Guided Pushing with 7-DOF Arm Using Reinforcement Learning

Praveen Krishnapur
e1373776@u.nus.edu

Abstract

In this project, we developed a 7-DOF robotic arm capable of learning to push an object to a desired location in a dynamically generated environment. The task involved precise control of the arm's end-effector to guide a ball into a target hole. Unlike traditional path-planning algorithms, which often struggle in dynamic environments with changing goals, our approach leveraged reinforcement learning (RL) to dynamically adapt and execute optimal behaviors.

Using the Proximal Policy Optimization (PPO) algorithm, we trained the robotic arm with a custom neural network architecture that integrates multi-headed attention and residual blocks. These enhancements improved the network's ability to focus on observation-goal relationships and mitigated vanishing gradient issues, enabling efficient learning. Training was conducted with both dense and sparse reward structures, highlighting the model's strong performance with dense rewards and its need for further refinement under sparse conditions.

Testing revealed a cumulative success rate of 88% over 100 episodes for fixed initial positions, demonstrating the model's effectiveness. However, jerky movements observed during simulation pose challenges for deployment in real-world settings, where smooth transitions are critical. Future improvements include motion-smoothing techniques and hybrid reward structures to enhance adaptability and robustness across diverse scenarios.

Contents

1 Introduction

Autonomous material handling and object transportation in dynamic environments, such as warehouses, factories, and healthcare facilities, have become critical areas of research. Traditional path-planning algorithms often struggle in such settings, where goal positions and object properties change frequently. Reinforcement learning (RL) provides a promising solution by enabling robots to learn optimal behaviors through trial and error, adapting to varying conditions without explicit trajectory planning.

Robotic arms are composed of multiple interconnected segments, with each joint providing a Degree of Freedom (DOF) that allows precise and flexible movement. While arms with six or more DOF can move their end-effector in any direction, traditional algorithms like Rapidly-exploring Random Trees (RRT) or Probabilistic Roadmaps (PRM) are limited in their ability to handle dynamic environments. These approaches often require pre-defined trajectories, making them less effective when goals and object properties change unpredictably. Other methods, such as Model Predictive Control (MPC), predict future states to generate optimal actions but are computationally expensive, particularly in high-dimensional systems like a 7-DOF robotic arm. RL overcomes these challenges by training robots to adapt dynamically and execute complex tasks efficiently. This project focuses on leveraging RL to train a 7-DOF robotic arm to push objects to desired locations, demonstrating precise control of its end-effector in a dynamically generated environment.

The problem addressed in this project involves training a 7-DOF robotic arm to perform guided motion in a constrained 3D environment. Using an open cup as its end-effector, the arm pushes a ball into a randomly positioned hole located within its reachable workspace. The task demands precise control of the robotic arm's power and angle to guide the ball accurately. The setup places both the robotic arm and the hole on a flat surface and their positions are randomized for each episode, creating diverse training scenarios that challenge the robot to adapt dynamically. Starting from a fixed initial position and joint configuration, the robot learns through reinforcement to efficiently guide the ball to the hole, achieving both precision and accuracy in its movements.

2 RL Cast

In reinforcement learning (RL), the framework consists of states, actions, observations, and rewards that interact to train the agent. For our project, these components are defined as follows:

2.1 State Space

The state space captures all the essential information needed to describe the environment at a given time. In our task, the state space includes:

- The 3D position of the robotic arm's end-effector.
- The 3D position of the ball in the world coordinate system.
- The relative position of the ball with respect to the end-effector.
- The rotation of the ball in Euler coordinates.
- Linear and angular velocities of the ball.
- Velocities of the end-effector and its components.

These features provide the agent with a comprehensive understanding of the environment to make informed decisions.

2.2 Action Space

The action space is defined as a continuous 4-dimensional vector:

- The first three dimensions represent Cartesian movements (*dx, dy, dz*) of the end-effector.
- The fourth dimension controls the opening and closing of the end-effector. However, in this project, the end-effector remains fixed for pushing actions.

