EVALUATING HIERARCHICAL REINFORCEMENT LEARNING FOR ROBOT MANIPULATION

A PROJECT REPORT

submitted by

SREEJITH KRISHNAN R
TVE18ECRA17

to

the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree
of

Master of Technology

in

Electronics and Communication Engineering

with specialization in

Robotics and Automation

CET Centre for Interdisciplinary Research

College Of Engineering
Thiruvananthapuram

NOVEMBER 2019

DECLARATION

I undersigned hereby declare that the project report ***"EVALUATING HIERARCHICAL REINFORCEMENT LEARNING FOR ROBOT MANIPULATION"***, submitted for partial fulfillment of the requirements for the award of degree of Master of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under supervision of **Prof. Linu Shine**, Assistant Professor, Department of Electronics and Communication Engineering, College of Engineering, Thiruvananthapuram. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place:
Date: Sreejith Krishnan R

CET CENTRE FOR INTERDISCIPLINARY RESEARCH

COLLEGE OF ENGINEERING TRIVANDRUM

CERTIFICATE

This is to certify that the report entitled **"EVALUATING HIERARCHICAL REINFORCEMENT LEARNING FOR ROBOT MANIPULATION"** *submitted by* **SREEJITH KRISHNAN R,** *to the APJ Abdul Kalam Technological University in partial fulfillmentof the requirements for the award of the Degree of* **Master of Technology** *in* **Electronics and Communication Engineering** *with specialization in* **Robotics and Automation** *is a bonafide record of the project work carried out by him under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.*

Dr. Ranjith S. Kumar
Associate Professor
Dept. of Mechanical Engg.
College of Engineering
Trivandrum

Prof. Linu shine
Assistant Professor
Dept. of Electronics Engg.
College of Engineering
Trivandrum

Prof. P.S. Shenil
Assistant Professor
Dept. of Electrical Engg.
College of Engineering
Trivandrum

ACKNOWLEDGEMENT

I express my gratitude to **Prof. Linu Shine**, Associate Professor and Guide, Department of Electronics and Communication Engineering, College of Engineering, Thiruvananthapuram for all the necessary help and guidance extended to me towards the fulfillment of this work.

With my deep sense of gratitude, I would like to thank my respected teachers, **Dr. Ranjith S. kumar**, Associate Professor, Department of Mechanical Engineering and **Prof. P.S. Shenil**, Assistant Professor, Department of Electrical and Electronics Engineering College Of Engineering, Thiruvananthapuram, for their valuable suggestions and solutions.

I hereby acknowledge my sincere thanks to **Dr.Sindhu G**, Dean Research, College Of Engineering, Thiruvananthapuram, for her motivation and inspiration provided.

I also thank all the staff of College Of Engineering, Thiruvananthapuram for their wholehearted support and cooperation.

Finally I would like to acknowledge my deep sense of gratitude to all my well-wishers and friends who helped me directly or indirectly to complete this work.

SREEJITH KRISHNAN R

ABSTRACT

In order to assist in general tasks, autonomous robots should be able to interact with dynamic objects in unstructured environments. Robot manipulation of objects is the key component in all autonomous robot applications requiring interaction with environment. In vision based robot manipulation, robot has to measure the environment state using camera and take actions according the measured state and goal. The main challenges in here are interpreting the noisy high dimensional data from camera and deciding actions according to stochastic and non stationary environment state.

Work done by Michel Breyer et. al. [1] found that using autoencoder to reduce dimensionality of camera data and using curriculum learning reduced the training time of agents. Also by using shaped reward functions instead of sparse reward function, they obtained 98% success rate on simulated environment. They also found that using RANSAC for detecting and filtering surfaces from camera data while using policies trained from simulated environment on real robot gave 78% success rate.

This project will evaluate the performance of hierarchical reinforcement learning methods for vision based robot manipulation. Traditional reinforcement learning methods are data inefficient (require lot of data for training), difficult to scale (due to large action and/or state space) and brittle due to over specialisation (difficult to transfer their experience to new even similar environments) [2]. Hierarchical reinforcement learning are intended to address these issues by learning to operate on different levels of temporal abstraction. Using this method, the entire task of robot manipulation can be split into smaller sub-tasks like reaching, grasping, lifting etc.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Background

In order to assist in general tasks, autonomous robots should be able to interact with dynamic objects in unstructured environments. Robot manipulation of objects is the key component in all autonomous robot applications requiring interaction with environment. In vision based robot manipulation, robot has to measure the environment state using camera and take actions according the measured state and goal. The main challenges in here are interpreting the noisy high dimensional data from camera and deciding actions according to stochastic and non stationary environment state.

