# Final Project Report for ESE 650
# Reinforcement Learning Based Dynamic Grasping

**Yihang Xu** YIHANGXU@SEAS.UPENN.EDU

## Abstract

In order to tackle dynamic grasping, I implemented a scalable variant of policy gradient algorithm, PPO (Proximal Policy Optimization). I apply it to train a 6-Dof robot arm (UR-10) with a unactuated end effector to learn to pick up a cuboid by model-free policy. The experiments show the that PPO could achieve satisfying training results with high update efficiency. The performance of the agent can be viewed at: https://drive.google.com/file/d/1pwhlTcNlxtmdFnFmCYU_6hzcIhi02UaB/view?usp=sharing

## 1. Introduction

### 1.1. Problem Statement

Imagine that the actuators of a robot's end effector are broken but we still have to finish the job before it can get repaired. In this scenario, simple functions like picking objects using the original strategy are disabled, we have to apply a non-equilibrium method to tackle this problem. The picking process can be divided into two parts: reach the object and pick it up. The first part can be done smoothly using path planning. So in this project, I will focus on picking up the object through reinforcemnet learning (PPO).

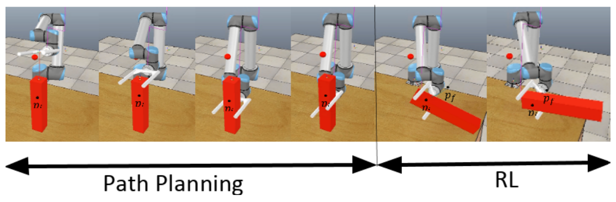

*Figure 1.* Picking path (from Caio)

### 1.2. Project Overview

In this problem, I use a robot arm from Universal Robots (UR-10) and a self-designed unactuacted end effector.

The main project structure is shown in *Fig.*2. The agent is trained with an actor-critic cycle. In each iteration, the agent interact with the environment with the action given by the policy network in the simulator to collect datasets, which will be used to generate losses to update the policy network and the value network. The details of the simulation are discussed in section 2 and the algorithms are described in section 3.

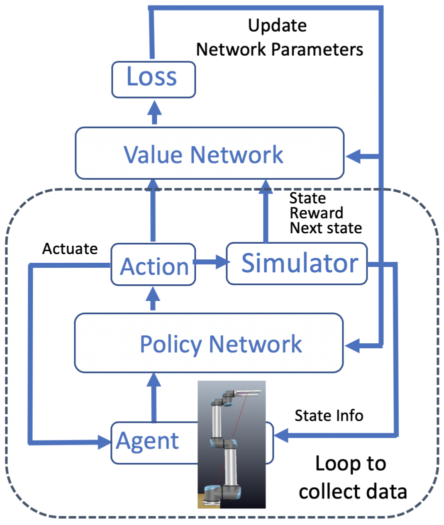

*Figure 2.* Project Overview

## 2. Simulation Setup

In this section, I will first introduce the simulation environment and the robot and cuboid inside. Then I will discuss the dynamics of this system which I use to generate the path. The environment and dynamics are originally built and analysed by Caio Mucchiani of GRASP.

### 2.1. Simulation Environment Description

*CoppeliaSim* is a robot simulator with integrated development environment. It's based on a distributed control architecture: each object/model can be individually controlled via an embedded script. In this project, I use python to write controller with a toolkit *PyRep*, which is built on top of *CoppeliaSim* and has simple and flexible API for controlling and scene manipulation.

### 2.2. Agent and Target Setup

The angent is the UR-10 robot as shown in *Fig.*3.It is mounted on the edge of a desk with 0.8m height with an upright initial position. *Fig.*4 shows the end effector and the cuboid in the simulation. The end effector is specially designed for picking up cuboid objects in a certain size range, and it's friction coefficient is set at 0.7, which is a high value to simplify this problem.

The red cuboid with a size of $0.05 \times 0.05 \times 0.4$ is initially placed on the desk in front of the robot.

