Deep Reinfocement Learning For Robotics Manipulation (II) (Under Construction)

Harry Li ‡, Ph.D.
Computer Engineering Department, San Jose State University
San Jose, CA 95192, USA

Email†: harry.li@ctione.com

***Abstract*—This note describes the unity implementation of the deep reinfocement learning for robotics system control. The objectives of this notes is to provide an implementation reference for state space, action space and reward functions. In addition, this note builds the implementation reference with the connection to the published work by Google deepmind team.**

I. Introduction

This note is prepared based on my discussion notes with CTI One intern team and SJSU graduate students. The core material to form this discussion is from the collection of a set of 4 document:

1. Harry Li's meeting notes, unity-ai-2021-3-12.odt (harry@workstation:/media/harry/ easystore/backup-2020-2-15/ SJSU/CMPE258/258-1-lec/lec7-unity/unity);
2. Harry Li's deep reinforcement learning white paper (Part I on Deep Reinforcement Learning);
3. Robot Arm training github repository, https://github.com/rkandas/RobotArmMLAgentUnity;
4. Published paper by Google Deepmind team.
5. Tutorial on 6 DoF with ML model.

`https://medium.com/xrpractices/how-to-train-your-robot-arm-fbf5dcd807e1`

Our objective of this review is to simulate 6 Degree of Freedom (DoF) model which has 4 Bends, 1 Rotate, and 1 end effector (Gripper) as shown in the figure to realize

1. reaching a random target pose;
2. pick & place an object.

Note, the ML training scope is to teach the robot arm reach the target component from any state. However, the arm should not go below the ground and the target object is always placed above the ground and within the reach of the robotic arm. We will discuss the implementation in the following area

1. the configuration process to carry out the experiment in Unity environment;
2. the policies design;

The goal beyond the review of the research paper is to carry out the real-world experiments in our lab environment on FD100 robot to demonstrate our designed polices and smooth transation to a real world environment for fast readily deployable results.

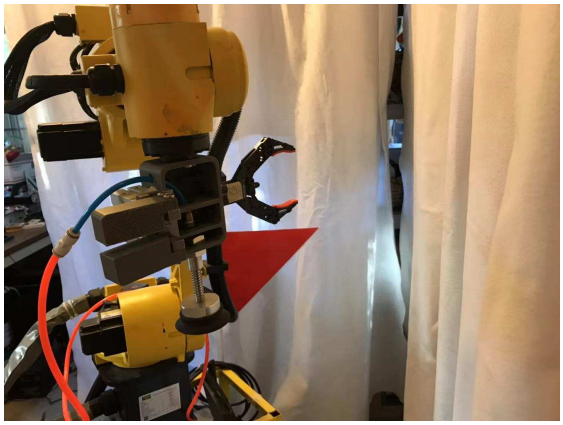

Fig. 1. An end effector (a set of three in this case) from CTI's FD100 robot below, whose movement forms a trajectory in 3D space".

The bigger view of the FD100 robot with an end effector (a set of three in this case) in the view.

Fig. 2. The bigger view of the FD100 robot with an end effector (a set of three in this case) in the view.

II. Experiment Implementation

The material presented here is based on the github material mentioned in the above section.

A. Implementation of State, Action and Reward Functions

Step 1. Create state table; For the 6 DoF robot, we have
Axis 1: the bottom-most axis which rotate 0 to 360 degrees [Rotate state];
Axis 2: the first bend axis from -90 to 90 degrees [Bend state 1];
Axis 3: second bend axis from -120 to 120 degrees [Bend state 2];
Axis 4: third bend axis from -90 to 90 degrees [Bend state 3];
Axis 5: fourth bend axis from -90 to 90 degrees [Bend state 4];
Axis 6: the gripper, Open or Close [Gripper state]

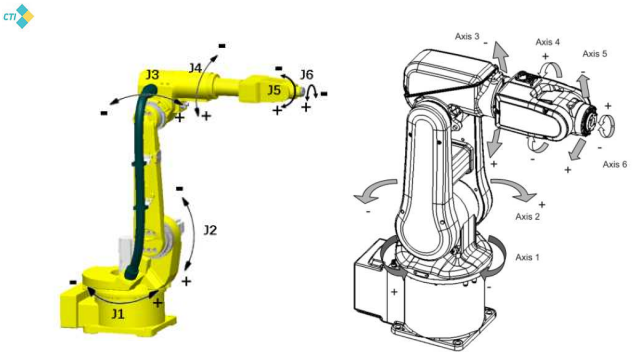

Fig. 3. The robot joints diagram.