Methods for robot manipulation can be broadly classified into traditional and data driven methods. In traditional methods, data from camera is interpreted using computer vision algorithms and actions are hand coded based on interpreted state. This approach works well for robots deployed for specific task like in automated production lines. But in stochastic and non stationary environments, it is not possible to hand code actions for all environment states. On the other hand data driven methods, particularly reinforcement learning have shown great potential in this use case. In reinforcement learning, an agent learns to take actions (policy) based on measured environment state by interacting with the environment through trial and error for maximising a feedback signal (reward or value function). The main challenge in this method is requirement of large amount of data for agent to learn. Collecting large amount of data from real robot will be slow and expensive and might require manual intervention. Instead, for faster and cheaper development, agent can be trained on simulated robot manipulation environment. However policies trained on simulated environments are not directly transferable to real robots due to various differences between simulation and real environments.

1.2 Research Gap

Current reinforcement learning methods used for robot manipulation have following limitations

- Low success rate when transferring policies from simulation to real world. Since it is extremely difficult and expensive to collect data and train using real robot, transferring policies from simulation to real world with minimal performance degradation is important
- Low sample efficiency requiring lot of data for training
- Low interpretability of learned models
- Difficult predictability when transferring of learned skills

Current hierarchical reinforcement learning method (RHPO) used for robot manipulation have following limitations

- Transferring policies from simulation to real world is not evaluated
- Requires manual task decomposition
- Low interpretability

1.3 Objectives

- Compare hierarchical reinforcement learning method RHPO with reinforcement learning methods DDPG and PPO for vision based robot manipulation
- Evaluate transferability of skills for similar robot manipulation tasks
- Explore methods for automatic task decomposition in RHPO
- Explore methods to explain RHPO sub-task policies

1.4 Outline of Report

- Chapter 1 gives an introduction to project and outline of this report
- Chapter 2 gives an overview of previous works done
- Chapter 3 gives details of simulation setup used to train reinforcement learning agents

- Chapter 4 gives brief introduction to various reinforcement learning algorithms evaluated and their performance in various environments

Chapter 2

Literature Review

Overall methods for robot manipulation can be classified into traditional and data driven methods. Traditional methods are rule based and can be applied for structured and deterministic environments. For stochastic and unstructured environments, data driven methods are used. In this literature review, we will focus of main data driven robot manipulation methods

2.1 Reinforcement learning methods

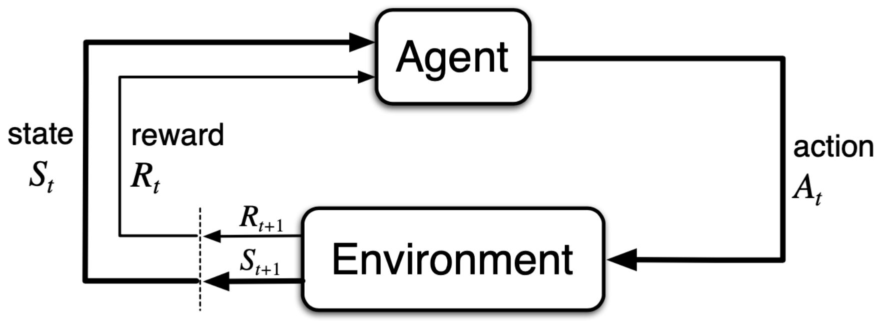


Figure 2.1: Reinforcement Learning; Source: [3]

In reinforcement learning, an agent learns to take actions (policy) based on measured environment state by interacting with the environment through trial and error for maximising a feedback signal (reward or value function). The main challenge in this method is requirement of large amount of data for agent to learn. Collecting large amount of data from real robot will be slow and expensive and might require manual intervention. Instead, for faster and cheaper development, agent can be trained on simulated robot manipulation environment. However policies trained on simulated environments are not directly transferable to real robots due to various differences between simulation and real environments.

2.1.1 Comparing Task Simplifications to Learn Closed-Loop Object Picking

Work done by Michel Breyer et. al. [1] found that using autoencoder to reduce dimensionality of camera data and using curriculum learning reduced the training time of agents. Also by using shaped reward functions instead of sparse reward function, they obtained 98% success rate on simulated environment. They also found that using RANSAC for detecting and filtering surfaces from camera data while using policies trained from simulated environment on real robot gave 78% success rate. Figure 2.2 shows the network architecture used in this work.

