Deep RL Arm Manipulation Project

Author: Tuan Le

Date: 10th December 2018

I. Introduction

This project aims to create a DQN agent and define a reward system to teach a robotic arm to achieve following objectives:

- Have any part of the robot arm touch the object of interest, with at least 90% accuracy for a minimum of 100 runs
- Have only the gripper base of the robot arm touch the object, with at least 80% accuracy for a minimum of 100 runs.

The project was implemented with Gazebo and PyTorch C++ plug-in to interface a simulated robotic arm with a DQN agent. The two objectives above were achieved by taking following steps to setup a DQN agent and train it.

- Create subscription to the camera and collision topics published by Gazebo environment
- Create a DQN agent with all required parameters
- Define velocity control and position control according to the tasks
- Design reward/penalty system for the robotic arm when hitting the ground and touching the desire object
- Design an interim reward system to encourage the robotic arm in reaching its objective
- Tune DQN hyperparameters to achieve the required accuracies above

II. Reward Functions and Control Strategy

Deep Q-Network (DQN) output is usually mapped to a specific action, which is the control of each joint for the simulated robotic arm. There are two types of controls can be used in this project, one is velocity control and the other is position control. My control strategy was velocity control for the first task.

Control Strategy

Objective 1

The first objective only requires any part of the robotic arm to touch the object. Both method of velocity and position control were tested to determine the most suitable one. The position strategy used in the first objective scenario had shown to be ineffective due to some overturn from the "actionJointDelta" variable value. This was fixed by reducing the value; however, this resulted in the robot could not reach the object sometime. The velocity control was tested to reach the object in a controllable manner. The result was satisfied and met the requirement on the objective 1. Further explanation on this result is written in the Results section.

Objective 2

For the objective 2, fine joint motions were priority; therefore, the position control was used. The velocity control strategy was tested but it took longer for the robot control its gripper.

Reward functions

The reward system was designed to reward points differently in each objective. In general, there are common rules applied on both cases, namely an object touching reward of 100 points and a ground hitting penalty of -100 points. Moreover, the number of attempts in which the robotic arm tried reach the object, was limited to 100 times. If the robot reaches the pre-defined number of attempt without winning, a penalty of -100 points will be issued, and the episode will be ended.

For the interim reward function, an interim reward/penalty will be issued according to the distance between the object and the robot moving part. At the end, the robotic arm will win a reward if its moving part touching with the object, and the episode is ended. Below are the formulas to calculate distance between the arm and the object.

There are two formulas to calculate the interim reward based on the distance between the object and robot parts. They are as follow:

$$\Delta Distance = D_{t-1} - D_t$$

$$\Delta AverageGoal = (\Delta AverageGoal \times \alpha) + (\Delta Distance \times (1.0 - \alpha))$$

The interim reward, then, is issued depend on each objective. Furthermore, several conditions were created to make the robotic arm's gripper base to touch the object.

Objective 1 – Because any part of the robotic arm can touch the object, the interim reward is issued according to the average goal delta.

$$R_{Interim} = \Delta AverageGoal \times interim\_REWARD$$

Objective 2 – Because only the gripper base can touch the object, the interim reward is issued with a condition below.

If the $\Delta Distance$ is greater than 0.001f, the interim reward is a tenth of a full winning reward (10 points). On other hand, if the $\Delta Distance$ is less than or equal 0.001f, the interim reward is issued like the formula below.

$$R_{Interim} = \Delta AverageGoal \times interim\_REWARD \times 0.1$$

The first condition is to encourage the robotic arm move faster toward the object. Then, the second condition reduces the speed to establish a refine movement, thus, overcome overshoot of the gripper base. To achieve this outcome, a small amount of interim point is given according to the average goal delta.