Each action value is bounded between -1 and 1, ensuring precise and smooth control.

2.3 Observation Space

The observation space includes the processed state information returned by the environment, consisting of:

- Current observations of the end-effector, ball, and their interactions.
- Achieved goal: The current position of the ball.
- Desired goal: The target position of the ball.

The observations allow the agent to evaluate its progress and plan subsequent actions.

2.4 Reward Function

The reward function incentivizes the agent to push the ball toward the goal:

- A **dense reward** computes the negative Euclidean distance between the ball's current position and the target position. This encourages the agent to minimize the distance to the goal and is defined as:

$$d = \|\text{achieved goal} - \text{desired goal}\|_2 \tag{1}$$

  Here, d represents the Euclidean distance between the achieved goal and the desired goal.

- A **sparse reward**, which assigns a binary reward based on the proximity of the ball to the goal, is defined as:

$$\text{Reward} = \begin{cases} 0 & \text{if } d < \text{distance threshold} \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

  The sparse reward provides feedback only when the ball reaches within a defined threshold of the goal, which mimics real-world scenarios but leads to longer training times.

The reward structure guides the agent to learn efficient strategies for accomplishing the task. Dense rewards enable faster convergence by continuously encouraging the agent to reduce the distance to the goal, while sparse rewards focus on task completion but require more fine-tuning to achieve optimal performance.

3 Environment - OpenAI Gym with MuJoCo Physics Engine

The environment for this project is developed using the MuJoCo physics engine integrated into the OpenAI Gym framework. It leverages packages such as MuJoCo-Py, Gymnasium Robotics, and NumPy for creating a realistic and adaptable simulation. The core of the environment is built upon the `MujocoRobotEnv` class from the Gymnasium Robotics library. This class forms the backbone of robotic simulations, managing the interactions between the robot's actions, observations, and the physics simulation. It also facilitates the control of actuators and provides essential data, such as joint positions, velocities, and contact forces.

3.1 Environment Description

The simulation involves a 7-DOF robotic arm equipped with an open cup as its end-effector. The task requires the arm to push a ball into a randomly positioned hole on a flat surface. Key features of the environment include:

- **Dynamic Initialization:** Each episode begins with randomized positions for the ball and the hole, ensuring varied training scenarios.
- **Physics Integration:** MuJoCo accurately simulates the dynamics of the robotic arm, the ball, and their interactions, including collisions and realistic movement constraints.
- **Action and Observation Management:** Actions involve the Cartesian movement of the end-effector, while observations include the ball's position, velocity, and rotation, as well as the end-effector's state.
- **Reward Mechanism:** The reward function incentivizes the arm to minimize the distance between the ball and the hole, guiding its learning process.

3.2 Development Learnings

The environment was developed based on FetchPush Env from the gymnasium robotics library:

- Custom XML files define the robot's structure and task setup, including the robot arm, the ball, and the target hole.