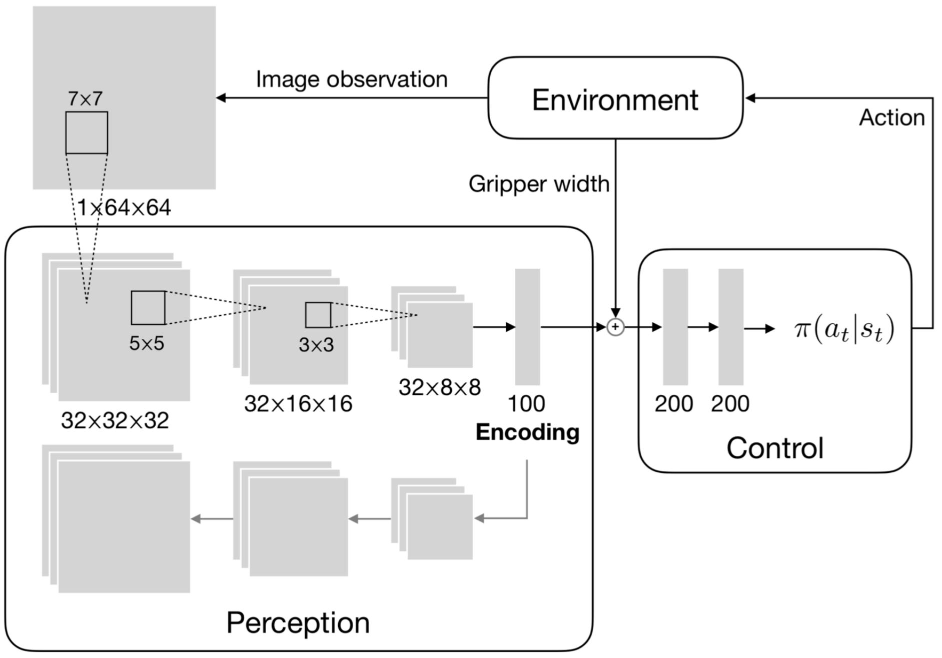


Figure 2.2: Comparing Task Simplifications to Learn Closed-Loop Object Picking: Network architecture; Source: [1]

2.2 Hierarchical reinforcement learning methods

Traditional reinforcement learning methods are data inefficient (require lot of data for training), difficult to scale (due to large action and/or state space) and brittle due to over specialisation (difficult to transfer their experience to new even similar environments) [2]. Hierarchical reinforcement learning are intended to address these issues by learning to operate on different levels of temporal abstraction.

2.2.1 Regularized Hierarchical Policies for Compositional Transfer in Robotics

This method proposes [4] hierarchical and modular policies for continuous control. The modular hierarchical policy used is defined as:

$$\pi_\theta(a|s,i) = \sum_{O=1}^{M} \pi_\theta^L(a|s,o)\pi_\theta^H(o|s,i) \tag{2.1}$$

Where π^H is the high level switching controller and π^L is the low level sub policy. The objective is to optimize

$$\underset{q}{max} J(q,\pi_{ref}) = E_{i\sim I}\left[E_{\pi,s\sim D}\left[\sum_{t=0}^{\infty}\gamma^t r_i(s_t,a_t)|s_{t+1}\sim p(.|_t,a_t)\right]\right] \tag{2.2}$$

subject to constraint

$$E_{s\sim D,i\sim I}\left[KL(q(.|s,i)||\pi_{ref}(.|s,i))\right] \leq \epsilon \tag{2.3}$$

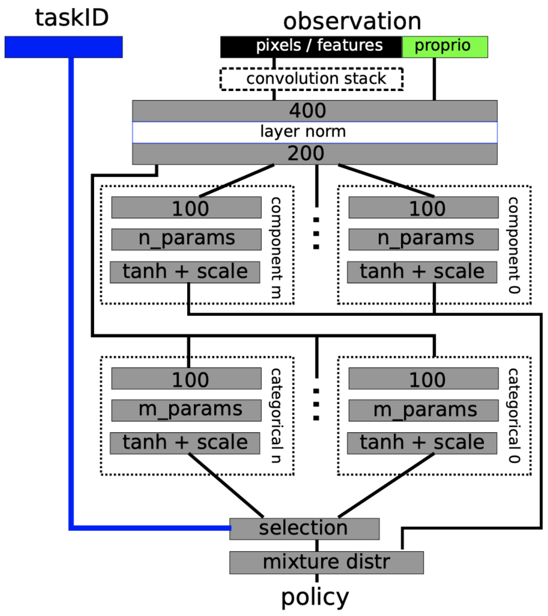


Figure 2.3: RHPO: Network architecture

Chapter 3

Simulation

All reinforcement learning agents are first trained in a simulated environment. In this project the simulated environment is created using Bullet Physics Simulator.

