Master Thesis

Huang ShengKai

Contents

1 Introduction

An intelligent robot is expected to not only be able to perceive the environment, but also interact with the environment.

Humans have a fantastic ability that is manipulate objects with the advanced dexterity capabilities of our arm and hands. Among all these abilities, object grasping is fundamental and significant in that it will bring enormous productivity to the society. For example, an industrial robot can accomplish the pick-and-bin task which is laborious for human labors, and a domestic robot is able to provide assistance to disabled people in their daily grasping tasks.

As much as being highly significant, robotic grasping has long been researched. This includes conventional approachs, recent End-to-End learning-based approachs, and the hybrid approachs, which learning-based approach replace some parts of the conventional approach.

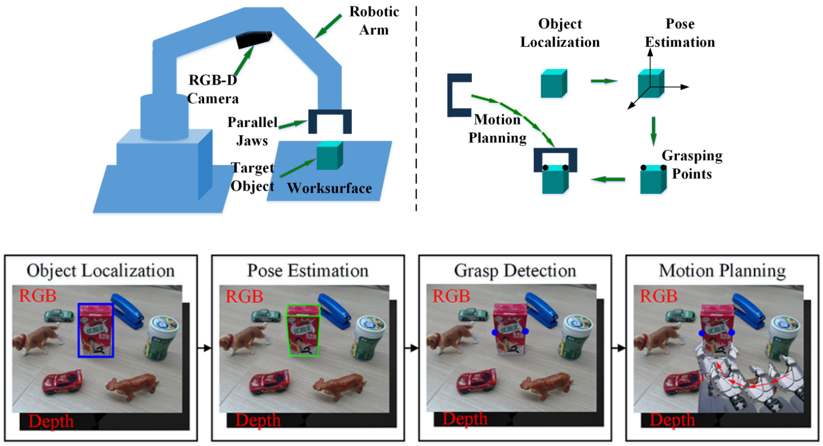


Figure 1: Conventional Approach of Robotic Grasping System

The conventional approach of robotic grasping system is considered as being composed of the following sub-systems[5]: the grasp detection system, the grasp planning system and the control system. As Fig.1 illustrate, grasp detection system is divided into three tasks: target object localization, pose estimation and grasp point detection. Each system is programmed separately, solved by different approachs.

Instead of using this pipeline of steps, one of the recent breakthroughs in robotics is leverage End-to-End learning-based approach, expecially Reinforcement Learning, where robotic could can reason a appropriate action directly from raw sensory observations, such as camera images. Typically as Fig.2 Sergey Levine lets a robot successfully complete some complex tasks, by single system, which the input is an raw image, the output is the robot's joint torque.

The most interesting thing is that this technique would allow the next gener-

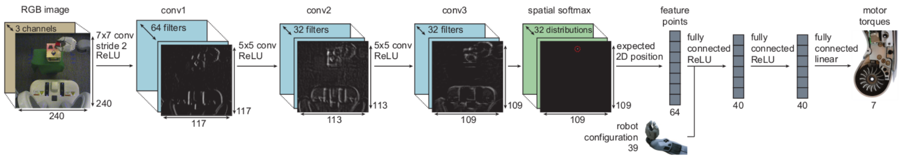


Figure 2: End-to-End learning-based approachs

ation of robots learn to accomplish new tasks directly, without hand-engineering steps to perform each sub-systems. In other words, the robot would accept the raw data from sensors, such as RGB camera images and directly infer the low level action(e.g. motor velocities) by a task agnostic learning algorithm.

Compire with End-to-End learning-based approach conventional approach also have many issues, which is easy for End-to-End approach.

- It is kind of error-prone, because the errors propagate from one step to another (e.g. inadequate grasp point results in poor or unsuccessful grasp).
- It relaies on cumbersome and strict sensor calibration(e.g. camera calibration, point cloud registration). And sensor can not be moved after calibration, which makes it difficult to adapt to complex environments.
- It requires different pipelines to deal with different tasks under different situations.

1.1 Previous Research

Reinforcement learning (RL) has recently achieved widespread success in sequential decision-making problems in robot.

In some works, the input is raw data(images, joint-angle, etc.), and the output is a low-level, high-dimensional continuous action(joint-angle, joint-velocity, offset of end-effector, etc.).

Deirdre Quillen et al.[13] evaluate the performance of Vision-Based Robotic Grasping benchmark with many off-policy RL approachs. Jan Matas et al.[8] combine many state-of-the-art off-policy RL algorithms to solve the problem of deformable objects manipulation. Mel Vecerik et al.[19] leverage demonstrations with Deep Deterministic Policy Gradient (DDPG) algorithm, and realized deformable-clip insertion task in real robot setting.

Meanwhile, in some work, the output is a high-level, low-dimensional discrete action macro(push, grasp, retract etc.). such as Ameya Pore et al.[12] learn the pick and place task by behaviour-based reinforcement learning approach.

1.2 Background

Reinforcement Learning

One specific branch of machine learning applicable to the task of robotic manipulation is Deep Reinforcement Learning. RL, in general, builds on behavioural psychology and investigates how to teach an agent to execute actions that lead to the highest cumulative reward. In practice, this would mean that the engineer would only need to define what is the goal of the robot and the robot would learn how to achieve it by itself. This is a massive decrease in the workload compared to a conventional approach where the engineers need to design and implement each subtask necessary to achieve the goal.

Moreover, a robot learning is adopt in achieve the task which is likely error-prone, because robot can learn how to recover from them. In contrast, the conventional approach need to prepare hard-code each possible mistake in advance. Otherwise, it would fail to accomplish the goal.

Then, we briefly cover the basic concepts of RL in a limited space.

Optimize Objective of RL The classic RL setting that can be represented as a Markov Decision Process (MDP) defined as a 6-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma, \mathcal{O})$, where $\mathcal{S}$ is the set of full states of the environment, $\mathcal{A}$ is the set of actions, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, γ is a discount factor, and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a deterministic state transition function. The decision process is partially observable and the agent receives observations o from the set of observations $\mathcal{O}$, followed by a sample process $o \sim \mathcal{G}(s)$.

The goal of optimization is to find a policy π, which maps states to actions to maximize accumlated discounted reward η_π.

$$\eta_\pi = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] \tag{1}$$

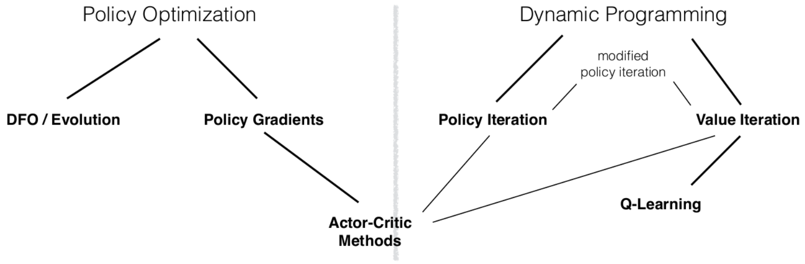


Figure 3: Taxonomy of model-free RL algorithms

Model-Free RL Figure 3 shows a taxonomy of model-free RL algorithms (algorithms that do are not based on a dynamics model). At the top level, we have two different approaches for deriving RL algorithms: policy optimization and dynamic programming.

Two difficulties in Robot Learning

1. Sparse Reward and Reward Engineering

   A sparse reward signal is series of rewards produced by the agent interacting with the environment, where most of the rewards received are non-positive. This makes it extremely difficult for RL algorithms to connect a long series of actions to a distant future reward.