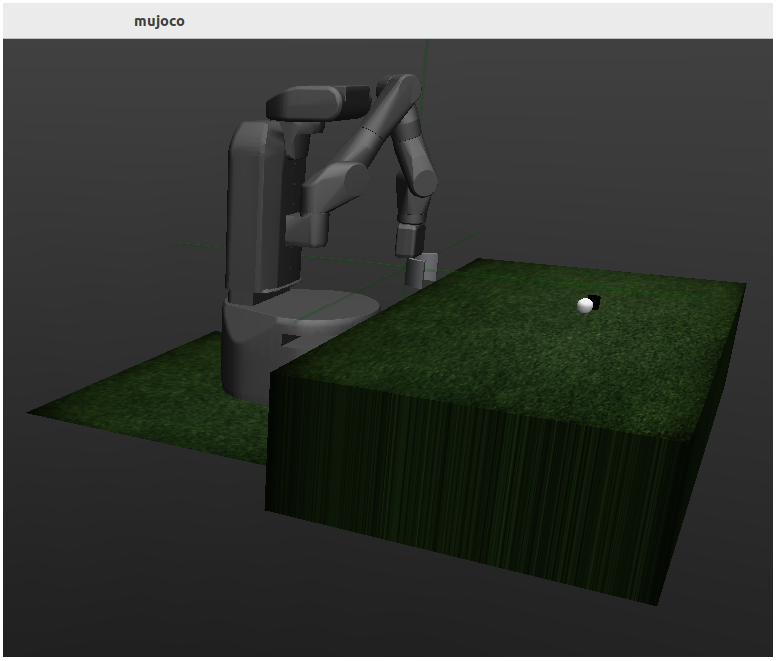

Figure 1: Environment consisting of the 7-DOF robotic Arm, Goal, and the object.

- The `PushEnv` class inherits from `MujocoRobotEnv`, customizing the robotic arm's behavior and observations for this specific task.
- Functions such as `compute_reward` and `_get_obs` calculate the rewards, observations, and actions tailored to the pushing task.

4 Neural Network Structure

The neural network used in this project is designed to serve as both the actor and critic in the Proximal Policy Optimization (PPO) algorithm. Its architecture processes observations and goals separately, combining their features to make predictions for actions and value estimates. This modular design ensures efficient learning in a dynamic environment, where the relationships between observations and goals are critical for successful task completion.

4.1 Key Components

The neural network is structured as follows:

- **Observation Network:** Processes the state observations, such as the end-effector's position, ball's velocity, and relative positions. Features are extracted through fully connected layers with normalization and activation functions.
- **Goal Network:** Processes the desired and achieved goals separately, combining them to understand the goal-specific context. This network also includes fully connected layers with normalization and activation functions to extract meaningful features.
- **Combined Features:** The outputs of the Observation and Goal networks are concatenated to form a comprehensive feature vector, which serves as the input for the Actor and Critic networks.

- **Actor and Critic Networks:** The Actor network predicts actions for the robotic arm, while the Critic network evaluates the value of the current state. Both networks use fully connected layers with shared architectural principles to ensure consistency in learning.

4.2 Changes Made to Neural Network (from last report)

- **Multi-Headed Attention Layer:** After the Observation and Goal networks, a multi-headed attention layer was added. This layer dynamically focuses on relevant observation-goal relationships, improving the network's ability to process complex and variable input data efficiently.

- **Residual Blocks:** After the Actor and Critic networks, residual blocks were introduced to facilitate deeper architectures. These blocks mitigate the vanishing gradient issue and promote the reuse of features learned in earlier layers. Each residual block contains one fully connected layer with input and output sizes of 256, followed by LayerNorm and ReLU activation functions.

The neural network architecture, with its modular design and added enhancements, is specifically tailored for use with the PPO algorithm. By combining attention mechanisms and residual connections, the network achieves improved stability and efficiency in training, enabling the robotic arm to learn optimal policies effectively.

Figure 2: Neural Network structure used for PPO Actor-Critic.

5 Learning Agent Training

The training of the learning agent in this project revolves around the Proximal Policy Optimization (PPO) algorithm, leveraging the neural network structure described in the previous section. The training process aims to optimize the policy and value networks for the 7-DOF robotic arm to push a ball to a desired goal in a simulated environment.

We explored two different reward structures during training:

- **Sparse Rewards:** Rewards are only provided when the ball reaches the goal within a defined threshold, making training slower but closer to real-world applications.
- **Dense Rewards:** The reward is calculated as the negative Euclidean distance between the ball and the goal. This provides continuous feedback to the agent, leading to faster convergence.

Using the dense reward structure and the PPO algorithm, we obtained the losses and rewards depicted in the graphs below. The figures illustrate the trends of total loss, value loss, policy loss, and total reward per episode during training. These results reflect the network's ability to minimize loss and improve rewards over time.