3.1 Standard setup

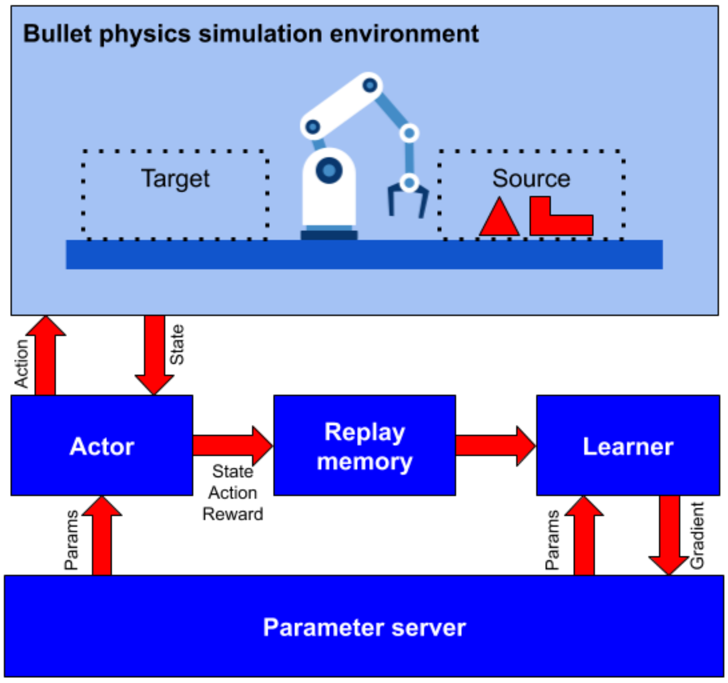


Figure 3.1: Standard simulation architecture

To reduce training time, agent training via simulation is parallelized and distributed similar to GORILA architecture [5]. In this project we use Ray RLLib [6] which provides abstractions for distributed reinforcement learning.

3.1.1 Simulation environment

Simulation environment is developed using Bullet Physics simulator python module. Environment will be setup for a specific task like table clearing or stacking of objects. Environment will accept actions from specific task which must be in specific format according to action space of environment. When environment receives an action, it will apply the action to simulated environment using PyBullet APIs. After action is completed, the environment will record the state according to state space of environment and calculate the reward according to task setup of environment. Environment returns state and reward after action is applied.

PyBullet Simulator

PyBullet is a python module for Bullet Physics C SDK used for physics simulation in robotics, games, visual effects and machine learning, with a focus on sim-to-real transfer [7]. PyBullet can load visual, physical and other properties of a body from URDF, SDF and Mujoco formats. If required, 3D mesh of bodies can be loaded directly using PyBullet APIs. For simulating robots, PyBullet supports forward and inverse kinematics, collision detection, coordinate transformations, forward dynamics simulation, inverse dynamics computation, joint state position, velocity and force/torque control.

PyBullet supports rendering using CPU renderer and OpenGL renderer. Visualization can be optionally turned off in PyBullet. With visualization disabled, PyBullet uses CPU renderer for rendering images captured by camera API. This is especially useful in reinforcement learning where we want to collect data from simulated environment as fast as possible.

3.1.2 Actor

Actor is responsible for reading policy from the parameter server and execute actions is the simulation environment and save the action, returned state, returned reward in experience replay memory. When an actor is started, it will create a new simulation environment process.

- Start new simulation environment process
- Begin new episode

- Read policy ($\pi(a_t|s_t)$) and environment state (s_t). Evaluate and select action a_t for state s_t by evaluating policy $\pi(a_t|s_t)$. Execute the action a_t on simulation environment environment and read the returned reward r_t and new state of environment s_{t+1}. Save $[a_t, s_t, r_t, s_{t+1}]$ to replay buffer
- End episode when simulation environment terminates an episode

3.1.3 Replay buffer

Replay buffer stores the trajectory of episodes. Trajectory of an episode is a list of $[a_t, s_t, r_t, s_{t+1}]$. Since multiple actor process are supposed to add observations to replay buffer, replay buffer is usually a distributed database accessible by actor processes. We use redis in memory database is used as replay buffer database.