   Thus, Designing a reward function that not only reflects the task goal but is also carefully shaped to avoid sparse reward is still a common challenge. For example, Popov et al.[11] used a reward function that consists of five complicated terms carefully weighed to get a stacking policy. It limits the applicability of RL because it takes much time to shape an appropriate reward function. On the other hand, oversimplified reward functions such as a binary signal could cause serious sparse reward issues. Although there are some data augmentation methods such as Hindsight Experience Replay[2] that could solve the partial sparse reward issue, it, however, is not easy to apply in the high-dimension state, such as images.

2. Demonstration Dependency

   The other tricky issue is that current image-policy-based robot learning greatly depends on demonstrations, which is a frequently-used method of Imitation Learning (IL). Demonstrations that are generated by a human to initialize policy[8, 19] or make a replica of the demonstration policy such as behavioral cloning (BC).

   BC or IL approaches, however, are limited because they do not consider the task or environment. Furthermore, human demonstrations can be suboptimal because humans also make mistakes or do some redundant actions. Moreover, agents will get into trouble when they encounter states that do not belong to the demonstration set.

1.3 Objectives

In this thesis, addressing the two issues pointed out above, under the goal-orientation sequential decision making RL scenario, we discusses an method, by which shaping a reward function much more efficiently.

The technique we proposed called Rank Temporal-Difference (Rank-TD), which allows agents to explore by following the expert-roughly-designed state trajectory. Furthermore, it can combine with any model-free RL algorithm.

We build its mathematical model and prove the correctness of the Rank-TD based on policy invariance theory[9]. Then, We discuss some sensable and useful property by analyzing a toy case. Finally, we train an agent with an image-based policy on pick-and-place task. We employed domain randomization in simulation to guarantee the potential of policy transfer from simulation to the real world without further training.

1.4 Thesis Layout

The core part of this Thesis starts with Charpter 2, where Chapter 2 we introduce Proximal Policy Optimisation (PPO) briefly, which adopt in final pick and place task. In Chapter 3, introduces the derivation process of the method and design a series of toy experiment to verify method. Then Chapter 4 is our main experiment, in which we train and evaluate a End-to-End image-based picking policy. Finally, in Chapter 5, we summarise the work we have done to achieve our results, and we follow up with a couple of suggestions for future work.

2 PPO and Parallelize Rollout

2.1 Proximal Policy Optimisation

In this section, we describe the theoretical basis of Proximal Policy Optimisation (PPO)[17] algorithm, which is adopted in our Pick and Place task.

PPO[17] follows the actor–critic architecture and it is based on the trust-region policy optimisation(TRPO)[15] algorithm. This algorithm aims to learn a stochastic policy $\pi_\theta\left(a_t \mid s_t\right)$. If action space is continue, $\pi_\theta\left(a_t \mid s_t\right)$ maps states with Diagonal Gaussian Distribution over actions, and if action space is discrete, $\pi_\theta\left(a_t \mid s_t\right)$ maps states with Categorical Probability Distribution over actions. In addition, the critic is a value function $V_\omega\left(s_t\right)$ that outputs the mean expected reward in state s_t. This algorithm has the benefits of TRPO and in general of trust region based methods[15] but it is much simpler to implement it. The intuition behind trust-region based algorithms is that, at each parameter update of the policy, the output distribution cannot diverge too much from the original distribution.

Let $r_t\left(\theta\right)$ denote the probability ratio defined in Equation (2), so that $r\left(\theta_{old}\right) = 1$.

$$r_t\left(\theta\right) = \frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{old}}\left(a_t \mid s_t\right)} \tag{2}$$

θ_{old} is the actor's parameter vector before the update. The objective of TRPO is to maximise the objective function $L\left(\theta\right)$ defined in Equation (3). Here, $\mathbb{E}\left[...\right]$ indicates the average over a finite batch of samples. The advantage $\hat{A}_t$ is a kind of policy gradient estimator, which called $\mathrm{GAE}-\lambda$[16], was popularised by Schulman et al. and indicates how good the performed action is with respect to the average actions performed in each state. To compute the advantages, the algorithm executes a trajectory of T actions and computes them as defined in Equation (4). $\mathrm{GAE}-\lambda$ has minimum bias and variance in current policy gradient estimator[16]. Here, t denotes the time index $[0, T]$ in the trajectory of length T and γ is the discount factor.

$$L\left(\theta\right) = \mathbb{E}\left[r_t\left(\theta\right)\hat{A}_t\right] \tag{3}$$

$$\hat{A}_t = -V_\omega\left(s_t\right) + r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V_\omega\left(s_T\right) \tag{4}$$

At each policy update, if the advantage has a positive value, the policy gradient is pushed in that direction because it means that the action performed is better than the average. Otherwise, the gradient is pushed in the opposite direction.

Without any constraint, the maximisation of the objective function $L\left(\theta\right)$ would lead to big changes in the policy at each training step. PPO modifies the objective function so that penalises big changes in the policy that move $r_t\left(\theta\right)$ away from 1 in each training step.

There are two reasons of why must we maintain $r_t\left(\theta\right)$ near to 1.

1. Objective function $L(\theta)$, which also be called surrogate objective, is the First-Order Approximation of accumlated discounted reward η_π (Equation (1)). That is rough approximation, only if the $\Delta\theta$ small enough, improving $L(\theta)$ will also improve η_π.

$$\nabla_\theta L_{\theta_0}(\theta) \mid_{\theta=\theta_0} = \nabla_\theta \eta_{\theta_0}(\theta) \mid_{\theta=\theta_0}$$

2. According to J.Schulman's Phd thesis[14], trust-region based policy optimization is proposed based on Kakade and Langford's work[7], which call conservative policy iteration, the new policy was defined to be the following mixture:

   $$\pi_{new}(a \mid s) = (1-\alpha)\,\pi_{old}(a \mid s) + \alpha\pi'(a \mid s) \tag{5}$$

   where $\pi' = \arg\max_{\pi'} L_{\pi_{old}}(\pi')$, with a critial Lower Bound is

   $$\eta(\pi_{new}) \geqslant L_{\pi_{old}}(\pi_{new}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2 \tag{6}$$

   This Lower Bound tall us that if $\alpha \ll 1$, the policy update would have a monotonic improvement guarantee. It is a really attractive conclusion. However the mixture policy are rarely used in practice.

   J.Schulman extends this imporvement guarantee to practical problems. The first principal modification is to replace α in Equation (5) with a measure between π_{old} and π_{new}, that is $\mathrm{D}_{TV}^{max}(\pi_{old}, \pi_{new})$. where

   $$\mathrm{D}_{TV}^{max}(\pi, \tilde{\pi}) = \max_s \mathrm{D}_{TV}(\pi(\cdot \mid s) \parallel \tilde{\pi}(\cdot \mid s)) \tag{7}$$

   Same as Kakade's Lower Bound, Equation (6), J.Schulman also deduces a Lower Bound that

   $$\eta(\pi_{new}) \geqslant L_{\pi_{old}}(\pi_{new}) - \mathrm{C}\mathrm{D}_{KL}^{max}(\pi_{old}, \pi_{new}) \tag{8}$$

   where $\mathrm{C} = \frac{2\epsilon\gamma}{(1-\gamma)^2}$, which γ is discount rate and usually set as 0.99 or 0.98. Thus, $\frac{\gamma}{(1-\gamma)^2}$ is a big number, it measures the horizon of agent. The farther the agent could see, the greater this value will be. Only if the $\mathrm{D}_{KL}^{max}(\pi_{old}, \pi_{new})$ small enough, improving $L(\theta)$ could improve η_π.