The figure is from the Turin Robot User Guide, pp. 23.

Step 2. Create action table. You can find a discrete set of possible actions to take and build a table (a list) of them one by one. For example for driving a car game, steer to left, to rigth, and keep straight are a set of 3 actions. Now for the continued action, for example, action left can be characterized as fuzzy membership function which gives a range [0,1] to describe how much left. So in this case, we will use Deep Deterministic Policy Gradient (DDPG) technique for this type of off-policy methods. The latest development in this area is the Soft Actor Critic (SAC) technique.

Step 3. Create reward (e.g. cost) function table to link each state to its corrensponding action. The reward for each action should be designed. The reward could be positive or negative (Penalty) in numerical value by intuition. Generally, normalized in [-1, 1] range. Here is the 3-point guidelines.

Definition 1. Reward Guideline

(1) When the robot exhibits the desired behaviour, the reward function $r_i > 0$*,*

(2) When the robot deviates from the desired behaviour, the reward function $r_i < 0$*,*

(3) When the robot gets into non-recoverable state like its arm causes harm, such as hitting the ground, the reward function $r_i = -1$*, e.g., hefty penalty,*

(4) When the robot reaches the target, the reward function $r_i = 1$*, e.g., hefty reward.*

Step 4. Create Q-table to link state, action, and reward functions together as illustrated in the example below.

Example: Suppose driving a video game car, one can drive the car to left, right, or keep straight. So the following Q-table can be generated.

State\Action	a1	a2	a3
s1	0.2	2	0.3
s2	2	1	3
s3	0.1	8	2

Q-Table

Fig. 4. Create Q-table to link state, action, and reward functions.

In this figure the actions and states are for driving a video game car. We will need to look into the similar table for 6 DoF robot. From this reference https://blogs.unity3d.com/2020/11/19/robotics-simulation-in-unity-is-as-easy- as-1-2-3_ga=2.25222392.499619345.1615574223-1264198522.1609136839, you can build a simulator to visualize your 6 DoF robot motion.

B. Resources for 6 DoF ML and Simulation

The resouces for 6 DoF ML and simulation:

1. Pick-and-place tutorial on GitHub.

`https://github.com/Unity-Technologies/Unity-Robotics-Hub/blob/main/tutorials/pick_and_place/README.md#part-3-naive-pick--place.`

2. Unity Robotics Hub on GitHub. https://github.com/Unity-Technologies/Unity-Robotics-Hub.

3. Train computer vision systems using Unity, see the computer vision blog series.

`https://blogs.unity3d.com/2020/05/01/synthetic-data-simulating-myriad-possibilities-to-train-robust-machine-learning-models/ .`

4. Unity official robotics page.

`https://unity.com/solutions/automotive-transportation-manufacturing/robotics_ga=2.226719515.725964444.1615792426-1264198522.1609136839`

You can also contact unity team directly with questions, feedback, or suggestions, at unity-robotics@unity3d.com.

III. Experiment Design

Unitys ML-agents training framework is built based on the Markov Decision Process (MDP) which for an ML-agent, e.g., the 6 DoF robot works in the following manner for each time step:

Step 1. The Agent sees a state (from vision camera, and from the robot controller sensors);

Step 2. The Agent takes actions (to drive any one or any combination of its joints and gripper);

Step 3. The Agent receives a reward for each of its action;

Step 4. Then the agent repeats from Step 1.

We have the following table to match each step above to ML-programming.

TABLE I
Programming Aspects for ML

Category	Description	Note % Improvement
State	CollectObservations	Agent observes
Action(s)	OnActionReceived	Agent action
Reward	AddReward	Reward/Penalty

A. Creating ML Agent

We create the Robot as RobotControllerAgent.cs. We perform initialization of the robot as follows, see the program line 32 to 37:

```
public override void Initialize()
{
   ResetAllAxis();
   MoveToSafeRandomPosition();
   if (!trainingMode) MaxStep = 0;
}
```

The robot is the ML Agent which should extend from class Agent, so we replace the default Monobehaviour to Agent in the script and start overriding the Agent methods as above initialization.