3.1.4 Learner

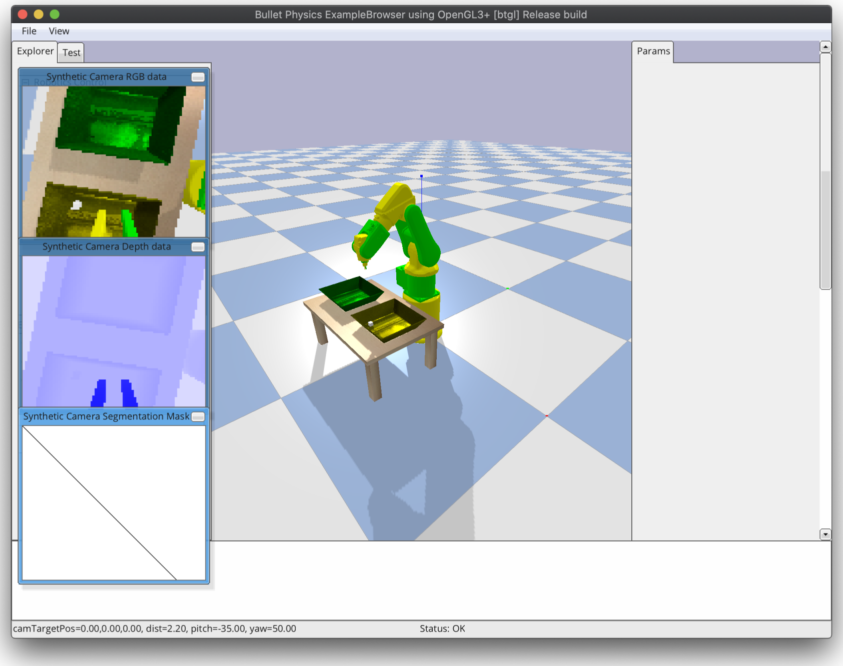
Learner reads the episode trajectories stored in replay buffer and optimize the policy stored in parameter server to maximize rewards. Learner process is specific to a reinforcement learning algorithm. Eg:- DDPG, PPO. Learner process uses deep learning frameworks like tensorflow or pytorch to model the policy, calculate the gradients for loss functions and do other computations.

3.1.5 Parameter server

Parameter server stores the policy. It is accessible by both actor and learner and also accepts gradient from learner and apply the gradient to update the policy represented by parameter server.

3.2 Table Clearing Environment

In table clearing environment, main components are a 6 axis ABB IRB 120 robot with MetalWork W155032001 parallel jaw gripper and a table with yellow and green tray. Object at random position and orientation is inserted into the source tray (yellow). The task is to move the object from source tray (yellow) to destination tray (green).


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build
File
View
Explorer
Test
Synthetic Camera RGB data
Synthetic Camera Depth data
Synthetic Camera Segmentation Mask
Params

camTargetPos=0.00,0.00,0.00, dist=2.20, pitch=-35.00, yaw=50.00 Status: OK

Figure 3.2: Table clearing pybullet environment

3.2.1 State space

A RGB-D camera is mounted on the end effector of the gripper. The output of this camera is shown in lFigure 3.3. The state space of the environment is a tensor of shape $84 \times 84 \times 4$. The first three channels are RGB pixel values and fourth channel is the depth value.

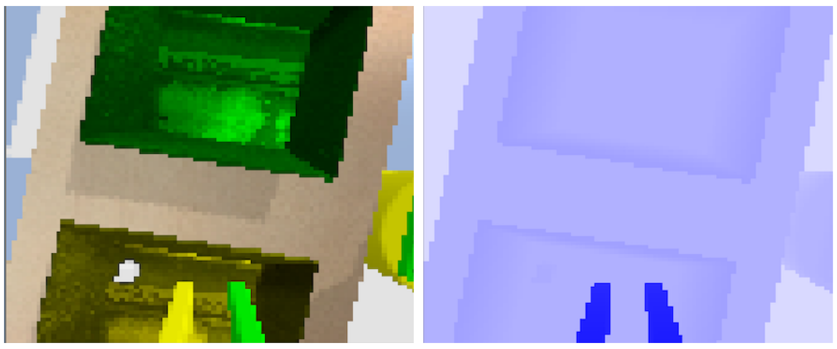


Figure 3.3: Gripper camera output; left: RGB, right: depth

3.2.2 Action space

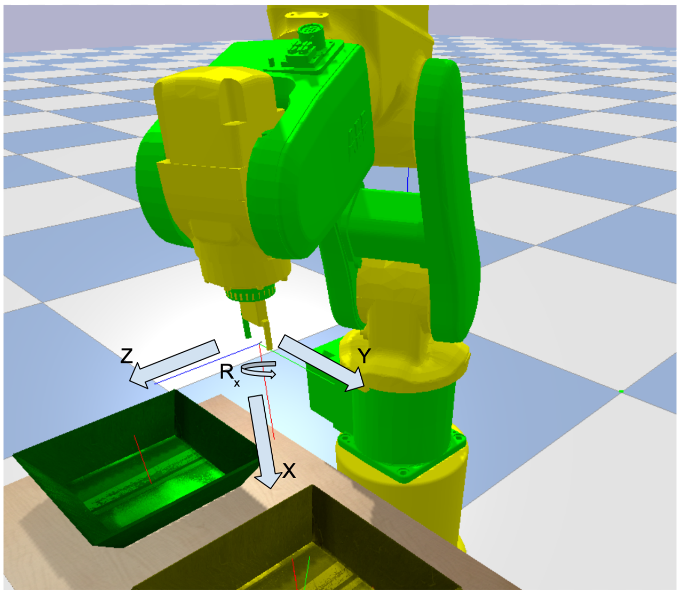


Figure 3.4: Table clearing environment action coordinate system

The end effector position can be controlled by making $[\delta x, \delta y, \delta z]$ movements with respect to $X, Y \& Z$ axis of a coordinate system attached to gripper as shown in Figure 3.4. The end effector can be rotated about approach vector by δr_x. The gripper fingers can be closed by setting binary variable *open*. The maximum/minimum values of a single action is shown in Table 3.1