   That is why in trust-region based algorithms, the new policy distribution cannot diverge too much from the original policy distribution during each parameter update of the policy.

The objective function of PPO algorithm clip version is Equation (9), which is a trick version of Equation (3) by laveraging clip function. It greatly simplifies the implementation of TRPO, without performance lost.

$$\mathrm{L}^{CLIP}\left(\theta\right) = \mathbb{E}\left[min\left(r_t\left(\theta\right)\hat{A}, clip\left(r_t\left(\theta\right), 1-\delta, 1+\delta\right)\hat{A}\right)\right] \tag{9}$$

where the δ is a hyper-parameter that changes the clip range.

To update the value function $V_\omega\left(s\right)$(the critic), the squared-error loss function is used (Equation (10))

$$J\left(\omega\right) = \left(r_t + \gamma r_{t+1} + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V_\omega\left(s_T\right) - V_\omega\left(s_t\right)\right)^2 \tag{10}$$

Algorithm 1 Proximal Policy Optimisation (PPO) with single worker.

```
for each $e \in episodes$ do
    Run a trajectory $\tau$, which length is T time-steps, following $\pi_{old}$;
    Compute advantage estimates $\hat{A}_1 ... \hat{A}_T$;
    Optimise critic's loss function $J$ w.r.t. $\omega$;
    Optimise actor's loss function $\mathrm{L}^{CLIP}$ w.r.t. $\theta$;
end for
```

2.2 Parallelize Rollout

A rollout is a simulation of a policy in an environment. It alternates between choosing actions based (using some policy) and taking those actions in the environment. This part will introduce the parallelization techniques, which used currently, and focus on the method used in our research.

PPO is a on-policy RL Algorithm, that the data sampled from environment must be generated from current policy. it can't learn from too old or external sample data. That means on-policy RL Algorithm has the worst sample-efficiency.

A efficient, scalable distributed RL training framework(architecture) has become a kind of necessity of RL industry application. By analyzing the single worker version PPO above (Algorithm 1). we could see that if we just roll-out one trajectory τ, the variance of statistic variable $A\left(a, s\right)$ would extremly huge. Although GAE $-\ \lambda$ help us soften the serious variance of the tail of the trajectory, it is never enough for algorithm to update policy just use one trajectory τ. Typically, a large number of workers are responsible for exploring the environment in the real practical application with PPO based algorithms. For example OpenAI trains a dexterous robot hand to solve Rubik's Cube with 29,440 CPU cores(worker process)[1], and trains OpenAI Five, which has started to defeat amateur human teams at Dota 2 game[3] with 57,600 workers, which shown as Fig.4.

RL is inherently comprised of heterogeneous tasks: Running Environments, Model Inference, Model Training. Three parts depend on each other, usually locate in different hardware devices(CPU, GPU or TPU).

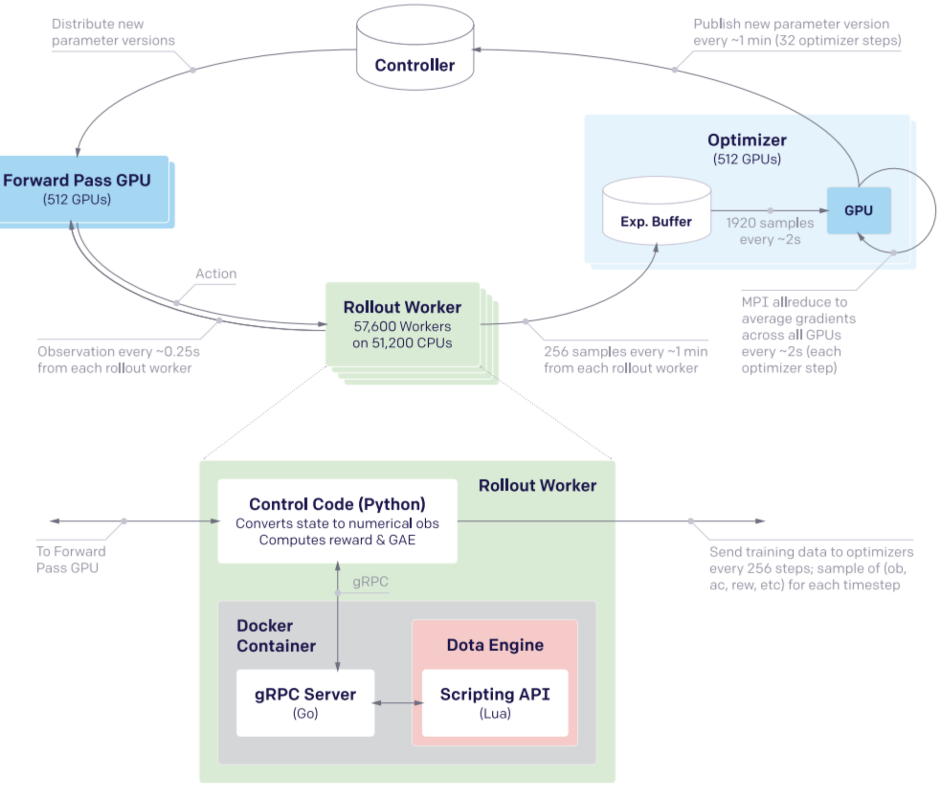

Distribute new parameter versions
Controller
Publish new parameter version every ~1 min (32 optimizer steps)
Optimizer
(512 GPUs)
Forward Pass GPU
(512 GPUs)
Exp. Buffer
1920 samples every ~2s
GPU
Action
Rollout Worker
57,600 Workers on 51,200 CPUs
MPI allreduce to average gradients across all GPUs every ~2s (each optimizer step)
Observation every ~0.25s from each rollout worker
256 samples every ~1 min from each rollout worker
Rollout Worker
Control Code (Python)
Converts state to numerical obs
Computes reward & GAE
To Forward Pass GPU
Send training data to optimizers every 256 steps; sample of (ob, ac, rew, etc) for each timestep
gRPC
Docker Container
Dota Engine
gRPC Server
(Go)
Scripting API
(Lua)

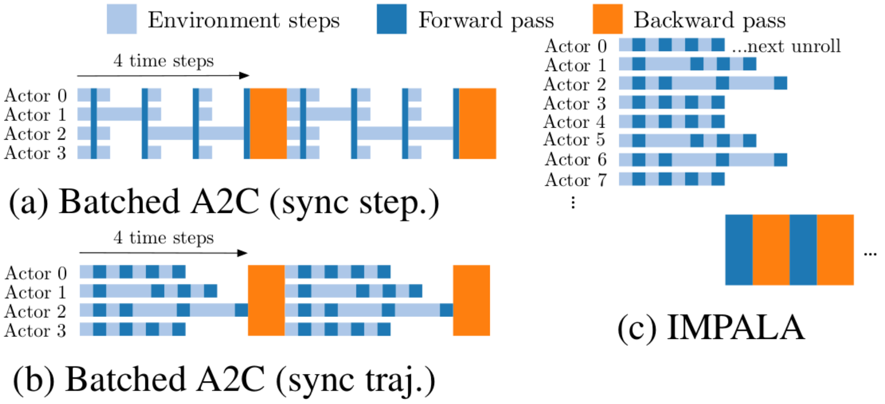
Figure 4: System Overview of OpenAI Five

Running Environments The environment process(usually simulator) accepts an action a_t, then simulate out the next state s_{t+1} and reward r_{t+1}, based on environment transform $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ and reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. This part runs on the CPU.

