Deep RL Manipulator

Panagiotis Angelakis

Index Terms—Robot, IEEEtran, Udacity, LaTeX, deep learning.

1 Introduction

The goal of this project is to teach a 3 DOF robotic arm to execute two primary functions. For that we need to create a DQN agent and design a reward mechanism suitable for achieving this functions.

- Have any link of the robot touch the prop placed on the ground with at least 90% accuracy.
- Have only the end effector touch the prop with at least 80% accuracy.

The project uses a Gazebo simulation for the manipulator so a C++ plug-in to interface the Gazebo with the DQN agent is needed. The project can be divided into several tasks those are :

- Subscribe to camera and collision topics published by the physics simulator Gazebo.
- Create the DQN Agent.
- Define a position based control for the Manipulator joints.
- Penalize if any part of the robot touches the ground.
- Interim Reward based on the manipulator distance to the prop.
- Reward based on Manipulator suscesfully touching the prop (any part or only the gripper base).
- Tuning the Hyperparamaters.

2 Reward Process

The purpose of our Deep Q-Network is to provide a particular action to be executed from the robot. Specifically for this project we want to map the output to move each joint of our robotic arm. Control of the joints can be position, velocity or a mix of both. Here we chose position control.

The Reward system was designed in such a way to train the robotic arm to first : have any part of the arm touch the object of interest and in the second iteration : only the gripper to touch the prop. Below we can see the reward diagram.

Each episode is limited to a certain number of movements and a penalty will be issued if we were not succesfull in this duration, also a penalty is issued when any part of the robot comes into contact with the ground. Furthermore for better training and results we provide an interim reward or penalty as the robot moves towards or away from the object (greedy approach). Finally when the robot arm succesfully touches the object of interest we issue a Win reward and we terminate the episode.

Interim rewards are issued based on a smoothed average of the delta of the distance bewteen the arm and the prop.

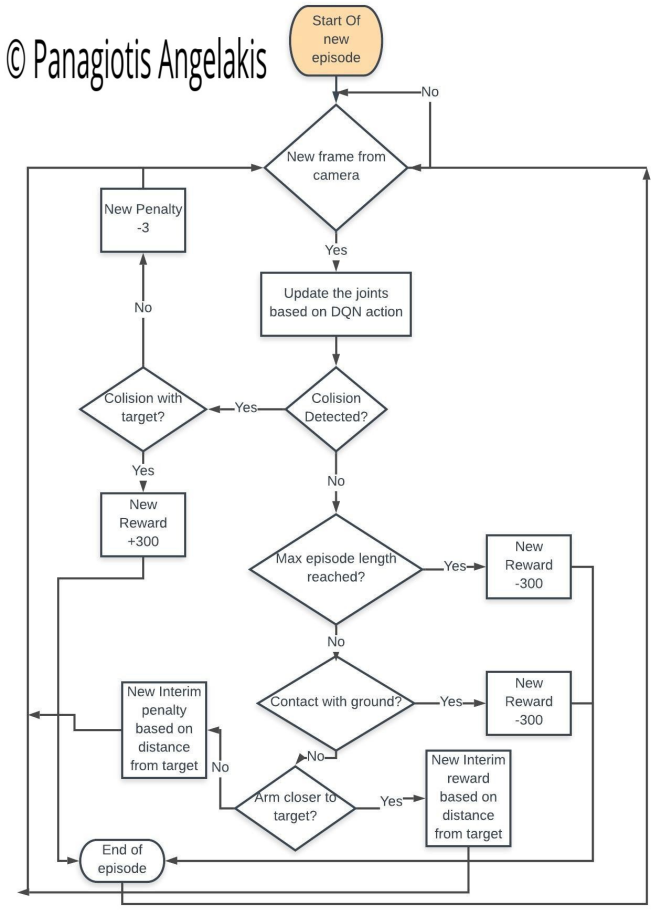

Fig. 1. Flow chart of the Reward issuing process

Distance delta = last Distance to Goal - Current Distance to goal Average Goal delta = (Average Goal delta * alpha) + (Distance delta * (1.0 - alpha)) Interim reward is analogous to Average Goal Delta.

