Deep RL Arm Manipulation

Erick Ramirez

Abstract—This document describes a solution of Deep reinforcement learning, in this case applied to an Arm manipulation project as port of the Robotics Nanodegree program. For the previous task a DQN agent has been used and reward functions have been defined to teach a robotic arm to touch a specific object.

Index Terms—Reinforcement Learning, DQN, robot Arm Manipulation

1 Objectives

The goal is to create a DQN agent and define reward functions to teach a robotic arm to carry out two primary objectives:

1) Objective 1: Have any part of the robot arm touch the object of interest, with at least a 90% accuracy.
2) Objective 2: Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy.

The code for this

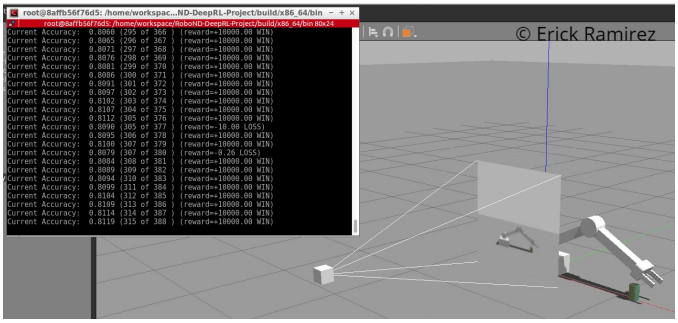

Fig. 1. The gripper base of the robot arm touch the object, and it has achieved the objective 2

2 Reward Functions

There is defined the following constants: REWARD_WIN and REWARD_LOSS. the WIN is used when the robot touches the object of interest, when it touches with the gripper base (objective 2) or any part of the arm (pro or gripper base). It is considered a LOSS when the arm does not touch the object of interest, because the length of the episode exceeds 100 steps or the arm has ground contact. At the beginning, it will be very rare to touch the object of interest, that is why there are other functions used as interim reward. it is because, they will guide to robot arm towards the object. This is the definition of reward functions: I have defined:

- REWARD_LOSS = -10
- REWARD_WIN = 10

2.1 Objective 1: Have any part of the robot arm touch the object of interest, with at least a 90% accuracy

- If any part of the robot touch the ground, the penalty is REWARD_LOSS * distance to the Goal * 10
- If any part of the robot touch the target and it is not the gripper base, the reward is REWARD_WIN * 10
- If the gripper base touch the target, the reward is REWARD_WIN *1000
- If the length of the episode exceeds 100 steps, the penalty is REWARD_LOSS *

In this case it will be consider a WIN if the robot touch the target, but it will have a better reward if the gripper base touch the target. Note: to use objective one in the file 'gazebo/ArmPlugin.cpp' update the line 69 to :'#define OBJECTIVE 1'

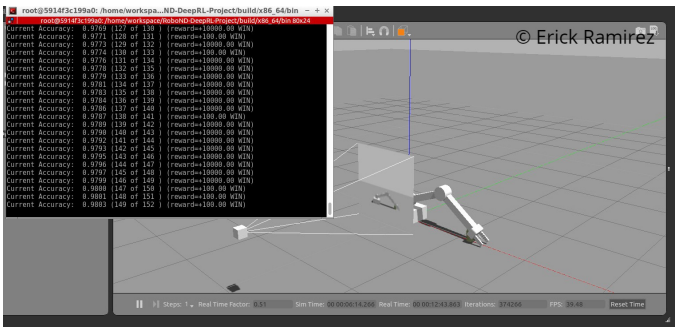

Fig. 2. arm reaching the objective 1

2.2 Objective 2:Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy

- If any part of the robot touch the ground, the penalty is REWARD_LOSS * distance to the Goal * 10
- If any part of the robot touch the target and it is not the gripper base, the penalty is REWARD_LOSS
- If the gripper base touch the target, the reward is REWARD_WIN *1000
- If the length of the episode exceeds 100 steps, the penalty is REWARD_LOSS *

In this case it will be consider a LOSS if the robot touch the target but it is not the gripper base, then it is avoiding to perform this action. It is consider a WIN only when the

gripper base touch the target. Note: to use objective one in the file 'gazebo/ArmPlugin.cpp' update the line 69 to :'#define OBJECTIVE 2'