Model Inference Sample an action from the current policy π, which $a_{t+1} \sim \pi(\cdot \mid s_t)$. This part is the forward calculation of a NN model, which usually run on GPU, if the model is a large-scale neural network. As we all know, the overhead of data transform between Host(RAM) and Device(GPU) is expensive. In order to make CPU and GPU more effecient, we usually collect a batch of state then infer a batch of action at once, as Fig.5(a). Thus we have to synchronous every actors to get a big batch of state. The disadvantage is obvious the slowest worker would slow everyone down.

Model Training Training means forward calculate surrogate objective loss, and get the gardent w.r.t. policy network weight θ. Then update the weight by using Stochastic gradient descent(SGD). This part is usually run on GPU.

Fig.5 illustrates three kinds of calculate pattern. (a) and (b) are different in synchronous timing, (a) is synchronous every worker each step, and (b) is synchronous every worker after each trajectory. The advantage of pattern (a) is it has better use of bandwidth, howerer, the slowest worker would slow everyone down. The ideal computing pattern is (c), which is provided by a framework call IMPALA[6], both the CPU and GPU are fully utilized.



(a) Batched A2C (sync step.)

(b) Batched A2C (sync traj.)

(c) IMPALA

Figure 5: The parallelization solution for single machine single GPU

In Pick and Place task, we adope calculate pattern (a), which synchronous every worker after each step. we use MPI to generate subprocess and control subprocess, with the primitives such as `fork, send, recv,` etc.

Naturally, We have the multi-worker version of PPO (Algorithm 2).

Algorithm 2 Proximal Policy Optimisation (PPO) with Multi-Rollout worker.

```
for each $e \in episodes$ do
    // Asynchronous for each worker
    for each $w \in workers$ do
        Run a trajectory $\tau_w$, which length is T time-steps, following $\pi_{old}$;
        Compute advantage estimates $\hat{A}_1 ... \hat{A}_T$;
    end for
    Optimise critic's loss function $J$ w.r.t. $\omega$;
    Optimise actor's loss function $L^{CLIP}$ w.r.t. $\theta$;
end for
```

2.3 Summary

In this chapter, we introduce Proximal Policy Optimisation (PPO) briefly. Then we analyze why policy should not be change too large during each policy improvement phase. Finally, we introduce the importance of parallelization technology in RL, and we introduce the parallelization technology what we adopted.

3 Rank Temporal Difference

3.1 Policy Invariance and Potential-Based Reward Shaping

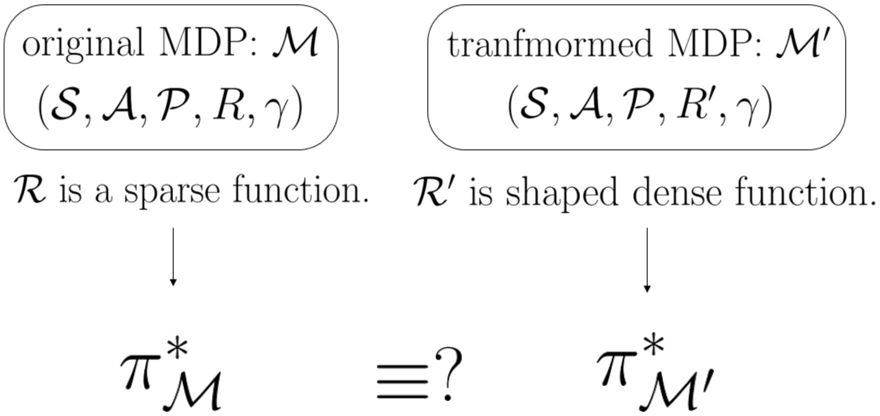

original MDP: $\mathcal{M}$
$(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$
tranfmormed MDP: $\mathcal{M}'$
$(\mathcal{S}, \mathcal{A}, \mathcal{P}, R', \gamma)$
$\mathcal{R}$ is a sparse function.
$\mathcal{R}'$ is shaped dense function.
$\pi^*_{\mathcal{M}}$
$\equiv?$
$\pi^*_{\mathcal{M}'}$

Figure 6: Whether the reward shaping has the Policy Invariance guarantee?

Andrew Y. Ng[9] proposed a formal of reward shaping:

For a MDP $M\left(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma\right)$, exists a transformed MDP $M'\left(\mathcal{S}, \mathcal{A}, \mathcal{P}, R', \gamma\right)$, where $R^{'} = R + F$. And $F : S \times A \times S \mapsto \mathbb{R}$ is a bounded real-valued function called the shaping reward function.

As Fig.6 shows that, the core point is the reward function R in $\mathcal{M}$ is too sparse for agent to find a optimal policy π^*, but it is easy in $\mathcal{M}'$ with a shaped dense reward function R'. If there is a consistency guarantee between $\pi^*_{\mathcal{M}}$ and $\pi^*_{\mathcal{M}'}$, which calls Policy Invariance for shaping reward function F. That means we could find $\pi^*_{\mathcal{M}}$ by solving $\pi^*_{\mathcal{M}'}$.

Fortunately, Andrew Y. Ng provides a important Theorem with a serise clear proofs.

If there exists a real-valued function $\Phi : S \mapsto \mathbb{R}$, that for all $s \in S - \{s_{goal}\}$, $a \in A$, $s' \in S$,

$$F\left(s; a; s'\right) = \gamma\Phi\left(s'\right) - \Phi\left(s\right) \tag{11}$$

Then, F is a potential-based shaping function, which is the **necessary and sufficient condition** of policy invariance.

In applications, Φ should of course be chosen using expert knowledge about the domain. According to chapter 5 of Andrew Y. Ng's paper[9], it is possible that for certain problems, it may be easier for an expert to propose a potential Φ for an "undiscounted" shaping function $\Phi\left(s'\right) - \Phi\left(s\right)$, even when $\gamma \neq 1$.

Policy Invariance Theorem provides a solid theoretical foundation for the development of Rank-TD method.

3.2 Mathematical Model of Rank-TD

Following the policy invariance theorem, under the goal-orientation sequential decision making RL scenario, we assume MDP $\mathcal{M}$ has a sparse reward function $R = \{0, 0^+\}$, which 0^+ is a very small positive number, only get if agent achieve the goal.

Then we build a transformed MDP $\mathcal{M}'$, which reward function $R' = R + F$ with a potential-based shaping F. Because 0^+ is a very small positive number, for simplicity, we ignore R, then $R' = F$ (the explain here is that the elimination of R is not allowed in theory, but it does not affect the numerical results.) $F : \texttt{rank}\left(s_{t+1}\right) - \texttt{rank}\left(s_t\right)$ has a formal as equation(11) . Here the $\texttt{rank}$ inherits the nature of the potential Φ, but it develop and extend with other limitation and practical sense.

The optimal policy π^* of $\mathcal{M}'$ is consistent with $\mathcal{M}$, which is:

$$\pi^* \doteq \arg\max_{\pi} \texttt{Pr}_{\tau\sim\pi}\left(s_T = \texttt{goal}\right) \tag{12}$$

where trajectory $\tau\left(s_0, o_0, a_0, s_1, o_1, a_1 \cdots s_t, o_t, a_t \cdots s_T\right)$ is generated under the policy $\pi\left(a_t \mid o_t\right)$.