IV. Mathematic Formulation

Robotic operations can be characterized as a robot making sequential movement, e.g., control actions in a stochastic environment. The control actions are torques from the controllers to robot joints over a sequence of time steps. From the deep reinforcement learning point of view, the goal of robot operations is to maximize a long term reward. By the mathematical nature, these sequential decision process is modeled as a Markov Decision Process (MDP).

A large class of sequential decision making problems can be formuated as Markov decision processes (MDPs). An MDP consists of

1. a set S of states, denoted as S = $\{s_1, s_2, ..., s_N\}$, and

2. a set A of actions, denoted as A = $\{a_1, a_2, ..., a_M\}$ as well as

3. a transition function T and

4. a reward function R,

denoted as a tuple $< S, A, T, R >$. When in any state $s \in S$, an action $a \in A$ will lead to a new state with a transition probability $P_T(s, a, s\prime)$, and a reward R(s, a) function.

The stochastic policy $\pi : S \rightarrow D$ maps from a space state to a probability over the set of actions, and $\pi(a|s)$ represents the probability of choosing action a at state s.

The goal is to find the optimal policy π^* to produce the highest rewards [Rein, 2020]:

$$\arg\max_{\pi \in \Omega_\pi} E[\sum_{k=0}^{H-1} \gamma^k R(s_k, a_k)]\} \tag{1}$$

The composition of the above equation can be described as follows. First, define reward function at time k:

$$R(s_k, a_k) \tag{2}$$

with discounted factor γ at time k, and $\gamma < 1$ (to formulate the older state, as the k becomes bigger, its effects on the reward will get smaller):

$$\gamma^k R(s_k, a_k) \tag{3}$$

Note In finite-horizon or goal-oriented domains, choose discount factors close to 1 to encourage actions towards the goal, whereas in infinite-horizon domains choose lower discount factors to achieve a balance between short- and long- term rewards.

For all the experiments up to horizon H, we have summation:

$$\sum_{k=0}^{H-1} \gamma^k R(s_k, a_k) \tag{4}$$

For the stochastic nature of these rewards, we use statistical expectation as

$$E[\sum_{k=0}^{H-1} \gamma^k R(s_k, a_k)] = \int_{\inf} \gamma^k R(s_k, a_k) P(s_k, a_k) ds \tag{5}$$

Hence, we have the average discounted rewards under policy π.

V. DRL Connection to Robot Control

Definition 1. *Trajectory τ. A trajectory τ of a robot motion is defined as a sequence of state-action pairs in time sequence $t_1, t_2, ..., t_N$, denoted as $\tau = (s_1, a_1, s_2, a_2, ..., s_N, a_N)$.*

Consider any trajectory τ of a robot motion, e.g., $\tau = (s_1, a_1, s_2, a_2, ..., s_N, a_N)$, the trajectory of the end effector in 3D space. See an end effector (a set of three in this case) from CTI's FD100 robot below,

Definition 2. *Reward r. A reward r is defined as numerical value assigned to each state-action pair $s_i a_i$, e.g., formulate as $r : S \times A \rightarrow R$, where $S \times A = (s_1 a_1, s_2 a_1, ..., s_1 a_N, s_2 a_N, ..., s_N a_1, ..., s_N a_N)$.*

A reward r is formulated as $r : S \times A \rightarrow R$, where $S \times A = (s_1 a_1, s_2 a_1, ..., s_1 a_N, s_2 a_N, ..., s_N a_1, ..., s_N a_N)$.