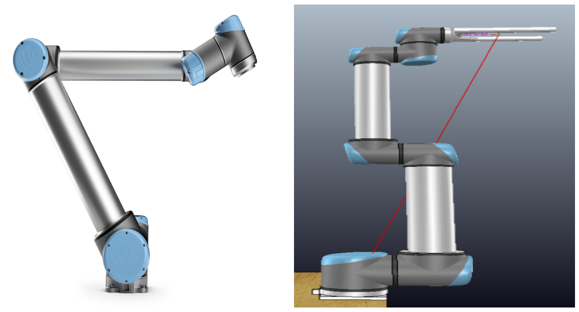
*Figure 3.* UR-10 and UR-10 in Simulation

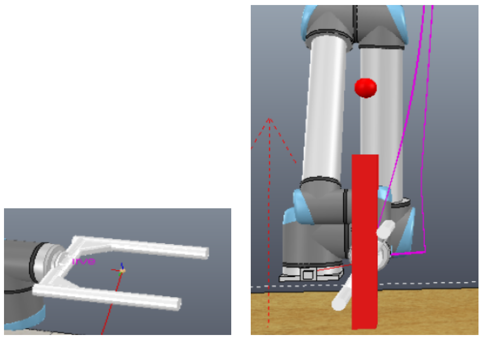
*Figure 4.* End Effector and Cuboid in Simulation

### 2.3. System Dynamics Introduction

The path of the end effector when picking is a parabola. As shown in *Fig.*5, when the end effector follows a parabola, it can reach a statically stable terminal state. The specific parabola I choose in this problem (*Fig.*6):

$$\gamma = z = y^2 - 0.4y + 1.04$$

To simplify this problem, this parabola is specifically designed so that the cuboid's initial position is at the minimum of the parabola. But this setting is not necessary because the robot should be able to pick up the cuboid along different paths after trained in more general settings.

Since a exact path is provided, the input of the policy is the current y coordinate ($y$) and the output are the velocity along y axis ($v_y$) and the angular velocity of the end effector ($\theta_{ee}$).

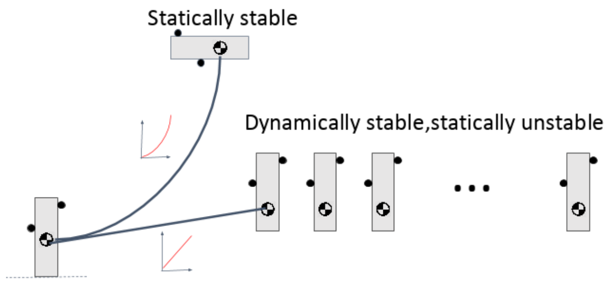

*Figure 5.* Picking Dynamics (from Caio)

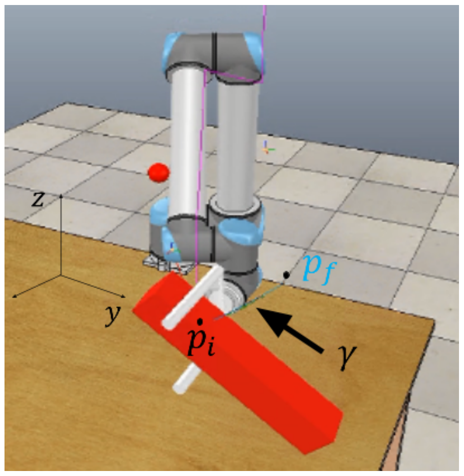

*Figure 6.* Planned Path (from Caio)

## 3. Algorithm Description

### 3.1. PPO

Policy gradient algorithms (5) is a common RL approach for continuous control. The original algorithm can have high variance and is sensitive to parameter settings. The Proximal Policy Optimization (PPO) algorithm addresses the problem with its robust performance, and it is simpler to implement and more efficient in sampling than other variants of policy gradient like TRPO (3). Also, PPO is more convenient to be used with recurrent neural networks (RNNs) and in a large-scale distributed setting.