Action	Limit
$\delta x, \delta y, \delta z$	$[-1, 1]$ cm
δr_x	$[-10, 10] deg$
open	1, 0

Table 3.1: Table clearing environment action space

3.2.3 Reward

Let s_t and s_{t-1} be environment state at time t and $t-1$ respectively. The total reward returned by the environment for action a_t at time t is the sum of following rewards

- If object is not grasped and object has not moved far from initial position and gripper to object distance at time t is less than $t-1$, reward is +1 else -1

- If object is grasped and gripper to destination tray distance at time t is less than $t-1$, reward is +1 else -1
- For each action, reward of -1 is given to minimize time to complete task
- If body of robot including gripper touches any body other than target object, robot is assumed to be collided and a reward of -1000 is given
- If at time $t-1$, object is not grasped and at t, object is grasped, then a reward of +100 is given. Object is assumed to be grasped when target object is only in contact with gripper fingers and object is having a height of at least 5cm above source tray
- If at time $t-1$, object is grasped and at t, object is not grasped and target object is not at destination tray, object is assumed to be dropped and a reward of -200 is given
- If at time $t-1$, object is not at destination tray and at t, object is at destination tray, then object is assumed to be delivered and reward of +200 is given

3.2.4 Episode termination

A simulation episode is terminated at following conditions

- Robot or gripper body is in contact with any body other than target object (collision)
- Target object reached destination tray (delivered)
- Duration of episode is greater than 3 minutes
- Number of actions taken in episode is greater than 3000

Chapter 4

Results

4.1 Proximal Policy Optimization (PPO)

PPO is an on policy, stochastic and policy gradient based reinforcement learning method [8]. The objective function of PPO is

$$L^{CLIP}(\theta) = \hat{E}_t \left[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{4.1}$$

Where

- θ is the policy parameter
- $\hat{E}_t$ denotes the empirical expectation over timesteps
- r_t is the ratio of probability under the new and old policies, respectively
- $\hat{A}_t$ is the estimated advantage at time t
- ϵ is a hyperparameter, usually 0.1 or 0.2

4.1.1 Architecture

PPO requires two neural networks, value network to calculate advantage of taking an action and policy network for predicting mean and variance of an action to maximize reward at a given state. We use 3 layer CNN for extracting features from image of gripper camera. The input image shape is $84 \times 84 \times 4$. Fourth channel is depth value. The value network and policy network can share this 3 layer CNN for feature extraction. But since loss functions of value network and policy network are different, sharing the CNN layers might optimize the CNN layers for either value or policy network. This can be fixed by scaling the loss functions of value and policy network to same scale.