We can build tables for S, A and R repectively, as follows

TABLE II
TABLE I. STATE TABLE FOR FD100 ROBOT

Category	Description	Note % Improvement
$s_{j1-angle}$	[,]	Continuous
$s_{j1-speed}$	[,]	Continuous
$s_{j1-accel}$	[,]	Continuous
$s_{j2-angle}$	[,]	Continuous
$s_{j2-speed}$	[,]	Continuous
$s_{j2-accel}$	[,]	Continuous
...	...	...
$s_{j6-angle}$	[,]	Continuous
$s_{j6-speed}$	[,]	Continuous
$s_{j6-accel}$	[,]	Continuous

TABLE III
TABLE II. ACITION TABLE FOR FD100 ROBOT

Category	Description	Note % Improvement
$a_{j1-angle}$	[,]	Continuous
$a_{j1-speed}$	[,]	Continuous
$a_{j1-accel}$	[,]	??? Check this
$a_{j2-angle}$	[,]	Continuous
$a_{j2-speed}$	[,]	Continuous
$a_{j2-accel}$	[,]	Continuous
...	...	...
$a_{j6-angle}$	[,]	Continuous
$a_{j6-speed}$	[,]	Continuous
$a_{j6-accel}$	[,]	??? Check this

TABLE IV
TABLE III. REWARD TABLE FOR FD100 ROBOT

Category	Description	Note % Improvement
$s_{j1-angle} \times a_{j1-angle}$	[,]	Continuous
$s_{j1-speed} \times a_{j1-angle}$	[,]	Continuous
$s_{j1-accel} \times a_{j1-angle}$	[,]	Continuous
...	...	...

Note, check the datasheet of FD100 to fill in the numerical values in the description section of the above tables.

VI. POLICY ON ROBOT OPERATIONS

Definition 3. *Policy π. A policy π in Robotics is a set of guidelines for a robot controller to follow to deliver its control action a_i upon its current state s_i, which is denoted as $\pi(a_i|\ s_i)$.*

A policy π leads to the robot control to deliver its control action as a mapping function $s_i \rightarrow a_i$.

A policy pi can be either stochastic which is characterized by a conditional probability as

$$\pi(a_i|s_i) : s_i \rightarrow Pr(a_i|s_i), \tag{6}$$

or deterministic

$$\pi(a_i|s_i) : s_i \rightarrow a_i = \mu(s_i). \tag{7}$$

VII. POLICY AS DNN

We now introduce a notation to policy π as π_θ where a set of variables which affects π. In deep reinforcement learning (DRL), a policy π_θ is formulated as a deep neural network (DNN), where θ is the general parameter storing all the networks weights and biases $W = (w_{i,j})$.

Definition 4. *Policy π_θ. A policy π_θ is a deep neural network (DNN), where θ is the collection of all parameters of the neural networks (NN) weights and biases, simply denoted as $W = (w_{i,j})$.*

A typical realization of π_θ is a gaussian Multi-Layer Perceptron (MLP) net, which samples the action to be taken from a gaussian distribution of actions over states as follows

VIII. QUICK REVIEW

The objectives of deep reinforcement learning is to develop algorithms to make robot has the ability to learn new manipulation policies

(1) from scratch, without user demonstrations and

(2) without the need of a task-specific domain knowledge.

A. Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) [1]

B. Deep Q-Network

Deep Q-Network with Normalized Advantage Func- tions (DQN-NAF) [

C. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) [3] and

D. Vanilla Policy Gradient

Vanilla Policy Gradient (VPG) [4].

E. hyper-parameters selection and tuning procedures

The hyper-parameters selection and tuning procedures

as well as demonstrate the robustness and adaptability of TRPO and DQN-NAF while performing manipulation tasks such as reaching a random position target and pick & placing an object.

Moreover, their model-freedom guarantees good performances even in case of changes in the dynamic and geometric models of the robot (e.g., link lengths, masses, and inertia).

???

ACKNOWLEDGMENT

I would like to express my thanks to CTI One engineering member and CTI One engineering intern team, as well as to SJSU graduate research assistants for letting me introducing DRL in Robotics to our next project and for the codings in Deep Learning, Robotis and embedded systems.

REFERENCES

[1] [Franceschetti, 2020] Andrea Franceschetti, Elisa Tosello, Nicola Castaman, and Stefano Ghidoni, "Robotic Arm Control and Task Training through Deep Reinforcement Learning", https://arxiv.org/pdf/2005.02632.pdf, May 2020.

[2] [Reinforcement, 2020] Reinforcement learning, https://en.wikipedia.org/wiki/Reinforcement_learning, 2020.