PPO does the policy optimization process by minimizing a "surrogate" objective function using optimizers like SGD and Adam. The loss function has various versions of forms, one of which is called the clipped probability ratios:

$$L(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta), clip(r_t(\theta), 1-\epsilon, 1+\epsilon))\hat{A}_t] \quad (1)$$

$r_t(\theta)$ denotes the probability ratio $\frac{p_\theta(a_t|s_t)}{p_{\theta_{old}}(a_t|s_t)}$. Epsilon is a hyperparameter, say, $\epsilon = 0.2$. The goal of the clipped term $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ is to constrain the probability ratio inside the interval $[1-\epsilon, 1+\epsilon]$ so that the updated policy and the old policy used in sampling can stay close. Then I take the minimum of the unclipped and clipped objective. With that we ignore changes in policy that would make the objective function increase, and apply changes that decrease the objective. An alternative approach of the clipped surrogate loss is to apply a penalty on the defined KL divergence in order to achieve the same goal of limiting the policy divergence in the optimization process. Experiments have shown (3) that objective with KL penalty often performed worse than the clipped surrogate objective, so in this project I decided to implement the latter approach.

The advantage estimate $\hat{A}_t$ in equation 1 can be expressed in the following form, which is the sum of exponentially reduced future rewards minus the estimated value function (the baseline): $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V(s_t)$. However, a more popular method (4) is to run the policy for T time steps (by setting the maximum number of samples), get sampled data, and compute the variance-reduced $\hat{A}_t$ using the learned value function V:

$$\hat{A}_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} + \gamma^{T-t} V(s_T) - V(s_t) \tag{2}$$

Generalizing the equation, we can add a parameter $\lambda$ to change the influence of future reward estimates as follows. Note when $\lambda = 1$, it reduces to 2.

$$\begin{aligned} \hat{A}_t &= \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-1-t}\delta_{T-1} \\ where\ \delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \tag{3}$$

The pseudocode for the entire training process is shown in Algorithm 1. In each episode, I run policy in the environment to collect data before the cuboid falls. Then I compute the advantage estimates, construct the surrogate loss and optimize both the policy network and value network.

**Algorithm 1** Proximal Policy Optimization

```
for $itetation = 1, 2, ..., N$ do
    Run policy $\pi_\theta$ in environment for T timesteps, collect $s_t, a_t, r_t$
    Compute advantage estimates $\hat{A}_t$ (Equation 3)
    $\pi_{old} \leftarrow \pi_\theta$
    for $epoch = 1, 2, ..., M$ do
        Write surrogate loss (Equation 1), update policy network $\theta$ using gradient method
        Write value loss (mean-squared error), update value network using gradient method
    end for
end for
```

| Scenario | Reward |
|---|---|
| Every step | -1 |
| Cuboid falls | -20 |
| Cuboid stays on but not horizontal | $(z - z_0) \times 100 + 20$ |
| Cuboid horizontal and stable | +1000 |

*Table 1.* Rewards

## 4. Implementation and Results

### 4.1. Overview

**Agent** The agent I trained is the UR-10 robot, which has 6 DOF and is controlled by 6 joints in 3D space. The agent has a 1D continuous state space, and 2D continuous action space. The simplified conditoins discussed in section 2 makes the model and the training easier.

**Rewards** The rewards I designed are listed in *Table 1*

When the cuboid falls the agent receives a penalty of -20. When the cuboid stays on the end effector it receives a reward equals to the increased height compared to the initial height $(z - z_0)$ scaled by 100, and a reward of +20 to encourage stability. When the Cuboid reaches goal stably, it receives a reward of +1000

**Hyperparameters & Networks** The base hyperparameter (3) and network structures in the experiment are listed in Table 2 and 3.

### 4.2. PPO Implementation Details

Except from theoretical guidance from the research paper, I also explore and implement practical tricks to help the system converge in high efficiency.

**Optimizer Batch** To break the correlation of the sequential data collected by continuous sampling from environment, I shuffle and break the data into mini-bathes with size 64 in each epoch.

**Action Noise** To explore the state space and increase the robustness of the system, Gaussian noise is introduced to action space. This high dimensional Gaussian distribution is captured by a mean action vector and a standard devia-

| | |
|---|---|
| Batch size per step | 256 |
| Clip param ($\epsilon$) | 0.2 |
| Optimizer epochs per iteration | 10 |
| Optimizer step size | $1e-4$ |
| Optimizer batch size | 64 |
| Discount ($\gamma$) | 0.995 |
| GAE ($\lambda$) | 0.97 |

*Table 2.* Base Hyperparameters

| Layer | Input | Activation fun. | Output |
|---|---|---|---|
| Policy network structure | | | |
| fc1 | 1 | ReLU | 5 |
| fc2 | 5 | ReLU | 2 |
| Value network structure | | | |
| fc1 | 1 | ReLU | 5 |
| fc2 | 5 | ReLU | 1 |

*Table 3.* Network Structures

tion vector, which are both generated by the trained policy network.