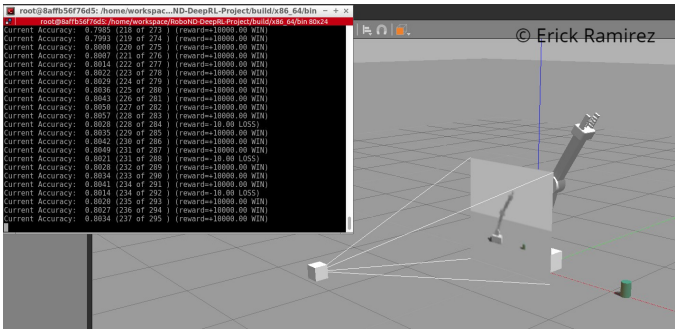

Fig. 3. arm reaching the objective 2

2.3 Issue an interim reward based on the distance to the object

it uses average_delta function (in the code it is defined as avgGoalDelta) which is: avgGoalDelta = (avgGoalDelta * ALPHA) + (distDelta * (1.0f - ALPHA));

The avgGoalDelta function was really useful for the definition of the interim reward function, because it is proportional to the average speed towards the goal. which means that we expect to compute the smoothed moving average of the delta of the distance to the goal: rewardHistory = REWARD_WIN * avgGoalDelta / (20);

2.4 Joint control

For this robot we are using a revolute join,

- Velocity Control: The current value for a joints velocity is stored in the array "vel" with the array lengths based on the number of degrees of freedom for the arm. in this case it was modified the following
  - velocity = vel[action/2] + even_odd * actionVelDelta;
- Position control: The current value for a joints position is stored in the array ref with the array lengths based on the number of degrees of freedom for the arm
  - joint = ref[action/2] + even_odd * actionJointDelta;

Increase and decrease are the two outputs for every action. and the result is based on if the is even or is odd.

3 DQN Hyperparameters

For both objectives the hyper-parameters are the following: The input size is 64 64, this is the resolution of the image to be processed. the optimizer used is RMSprop because it is a good choice for recurrent neuronal networks(RRNs), and the long-short-term memory was enabled (LSTM = true) to track the long-term memory and the short-term memory. The learning rate used is 0.0001 (this was elected based on try and error). the batch size used is 256 and the LSTM_SIZE is 256 and the REPLAY_MEMORY 10000. Most of the tunning was related to test multiple scenarios once the reward functions were define.

4 Results

You can show in Fig 3. and Fig 4. that in both simulation the DQN agent's performance, the objective 1 and objective 2 have been achieved. The ALPHA value had a great impact in the behavior to achieve both objectives. And for the objective 2 the major issue was to assign a correctly reward function, including the interim reward based on the distance to the object. As it was mentioned before, for both cases the hyper-parameters selected are the same. There are some shortcomings on this model: for objective 2, the model doesn't even start to the right direction, and it start to repeat a win once it reached for first time the target with the gripper base. in order to repeat this action, is adding a big number of reward to that action. Sometimes it is not performing a smooth path of actions to reach the target.

This is the result of the objectives achieved:

4.1 Objective 1: Have any part of the robot arm touch the object of interest, with at least a 90% accuracy

the agent started to have an accuracy greater than 90% almost immediately, as it can be appreciated in the Fig 2. during the test 152 it already got 149 WINs.

4.2 Objective 2:Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy

the agent started to have an accuracy greater than 80% after 275 runs, as it can be appreciated in the Fig 3.

5 Future work

The accuracy can be improved by more tunning on the DQN hyperparameters, and the reward functions can be more complex functions to evaluate in order to perform a smoother actions to reach the target.. About the robot, it could have more sensors to evaluate the distance or the map related to the object to be achieved. For instance the camera resolution could be increased to evaluate more than 60x64 inputs.Also, for more complex actions, arms and joints can be added to increase the degrees of freedom. And it could be tested on a arm with more degrees of freedom, this will be more challenging but feasible to do, and the implementation in the JetsonTx2 will be very interesting.