Reward System Flowchart

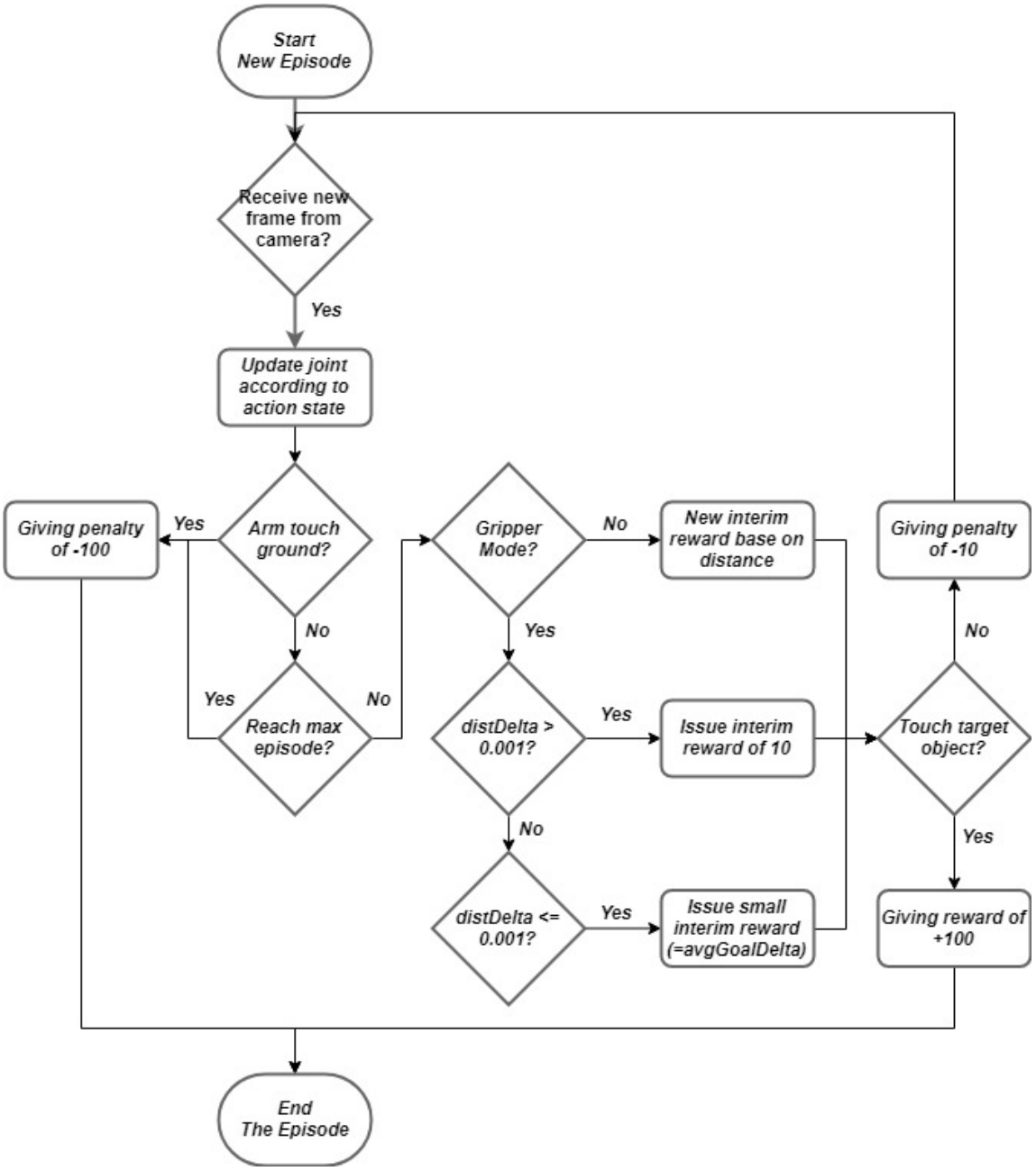

Figure 1: Flowchart of the reward function

III. Hyperparameters

With two objectives above, these hyperparameters were adjusted accordingly. Below will give a brief explanation to each hyperparameter function and value choice rationale.

INPUT_WIDTH and ***INPUT_HEIGHT***: a camera frame which fed into the DQN agent every simulation iteration. Big frame size will take more memory and computing power to process. Therefore, 512x512 size is redundant. The size was reduced to 64x64 to have more accurate learning.

OPTIMIZER: Adam, RMSprop, and Adagrad are three of the many variation of gradient descent optimizers. The robotic arm was tested with RMSprop and Adam but RMSprop results in a fast-stable outcome. Thus, RMSprop was chose for both Task 1 and 2.

LEARNING_RATE: This hyperparameter determines an increment of weights toward the direction of gradient decent. Subsequently, it leads to change in the training speed of a neural network. High

learning rate makes training finish faster and vice versa. However, moving fast also means overshooting and increase the risk of validation loss. With these information, the learning rate was experimented in a range between 0.1 and 0.01. The final values were 0.01 and 0.02 for task 1 and 2 respectively.

REPLAY_MEMORY: REPLAY_MEMORY is a cyclic buffer that stores the transitions observed recently. A transition is a state change of a DQN agent. Plus, REPLAY_MEMORY can randomly select a batch of transition for training. The default value of this hyperparameter is 10000 which means it can store up to 156 states (10000/batch_size). For this robotic arm with two joints, 156 of stored states is a reasonable value.