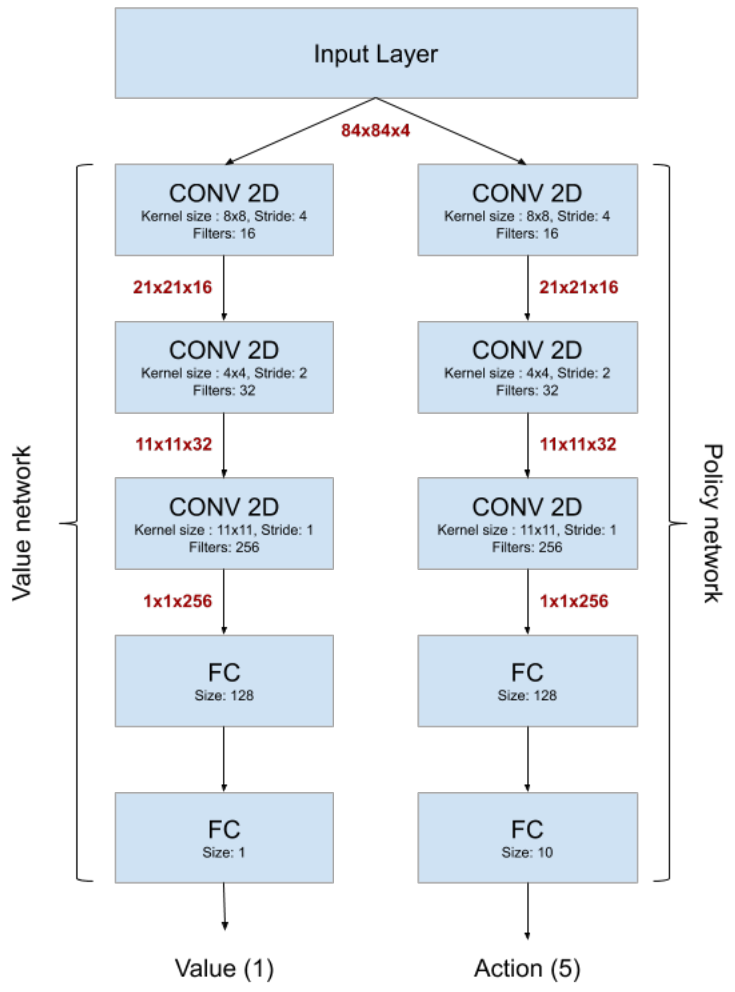


Figure 4.1: PPO network architecture

4.1.2 Table clearing environment

Simulation

After training PPO agent on table clearing environment using 8 core 42GB RAM Nvidia T4 GPU system for 4M timesteps, results are shown below