Rank function is defined as a mapping $\texttt{rank} : \mathcal{S} \to \mathbb{N}$, satisfied that if trajectory τ is generated under the optimal policy π^*, then

$$\texttt{rank}\left(s_0\right) \leqslant \texttt{rank}\left(s_1\right) \leqslant \cdots \leqslant \texttt{rank}\left(s_t\right) \leqslant \cdots \leqslant \texttt{rank}\left(s_T\right)$$

The difference between $\texttt{rank}$ and potential Φ is that, We don't allow too many **inverse rank transform** during agent interact with the environment.

Finally, the reward R' defined as

$$r\left(s_t, s_{t+1}\right) \doteq \begin{cases} -1, & s_{t+1} = \text{terminal condition} \\ \texttt{rank}\left(s_{t+1}\right) - \texttt{rank}\left(s_t\right), & \text{other} \end{cases}$$

The objective of optimization is to find an a optimal policy $\pi^*\left(a_t \mid o_t\right)$.

$$\begin{aligned} \max \quad & \texttt{Pr}_{\tau\sim\pi}\left(s_T = \texttt{goal}\right) \\ s.t. \quad & \text{cout}(\texttt{rank}\left(s_t\right) > \texttt{rank}\left(s_{t+1}\right)) \leqslant \varepsilon, \quad \varepsilon \in \mathbb{N} \ . \end{aligned}$$

ε is a hyper-parameter that restricts the times of inverse rank transform during roll-out. When an episode arises more than ε inverse rank transforms, the episode would be terminated.

3.3 Summary

In this chapter we introduce the Theorem of Policy Invariance, and we expand RankTD from potential-based reward shaping. We assume that RankTD has faster policy convergence.

4 Experiment: Toy Tasks

4.1 Toy Task Description

We design a toy task (Fig.7) to verify the idea of Rank TD.

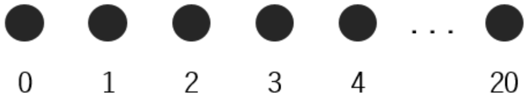


Figure 7: Toy case: agent exists in a world with 21 discrete states spaces. The agent can choose one from action space $\mathcal{A} = \{\Leftarrow\Leftarrow, \Leftarrow, \bigcirc, \Rightarrow, \Rightarrow\Rightarrow\}$. The goal is to arrive state 20.

Assuming an agent exists in a world with state space $\mathcal{S} = \{0, 1, ..., 20\}$ and action space $\mathcal{A} = \{\Leftarrow\Leftarrow, \Leftarrow, \bigcirc, \Rightarrow, \Rightarrow\Rightarrow\}$ which means `two steps backward, one step backward, stand still, one step forward, two steps forward.` The agent initialized in 0 state each time, which $p\left(s_t = 0 | t = 0\right) = 1$. The deterministic environment dynamic transform $s_{t+1} \leftarrow \mathcal{P}\left(s_t, a_t\right)$ defined as Table 1. Red "-1" means that the agent falls into a trap and terminates the episode. Green "20" is the goal state for the agent. In this toy case, observation o_t is equivalent to the internal full state s_t, which $o_t = s_t$.

Table 1: Dynamic Transform Table

s_{t+1} a_t s_t	$\Leftarrow\Leftarrow$	$\Leftarrow$	$\bigcirc$	$\Rightarrow$	$\Rightarrow\Rightarrow$
0	0	0	0	1	2
1	0	0	1	2	3
2	0	1	2	3	4
3	-1	-1	-1	4	5
4	2	3	4	5	6
5	3	4	5	6	-1
6	4	5	6	7	8
7	5	6	7	8	9
8	6	7	-1	-1	-1
9	7	8	9	10	11
10	8	9	-1	-1	-1
11	9	10	11	12	13
12	10	11	12	13	14
13	11	12	-1	-1	-1
14	12	13	14	15	16
15	13	14	15	16	17
16	-1	-1	-1	17	18
17	15	16	17	18	19
18	16	-1	-1	-1	-1
19	17	18	19	20	20

4.2 Reward Setting

We prepared three kinds of reward functions. The first is that the agent gets a bonus of 1 only when it reaches the target state, or a penalty -1 when it falls into the trap, and 0 at other times (Equation 13).

$$r_{\mathbf{sparse}}\left(s_t, s_{t+1}\right) \doteq \begin{cases} -1, & \boldsymbol{s_{t+1}=-1} \\ 0, & \textbf{other} \\ 1, & \boldsymbol{s_{t+1}=20} \end{cases} \tag{13}$$

The second reward is set as the temporal difference between the previous state and current state, and -1 when it falls into the trap (Equation 14).

$$r_{\mathbf{stateTD}}\left(s_t, s_{t+1}\right) \doteq \begin{cases} -1, & \boldsymbol{s_{t+1}=-1} \\ s_{t+1}-s_t, & \textbf{other} \end{cases} \tag{14}$$

We try to find a policy $\pi\left(a_t \mid o_t\right)$ by the Q-learning Algorithm under two reward settings with same hyper-parameter, architecture, and random seed.

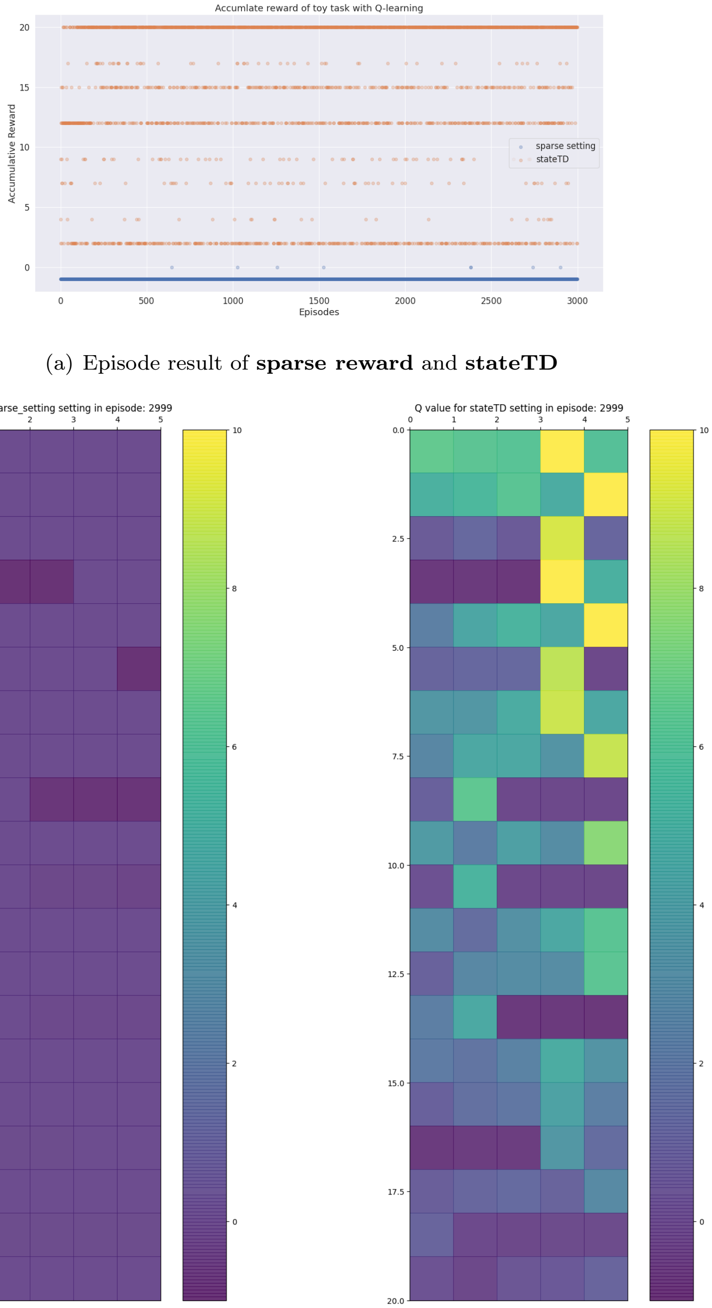
The result (Fig.8) shows that if the positive feedback signal is too sparse, the agent would learn nothing. On the contrary, the immediate feedback signal (state temporal difference) setting, which could reflect the change of completing the task, allows learning to be more smoothly.