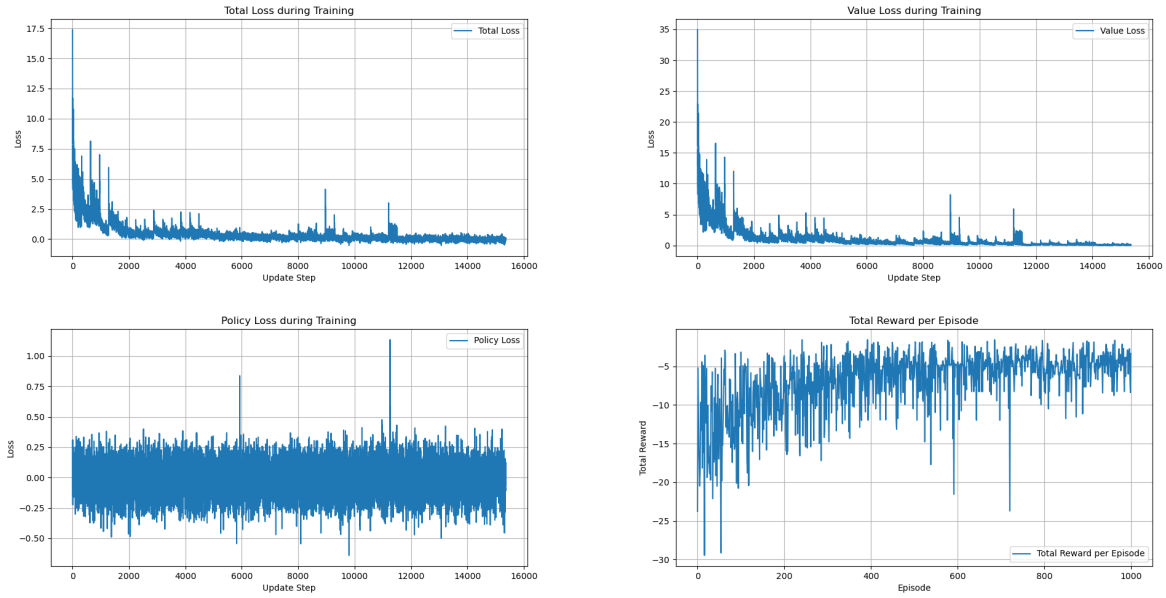

Figure 3: Training Results: (Top-left) Total Loss, (Top-right) Value Loss, (Bottom-left) Policy Loss, (Bottom-right) Total Reward per Episode for Dense Reward Structure.

Summary of Training Results

The training results reveal key insights into the performance of the PPO algorithm:

- **Loss Minimization:** Both total loss and value loss decrease consistently over training steps, indicating that the network effectively optimizes the policy and value functions.
- **Policy Stability:** The policy loss fluctuates but remains stable overall, reflecting balanced exploration and exploitation during training.
- **Improvement in Rewards:** The total reward per episode improves steadily, demonstrating the agent's ability to push the ball closer to the goal with higher accuracy and efficiency over time.

Agent Testing and Evaluation

After training, the agent was tested on 100 episodes with non-random initial positions of the ball and goal. The cumulative success rate over these episodes was plotted, as shown in the figure below:

Figure 4: Cumulative Success Rate over Test Episodes.

The results demonstrate a strong performance, with the agent achieving a cumulative success rate of 88%. This indicates that in most episodes, the robotic arm was able to successfully push the ball into the hole. The high success rate validates the effectiveness of the dense reward structure and the Proximal Policy Optimization (PPO) algorithm in training the agent for precise control tasks.