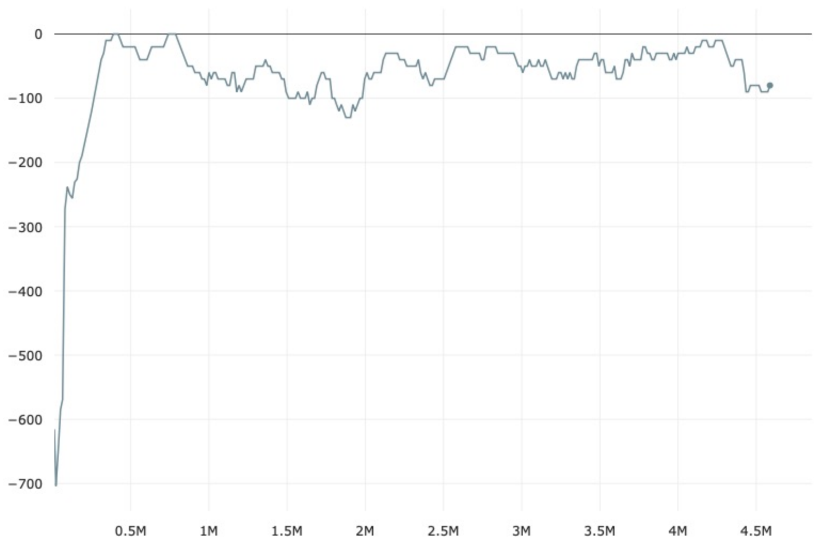


Figure 4.2: Collision penalty. Optimum value is 0

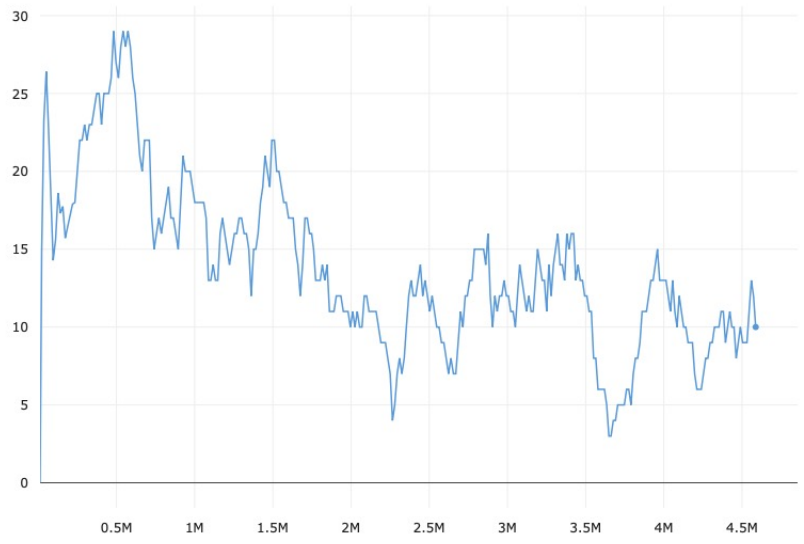


Figure 4.3: Grasp reward. Optimum value is 100

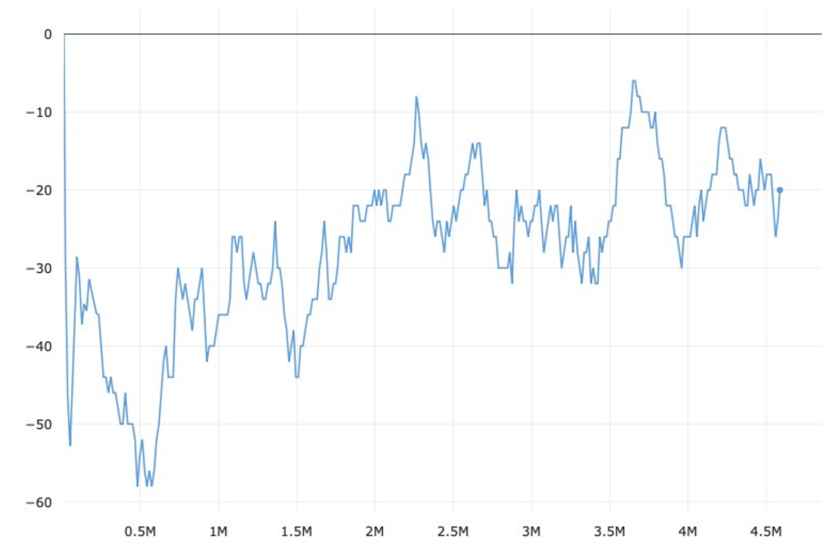


Figure 4.4: Drop penalty. Optimum value is 100

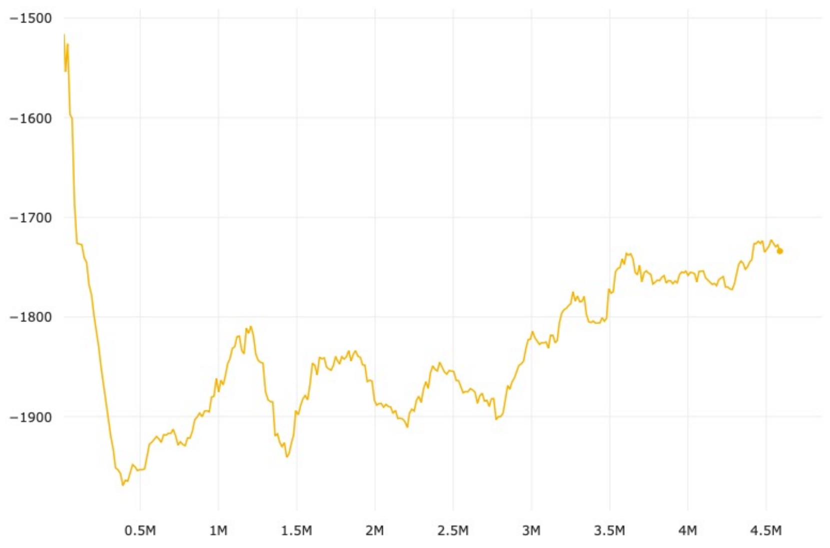


Figure 4.5: Mean episode reward. Optimum value ≥ 200

For attaining optimum reward, network needs to be trained for approximately 80M timesteps.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

- Reward shaping is critical. Small changes in reward function can change the learning process.
- Providing +ve reward when end effector moves towards target and -ve reward when end effector moves away will not work. It will teach robot to move the target object instead of grasping it.
- Initializing each episode at a random state like grasped and not grasped can improve training speed
- PPO is not sample efficient and requires lot of training data. After 4M timesteps, PPO was not able to learn optimum policy. Training for more timesteps (80M) might help.

5.2 Future works

- Training PPO for more timesteps (80M)
- Including DDPG and RHPO baseline models
- Evaluating baseline models on real robot
- Evaluating skill transferability in RHPO
- Exploring methods for automatic task decomposition
- Exploring methods for sub policy interpretability in RHPO

Bibliography

[1] M. Breyer, F. Furrer, T. Novkovic, R. Siegwart, and J. Nieto, “Comparing task simplifications to learn closed-loop object picking using deep reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 4, pp. 1549–1556, April 2019.

[2] T. Gradient, “The promise of hierarchical reinforcement learning.” https://thegradient.pub/the-promise-of-hierarchical-reinforcement-learning/, 2019.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* The MIT Press, second ed., 2018.

[4] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, M. Neunert, T. Hertweck, T. Lampe, N. Siegel, N. Heess, and M. Riedmiller, “Regularized hierarchical policies for compositional transfer in robotics,” 2019.

[5] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, “Massively parallel methods for deep reinforcement learning,” *CoRR*, vol. abs/1507.04296, 2015.

[6] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, “Ray rllib: A composable and scalable reinforcement learning library,” *CoRR*, vol. abs/1712.09381, 2017.

[7] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning.” http://pybullet.org, 2016–2019.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *CoRR*, vol. abs/1707.06347, 2017.