The 3th reward is set similar as the above setting(Equation 14), but we add a terminal condition: inverse rank transform more than twice (Equation 15).

We can imagine that the rank function is a sequence of waypoints, which also represents the completeness of a task, as defined by the expert according to the expert's understanding of the task.

$$r_{\mathbf{RankTD}}\left(s_t, s_{t+1}\right) \doteq \begin{cases} -1, & \boldsymbol{s_{t+1}=-1} \text{ and inverse rank transform } > 2 \\ s_{t+1}-s_t, & \textbf{other} \end{cases} \tag{15}$$

The result (Fig.9) shows that RankTD reward setting can reduce the scale of explorations and accelerate the convergence of policy.



(a) Episode result of **sparse reward** and **stateTD**

(b) Q value of sparse reward setting

(c) Q value of **stateTD**

Figure 8: Q learning cannot solve this task in a sparse reward setting; however, gets great results in the reward setting, which is based on state temporal differences.

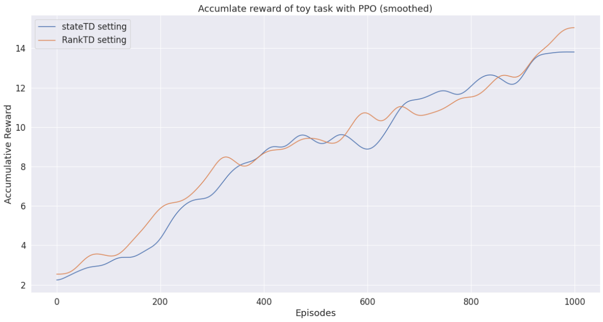


Figure 9: Toy task with statdTD and RandTD reward setting solved by PPO. RandTD reward setting has a faster convergence.

4.3 Bottleneck Issue

Secondly, we modify the toy case, get a useful conclusion.

If the state transitions between the adjacent ranks are not too complicated, it would difficult for agent to explore the path from one rank to a higher rank, in other words, when the path becomes narrower, learning will be blocked. We call it *bottleneck issue.* For example, if we change rows 2, 3, and 4 of the original state transform table, such as Table 2.

Table 2: Narrower Version Dynamic Transform

s_{t+1} a_t s_t	⇐⇐	⇐	○	⇒	⇒⇒
2	0	1	2	3	-1
3	-1	-1	-1	4	-1
4	-1	-1	-1	5	-1

The result (Fig.10) shows that as the process of state transform becomes more complex and the ascending path becomes narrower, the difficulty of agent exploration increases.

4.4 Summary

In this chapter we design a series toy tasks to verify the idea of RankTD. From the comparison between **sparse setting** and **stateTD** by Q-learining, we can see that **stateTD** is much easier to learn the optimal strategy. From the comparison between **stateTD** and **RankTD** by PPO, we can see that **RankTD** get faster policy convergence. Then we disscuss the *bottleneck issue*, when state transform becomes more complex, or rank function is too rough, agent is difficult to find the optimal policy quickly.

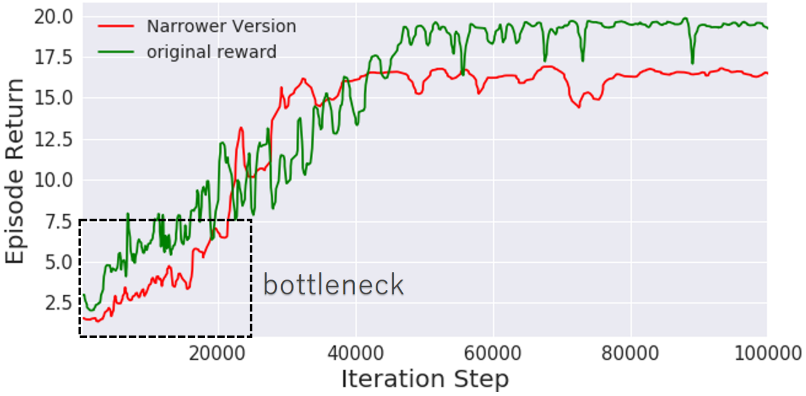


Figure 10: More complex state transform of the task makes it more difficult for the agent to explore. When applied in complex problems, if we fail to grasp the scale or granularity of the rank function, agents are unable to learn the good policy.

5 Experiment: Pick and Place Task

In order to extend the theory to practical application. We design a **Pick and Place** robot task, where the robot-arm grasps an object and puts it in a basket.

5.1 Environment

The robot environment, which developed and published by Google Brain[13] is built on the PyBullet simulator. PyBullet[4] is a physical simulator developed for games, visual effects, robotics and reinforcement learning.

There are many diverse and highly varied situations in robotic grasping. One of the most important challenges in learning-based grasping is generalization: can the system learn grasping patterns and cues that allow it to succeed at grasping new objects that were not seen during training? We prepared 90 various objects for the training phase, and 10 objects for the testing phase. We want to know whether agents can generalize various grasping strategies for different kinds of objects.

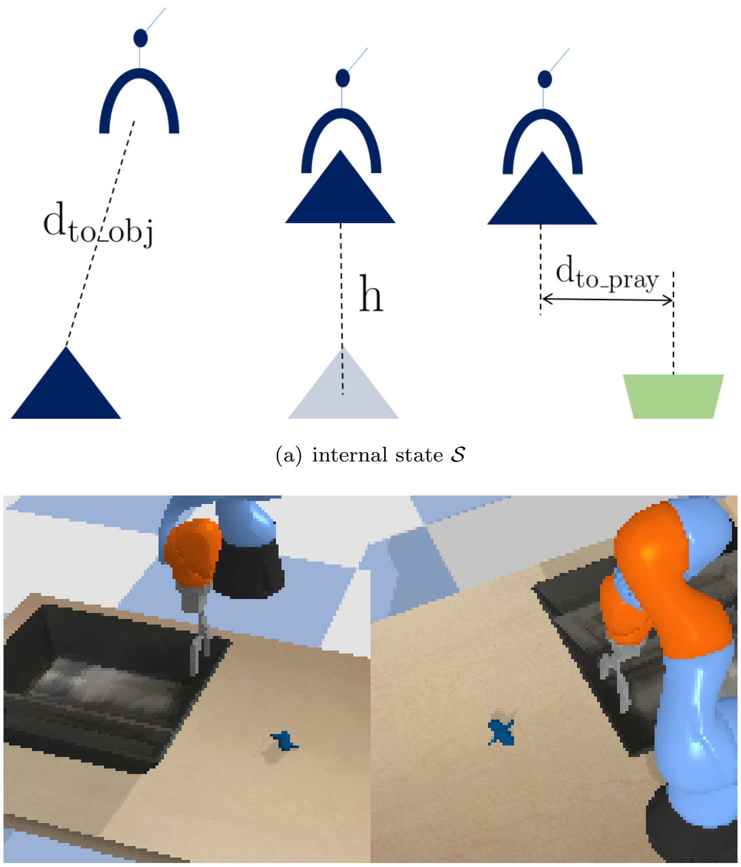
5.2 Task Description

Action $\mathcal{A}$: The robot-arm is 7DOF kuka-iiwa because this task does not need to change the orientation of the end-effector, thus the orientation is fixed vertically. The robot is controlled by 5-dimensional action. The first three dimensions are the relative change of cartesian coordinates$(\Delta x, \Delta y, \Delta z)$ of the end-effector, while the fourth dimension is the relative change of gripper's rotation angle$(\Delta\theta)$, and the last dimension is the instruction of the gripper open and closed (negative for closing and positive for opening).