The above graphs correspond to the dense reward structure with non-random initialization of the ball position. Below, we present the total reward graph for the dense reward structure with random initialization of the ball's position:

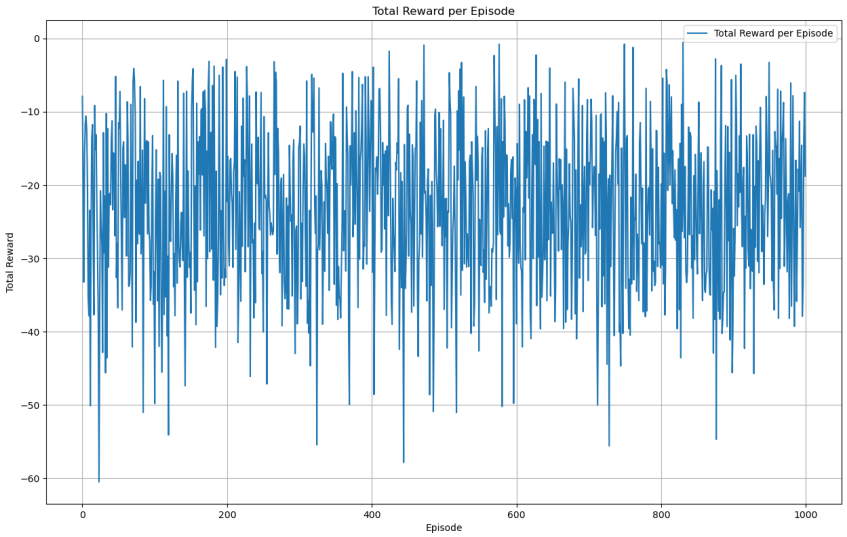

Figure 5: Total Reward per Episode for Dense Reward Structure with Random Initialization.

Comparison of Dense Reward with Random and Non-Random Initialization

From the total reward graphs, the following key differences can be observed:

- With non-random initialization, the agent consistently learns and achieves higher rewards as it repeatedly trains in a fixed setup. This setup allows the agent to converge faster to a stable policy.
- With random initialization, the rewards show higher variability due to the agent encountering diverse starting scenarios. This reflects the agent's adaptability but requires more training time to stabilize and achieve consistent results.

Training with random initialization better prepares the agent for real-world scenarios, where initial conditions vary, but it demands more computational resources and time. Extending the training duration for random initialization is expected to improve the overall performance, allowing the agent to generalize its behavior more effectively across varied situations.

We also trained the model using a sparse reward structure for 500 episodes. Below are the graphs representing the total loss, value loss, policy loss, and total rewards for the sparse reward structure with non-random initialization:

