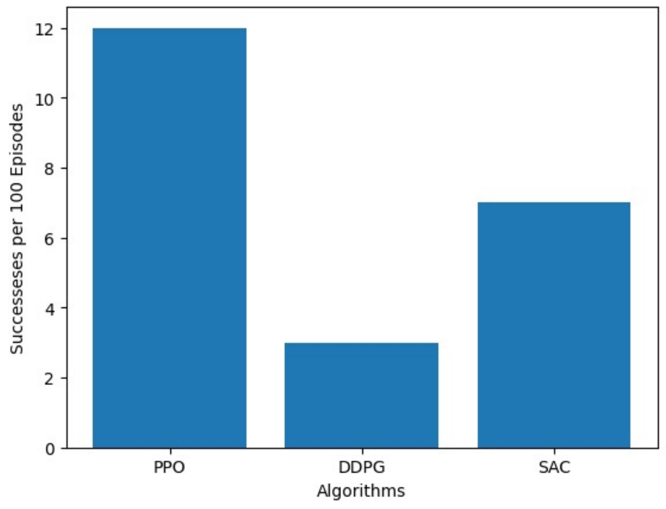


Fig. 4: Frequency of Successes

IV. Conclusion

A manipulator was trained and simulated using three reinforcement learning algorithms, Proximal Policy Optimization, Deep Deterministic Policy Gradient and Soft Actor-Critic. These trained models then showed varying levels of success in pushing an object to a fixed target, while successfully navigating among static obstacles.

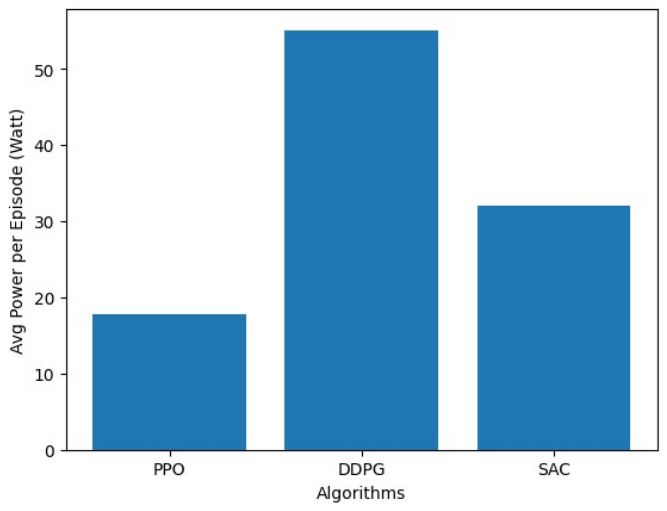


Fig. 5: Average Power per Episode

1) PPO shows success with the fastest training times and stable convergence. Excellent for quick training.
2) DDPG shows the most sensitivity to parameter tuning and could potentially show better results with specific tuning.
3) SAC also shows stable convergence, but takes the longest to train for the same number of steps. However, it is also noted that it achieves convergence in relatively lesser steps than the rest. Fastest convergence in terms of iterations.

V. Future Work

For this project, static obstacles, objects and targets were initialized with noises in their locations for variability. This could be extended into training with dynamic environment constituents, which would be further realistic when training for work spaces with human presence in them. In addition, longer training periods with more parameter tuning could potentially increase returns and provide for better training results.

ACKNOWLEDGMENT

I want to express my sincere gratitude to Prof. Robert Platt and teaching assistants for their valuable guidance and support throughout this project. Their mentorship has helped shape my understanding of the subject matter and provided me with the necessary tools and resources to complete this work.

REFERENCES

[1] Thumm, Jakob and Althoff, Matthias. (2022). Provably Safe Deep Reinforcement Learning for Robotic Manipulation in Human Environments.

[2] Q. Li, J. Nie, H. Wang, X. Lu and S. Song, ”Manipulator Motion Planning based on Actor-Critic Reinforcement Learning,” 2021 40th Chinese Control Conference (CCC), Shanghai, China, 2021, pp. 4248-4254, doi: 10.23919/CCC52363.2021.9550010.

[3] Y. Hu, L. Cui and S. Chai, ”Optimal Tracking Control for Robotic Manipulator using Actor-Critic Network,” 2021 40th Chinese Control Conference (CCC), Shanghai, China, 2021, pp. 1556-1561, doi: 10.23919/CCC52363.2021.9549419.

[4] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 5026-5033).

[5] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. J. Mach. Learn. Res., 22, 268:1-268:8.