Internal Full State $\mathcal{S}$: The Internal full states of the simulator as in Fig.11(a), include 1) the distance between the gripper and object $\mathrm{d}_{\mathrm{to\_obj}}$; 2) the height of the object to be lifted h; 3) the distance between the gripper and pray $\mathrm{d}_{\mathrm{to\_pray}}$; 4) gripper-pray and gripper-object collision detection.

Observation $\mathcal{O}$: The agent gets RGB observation with dimensions $256 \times 128 \times 3$ as in Fig.13(b), captured by two cameras that the one is located at the top-left of the robot, and the other one is located at the shoulder of robot. The image obtained at this location contains the relative positions of the pray, object, and gripper. Furthermore, in order to train in headless mechine, we don't allow the GPU to render the shadow, which could cause the judgment error of the relative position from different angle of view. To increase the robustness of the model, during the training phase, the position of the camera and the robot's base coordinate system will shake slightly.

Initial State Distribution : The gripper's initial position and the objects' scattered location are randomly generated in the legal region. The location of the pray is fixed and immovable.



(a) internal state $\mathcal{S}$

(b) observe image $\mathcal{O}$

Figure 11: Internal state and Observation

Success Condition (goal) : The object falls into the pray.

Terminal Conditions : 1) The end-effector is outside the legal area; 2) The gripper collided with the pray; 3) Exceed the threshold ε of inverse rank transform times, which $cout(\texttt{rank}\,(s_t) > \texttt{rank}\,(s_{t+1})) > \varepsilon$; 4) Exceeds the maximum episode length (128).

The internal full state s_t is hard to measure or achieve in the real world but easy in the simulator. Thus, we could leverage the expert's prior knowledge to define a sequence of waypoints by the simulator's internal full states, and then train an image-based policy, which could also work in the real world.

Training an end-to-end model in the real world requires too many samples. The current trend, therefore, is to learn an image-based policy in simulation and then transfer them to the real world, which is called *sim2real*.

For overcoming the *sim-real gap*, which is caused by a different sample process $o^{\text{sim}} \sim \mathcal{G}_{sim}\,(s)$ and $o^{\text{real}} \sim \mathcal{G}_{real}\,(s)$, we employed domain randomization[18] to facilitate a smooth domain transfer of the learned policy.

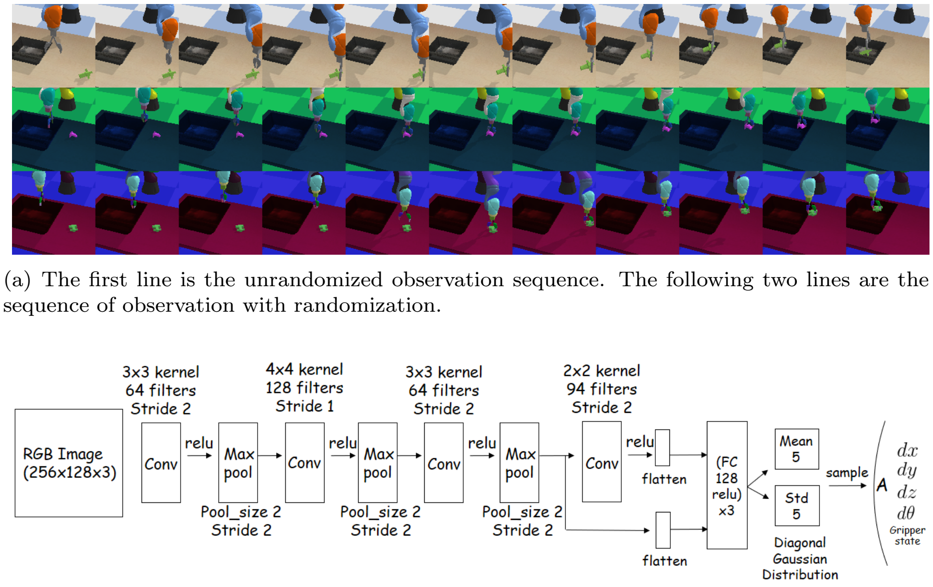
Table 3: rank function

rank	Internal Full State
0	$\text{d}_{\text{to\_obj}} > 57cm$
1	$37cm < \text{d}_{\text{to\_obj}} \leqslant 57cm$
2	$27cm < \text{d}_{\text{to\_obj}} \leqslant 37cm$
3	$18cm < \text{d}_{\text{to\_obj}} \leqslant 27cm$
4	$14cm < \text{d}_{\text{to\_obj}} \leqslant 18cm$
5	$9cm < \text{d}_{\text{to\_obj}} \leqslant 14cm$
6	$5cm < \text{d}_{\text{to\_obj}} \leqslant 9cm$
7	gripper not open
8	gripper open
9	$\text{h} \leqslant 1cm$
10	$1cm < \text{h} \leqslant 4cm$
11	$4cm < \text{h} \leqslant 7cm$
12	$7cm < \text{h} \leqslant 10cm$
13	$10cm < \text{h} \leqslant 15cm$
14	$\text{d}_{\text{to\_pray}} > 57cm$
15	$50cm < \text{d}_{\text{to\_pray}} \leqslant 57cm$
16	$40cm < \text{d}_{\text{to\_pray}} \leqslant 50cm$
17	$30cm < \text{d}_{\text{to\_pray}} \leqslant 40cm$
18	$23cm < \text{d}_{\text{to\_pray}} \leqslant 30cm$
19	$\text{d}_{\text{to\_pray}} \leqslant 23cm$
20	object drop into pray

5.3 Implemention

The rank function defined is based on the expert's understanding of the task using the internal state of the simulator as in Fig.11(a).

The policy is parameterized as a DNN θ. We adopt the PPO algorithm to find the optimal policy$\pi_\theta \to \pi^*$. The policy net is shown as (Fig.12(b)).



(a) The first line is the unrandomized observation sequence. The following two lines are the sequence of observation with randomization.

(b) Actor Net Architecture. The agent learns an actor model in the simulator that only depends on the observe image. We concatenate the last two layers of features for better using global features. Action a is sampled from a multivariate gaussian distribution, the mean μ, and the diagonal standard deviation σ only depending on the input image.

Figure 12: Vision-Based Robotic Grasping

The padding way of CNN is all *valid*. CNN extracts the features from high-dimensional images, and the full connection layer interprets and uses the CNN features. After the tanh active function, the mean, and the standard deviation of the gaussian distribution were output by the full connecting layer, and action a was sampled from the gaussian distribution.

Input images are scaled in $[0, 1]$ for stable training, and the last layer's weight of actor net initialized with 0 before training, which means the mean and standard deviation initialized with 0 before training.

During one training epoch, 5 Kuka workers roll-out episodes data that is based on current policy π_{θ_t}, and evaluate the policy, which updates the value function the same as the classic PPO algorithm. Then, they update the policy to $\pi_{\theta_{t+1}}$ based on the Trust Region policy optimization theory[15].