BATCH_SIZE: batch_size determines how many data will be sent and how many iterations is needed to finish a training. Additionally, a large batch_size will cost in large memory and computing usage. Through many experiment with 128, 64 and 32 in batch_size, the 64 is a right choice in both tasks. With batch_size of 64, a deep neural network can perform efficiently and effectively in training the robotic arm.

USE_LSTM: This is to turn on the long Short Term Memory as part of DQN. By enabling this parameter, DQN agent can consider past frames from the camera to improve its learning.

LSTM_SIZE: Size of a LSTM cell. Just like batch_size, the bigger the LSTM_size, the more computing usage will be needed. 128 was tested and achieved the required result.

REWARD_ALPHA: This is a learning rate a Q-learning part of DQN. Big reward_alpha leads to increase significantly to the probability of learned value. The robotic arm has totally six actions. Plus, task 1 and 2 have different requirement for the robot joint motion. Therefore, the reward_alpha value was chosen as 0.3 and 0.35 for task 1 and 2 respectively.

Summary tables

Table 3.1: All hyperparameters in task 1

Hyperparameter	**Value**
INPUT_WIDTH and INPUT_HEIGHT	*64x64*
OPTIMIZER	*RMSprop*
LEARNING_RATE	*0.01*
REPLAY_MEMORY	*10000*
BATCH_SIZE	*64*
USE_LSTM	*True*
LSTM_SIZE	*128*
REWARD_ALPHA	*0.3*

Table 3.2: All hyperparameters in task 2

Hyperparameter	**Value**
INPUT_WIDTH and INPUT_HEIGHT	*64x64*
OPTIMIZER	*RMSprop*
LEARNING_RATE	*0.02*
REPLAY_MEMORY	*10000*
BATCH_SIZE	*64*
USE_LSTM	*True*
LSTM_SIZE	*128*
REWARD_ALPHA	*0.35*

IV. Results

First task: Having any part of the robotic arm touches the object with accuracy of 90%

Before finely tuning hyperparameter, the robotic arm appeared to have display an overshoot situation in the robot when it tried to reach the object. Plus, another problem was that hyperparameter learning_rate and reward_alpha when not adjust and tune caused the robot to not converge. Reward_alpha with low value leads to slow learning in a Q-network. On the other hand, high reward_alpha value leads to unstable movements.

Suggested solutions to tackle the problem were listed as follow:

- Change to velocity control and limit to the range from -0.1 rad/s to 0.1 rad/s.
- Trial and errors approach to find a reasonable range of learning_rate and reward_alpha.

After applied both solutions. The result showed that the robotic arm was able to gradually touch the object. After 100 episodes, the robot achieved objective 1. The learning_rate range was adjusted to a range from 0.01 to 0.1. Then, the robotic arm was tested with 0.02, 0.04, 0.06, and 0.08. 0.02 is found to give a desired result but a fine tune to 0.01 gave more satisfied outcome; thus, 0.01 was the final value.