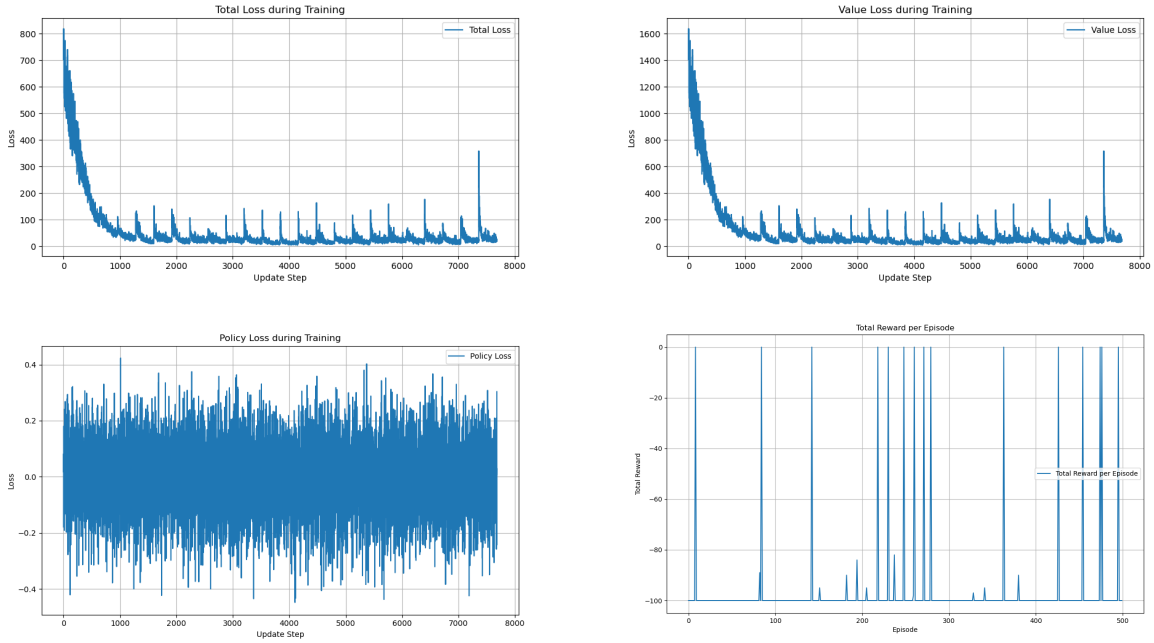

Figure 6: Training Results for Sparse Reward + Non-Random Initialization: (Top-left) Total Loss, (Top-right) Value Loss, (Bottom-left) Policy Loss, (Bottom-right) Total Reward per Episode.

Comparison of Sparse and Dense Reward Structures

The following comparisons can be made between the training results for the sparse reward structure and the dense reward structure with non-random initialization:

- **Training Stability:** The sparse reward structure shows higher variability in total and value losses compared to the dense reward structure. This indicates that the model struggles to converge effectively with sparse rewards.
- **Total Rewards:** Sparse rewards result in significantly lower total rewards across episodes, as the model receives limited feedback, making it harder to learn optimal strategies.
- **Learning Speed:** The dense reward structure facilitates faster learning and stabilizes quicker due to continuous feedback.

Conclusion

The dense reward structure with non-random initialization consistently outperforms the sparse reward structure in terms of stability, convergence, and reward maximization. While sparse rewards align more closely with real-world scenarios, further fine-tuning is required to improve performance. Techniques such as reward shaping, curriculum learning, or increasing the training duration could enhance the model's ability to perform well under sparse reward conditions.

6 Learnings and Reflection

The development and training of the robotic arm using reinforcement learning (RL) provided several insights into the advantages and limitations of the approach.

6.1 Advantages of Our RL Approach

- **Adaptability in Dynamic Environments:** Unlike traditional algorithms which require explicit trajectory planning and are computationally expensive in high-dimensional spaces, RL enables the agent to learn optimal behaviors dynamically. This makes it more suited to tasks with changing goals and object properties.
- **Enhanced Neural Network Architecture:** The inclusion of multi-headed attention layers in the neural network allowed the model to dynamically focus on relevant observation-goal relationships, improving learning efficiency. Additionally, the use of residual blocks helped subdue the vanishing gradient issue, facilitating deeper architectures and promoting feature reuse across layers.
- **Efficiency with Dense Rewards:** The dense reward structure provided continuous feedback during training, leading to faster convergence and enabling the agent to learn precise control strategies for the robotic arm.

6.2 Disadvantages and Limitations

- **Jerky Movements:** Despite high success rates, the trained model exhibited jerky and abrupt movements during simulation. This could pose challenges for transferring the policy to a physical robot, where smoother motion is essential for hardware longevity and task precision.
- **Vanishing Gradient Problems:** While the residual connections alleviated some gradient-related issues, deeper architectures still showed occasional instability during training, particularly when initializing with sparse reward structures.
- **Dependence on Dense Rewards:** The model heavily relies on the dense reward structure for effective learning. Performance on sparse reward structures was limited, requiring additional fine-tuning or reward shaping methods to achieve similar success rates.

6.3 Future Improvements

To address these limitations and further enhance the approach, the following improvements can be considered:

- Implementing motion-smoothing techniques to reduce jerky movements and ensure smoother control transitions.

- Exploring advanced optimization techniques or alternative activation functions to further mitigate vanishing gradient issues.
- Developing hybrid reward structures that combine dense and sparse feedback to make the model more robust and applicable to real-world scenarios.

The learnings from this project highlight the potential of reinforcement learning for complex robotic tasks while emphasizing the need for careful design choices to address inherent challenges and limitations.