The discount γ in our experiment is 0.9; The inverse rank transform threshold ε is 3; max episode length is 128.

We experiment on a platform with Intel i7-8700k, 32Gb RAM, GTX 1080Ti, with 48 workers and cost 72 hours for 3 000 000 times environment interaction.

5.4 Result

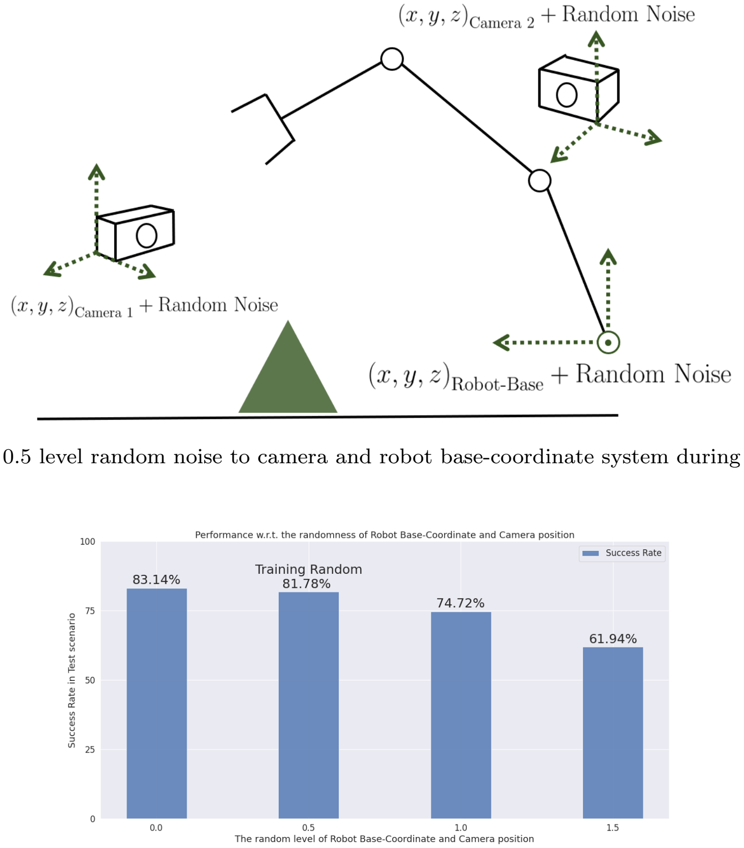
We evaluate our method that the agent can act following the expert expected manner.

The learning curve is shown in the upper of Fig.14(b). An interesting phenomenon is that because there is a penalty for hitting the pray, the agent knows to avoid hitting the pray before grasping the object.

We evaluate our model on the test scenario. The object is never encountered during training. The success rate for the complete process was 83.14% (2993/3600).

5.5 Summary

In this chapter, we design a **Pick and Place** robot task and train a image-based policy under RankTD reward setting. we design the rank function by human priori knowledge of task. Finally, we test our model in test scenario, which the object is never be seen in train phase, and we get 80% success rate.



(a) Add 0.5 level random noise to camera and robot base-coordinate system during training

(b) Then test in different level Random Noise

Figure 13: Add 0.5 level random noise to camera and robot base-coordinate system during training. Then test in different random level.

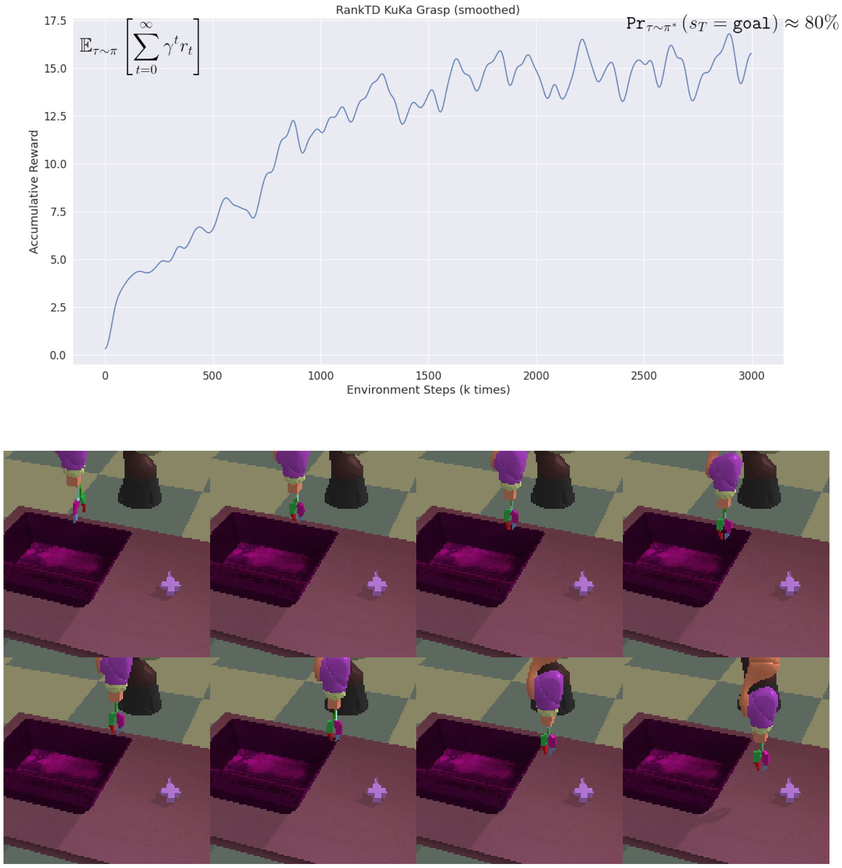


Figure 14: The upper is the training curve, and the lower is the pre-grasp behavior, which avoids hitting the pray.

6 Conclusion and Future work

6.1 Conclusion

In this Thesis, we proposed an approach to RL learning without tedious reward engineering and human demonstrations. In the simple case, we verified the validity of the method and discussed the relationship between the partition granularity of rank function and the bottleneck issue. Then, We apply this method in a complex pick-and-place task, in which the decision-making only require images.

6.2 Future work

There are still some parts could be improved. One is the representation of ovservation. We can deduct high-dimensional observation to low-dimensional state vectors by some unsupervised approach such as the AutoEncoder. Another is reducing the length of rank functions. It is difficult to construct the rank function of a complex task using single internal state variables. And if the rank is defined too long, it is hard to explore a high-rank state. One way to relieve this problem is to modify initial state distribution $p(s_0)$, proposed in DeepMimic[10], which makes high-rank state could aslo be fully explored.

References

[1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[2] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017.

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

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

[5] G. Du, K. Wang, and S. Lian. Vision-based robotic grasping from object localization, pose estimation, grasp detection to motion planning: A review. *arXiv preprint arXiv:1905.06658*, 2019.

[6] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable

distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[7] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

[8] J. Matas, S. James, and A. J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. *arXiv preprint arXiv:1806.07851*, 2018.

[9] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[10] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills, 2018. cite arxiv:1804.02717.

[11] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[12] A. Pore and G. Aragon-Camarasa. On simple reactive neural networks for behaviour-based reinforcement learning. *arXiv preprint arXiv:2001.07973*, 2020.

[13] D. Quillen, E. Jang, O. Nachum, C. Finn, J. Ibarz, and S. Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291. IEEE, 2018.

[14] J. Schulman. *Optimizing expectations: From deep reinforcement learning to stochastic computation graphs.* PhD thesis, UC Berkeley, 2016.

[15] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[16] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[18] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[19] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.