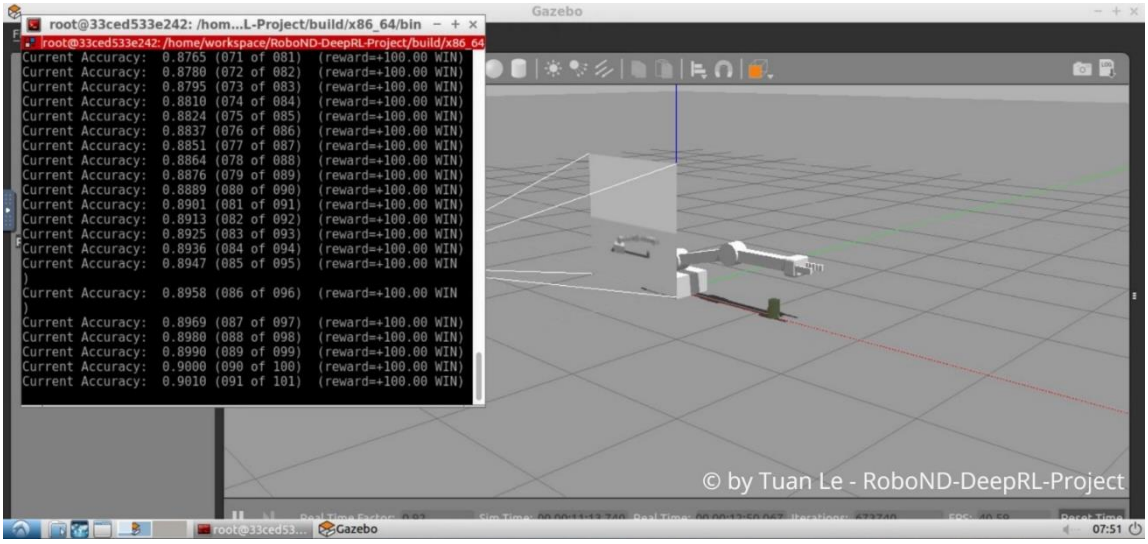

Figure 2: Task 1 screenshot shows the accuracy reached 90%

Second task: Having the gripper base touches the object with accuracy of 80%

Using the reward system from task 1 causes the arm to fully extend. Thus, this made the robot to over-reach the object. Solution to overcome this problem was to create a two-parts reward system, the first part is when the distGoal is greater than 0.001, the second part is when distGoal is smaller than 0.001. The first part is to make the robot closing the gap quickly through issuing a tenth of REWARD_WIN. The second part is to help the robot to achieve more refine movements by issuing small reward (a tenth of interim_REWARD multiply by avgGoalDelta).

After testing, the result shows that small rewards in the second part helped the robot to be able to adjust the second joint. At the same time, the gripper base was able to touch the object. Figure 3 captured the accuracy of 85% of the DQN agent after 140 episodes.

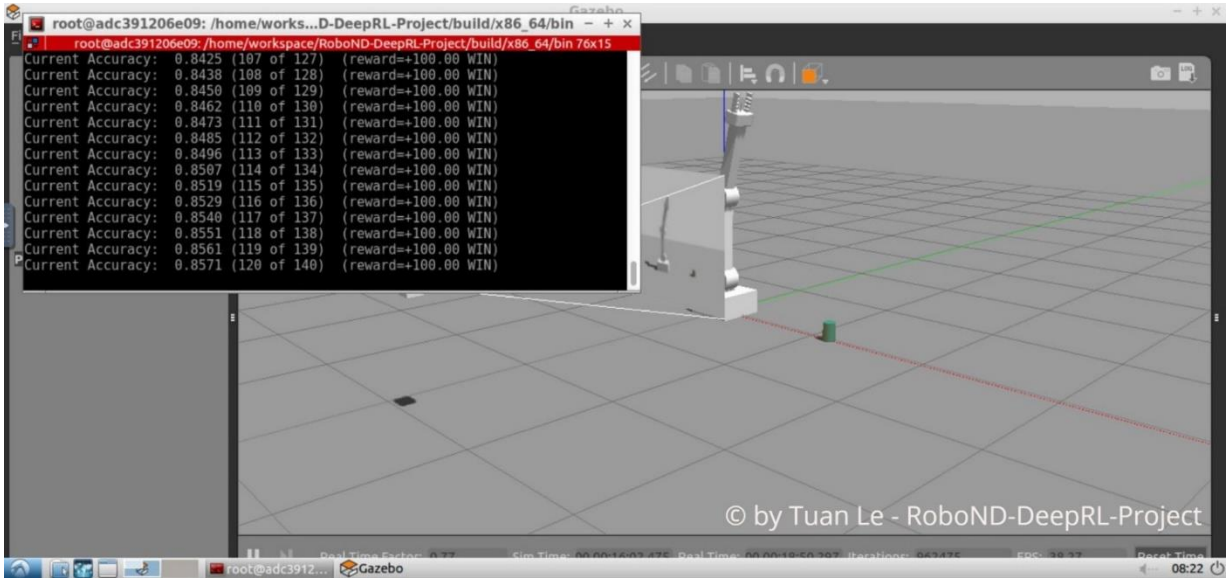

Figure 3: Task 2 screenshot shows the accuracy reached more than 80%

V. Future Work

There are many paths to explore to make the DQN agent achieve a natural movement and fast learning (achieve 90% accuracy within 90 episodes for both task 1 and 2). This can be reached by considering the two following approaches.

Firstly, exploring further and deeper hyperparameter effects to find a better hyperparameter set. At the same time, create a more detail-oriented reward system that cooperates with the control strategy used. This approach will require significant of time of experiments. The functions that impact on the joint angle and velocity of the robotic arm, was largely affected by the action set. From this, a reward system can be designed to input its reward results into the control functions.

Secondly, introducing a path planning algorithm into the system. The path planning will help the robotic arm to create a better trajectory for its path. However, this is out of the scope the project. Considerations about computing power need also to include if the path planning is included.