**Normalize Reward** For stability purpose, I normalize the discounted rewards (7). During sampling, the trajectory value might across magnitude due and some noise. It would sometimes lead network weights to extreme values in backpropagation. By normalize reward, I could still capture the action feature and keep values in a specific convenient range. Thus, it results in faster learning.

### 4.3. Results

The curve of average evaluation rewards and iteration number is shown in *Fig.7*. The reward increases steadily at the beginning, then stabilizes after 600 iterations with a reward around 600. The fluctuation of the reward is probably caused by the large learning rate, which will be decreased in the future. The policy generated with a reward of 600 is capable to guide the agent pick up the cuboid (*Fig.*8), but sometimes the cuboid falls. I tried several tuning methods expecting to have better performance but haven't found a better set of hyperparameters yet due to the limited computational power and time.

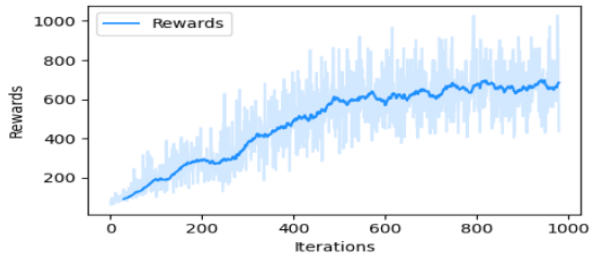

*Figure 7.* Reward Curve

*Fig.* 8 shows that the robot successfully picks up the cuboid using the trained policy network and the change of success rates during the training process. The success rate increases during training and eventually reaches 0.7, which acceptable but there is still much room for improvement.

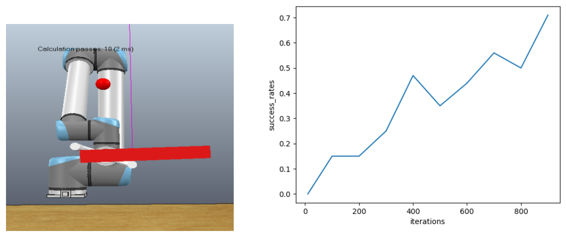

*Figure 8.* A Successful Pick and the Success Rate in Training

## 5. Analysis

Surprisingly, the agent picks up the cuboid almost merely by rotating at the initial position under the trained policy, instead of following the parabola path designed in section 2.3. This is probably caused by the high friction coefficient ($\mu = 0.7$) because it won't be necessary for the end effctor to move far along the parabola to prevent the object from slipping down with a high friction coefficient. I will try to decrease $\mu$ in the following experiments and give further analysis according to the results.

## 6. Conclusion and Future work

I successfully trained a the UR-10 robot to pick up the cuboid with PPO. This project shows the promise of PPO's feasibility and flexibility on difficult task with continuous state and action space.
As for possible future work, I plan to further tune the hyperparameters and the network structures to accelerate training and achieve better models.
Also, as I mentioned before, this is an over-simplified model with a settled path, a high friction coefficient, and simple network structures. In the future, we will generate different path in each iteration and make the robot to learn how to generate trajectories accordingly.

## References

[1] Yuxi Li: Deep Reinforcement Learning: An Overview. CoRR abs/1701.07274 (2017)

[2] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, David Silver: Emergence of Locomotion Behaviours in Rich Environments. CoRR abs/1707.02286 (2017)

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov: Proximal Policy Optimization Algorithms. CoRR abs/1707.06347 (2017)

[4] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley,

David Silver, Koray Kavukcuoglu: Asynchronous Methods for Deep Reinforcement Learning. ICML 2016: 1928-1937

[5] Ronald J. Williams: Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Mach. Learn. 8: 229-256 (1992)

[6] Henderson, Peter, et al. ”Deep reinforcement learning that matters.” Thirty-Second AAAI Conference on Artificial Intelligence. 2018.

[7] Hado van Hasselt, Arthur Guez, Matteo Hessel, David Silver: Learning functions across many orders of magnitudes. CoRR abs/1602.07714 (2016)