3 Hyper Paramaters

As any Deep learning project there are several Hyper paramaters that are needed to be adjusted before we have satisfactory results. Those are :

- **Input Width x Input Height** : every camera frame coming from the simulation is fed into the DQN agent, which afterwards performs an action. Bigger frame size requires more memory and computing

power. 64x64 had enough detail for our purposes and didn't hidder the accuracy score, while 512x512 created memory related errors.

- **Num Actions** : Based on the particular robot number of actions that the DQN outputs must be decided. In our case the robotic arm had 3 joints each one requiring two actions, one to increase and one to decrease the position or velocity of each joint. So Num of Actions = 2 * DOF = 6.
- **Optimizer** : Gradient descend comes in many flavours : RMSprop, Adagrad, Adam.All dynamically adjusting the learning rate. For this project RMSprop was tested with satisfactory results.
- **Learning rate** : This paramater tell the optimizer what step to use to move the weights in the direction of the gradient for each batch. If we choose a very small learning rate the training may be more reliable but it also needs a lot of time to train, while if the learning rate is too large it can overshoot local minimum. This is an iterative process to determine an appropiate learning rate. For the first task a = 0.1 was chosen and perform well, while for task 2 which was much harder 0.1 didn't reach the desired accuracy but with a = 0.01 it eventually got there.
- **Batch Size** : The bigger the Batch Size less iterations will be needed but at the cost of more memory and compute power, this can also affect the suitable size of the Replay memory. Here we chose the batch size to be **32**.
- **Replay Memory** : It consists of a cyclic buffer that stores the transitions, which the DQN observes for later reuse by sampling from it in random. This is a common practice for DQN networks because it greatly stabilizes and improves the DQN training procedure. It was chosen to be 10000 which can store approximately 10000/Batch Size = 312 states to be reused randomly. If we had an arm with more DOF this could be further increased.
- **Use LSTM** : Enabling the (Long Short Term Memory LSTM) will allow the DQN to take into consideration not just the current frame provided by the camera sensor but also multiple past frames increasing the robustness of the learning process.
- **LSTM Size** : It is the size of each LSTM cell, 512 was causing memory errors so 256 was chosen.
- **Reward Win** : The Reward that will be issued when the robot succesfully touches the prop, here was set to 300.
- **Reward Loss** : The penalty that will be issued in case of the robot 1) Touching the ground (any part) 2) Exceeding the number of iterations allowed per episode. For Task 2 we also deduct 10% 300 = 30 points when the robot touches the robot but not with the gripper base.
- **Reward Mult** : A multiplier used to control the amount of points given in each interim reward or penalty based on the change of the distance bewteen the arm and the prop. Here we chose 200 (delta distance ranges from 0.4 to 1.4) giving us a reward ranging from 80 to 280
- **alpha** : Smoothing factor to control average distance, and was chosen to be 0.3
- Some Additional paramaters remained unchanged: Input_Channels = 3, Gamma=0.9, Eps_Start=0.9, Eps_End=0.05 and Eps_Decay=200.

4 Results

Before the process of fine tuning the hyper-paramaters and setting the appropiate reward system the DQN was performing poorly with very low accuracy, not completing the objective or crushing due to memory related errors. However after fine tuning the Hyper paramaters, for task 1 we reached 100% due of the arm essentially falling down touching the prop.

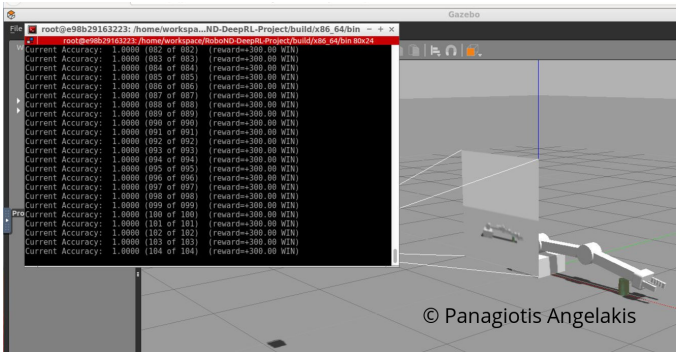

Fig. 2. Result of any part of the robot touching the prop

For task2 which only the gripper was allowed to touch the prop, only the learning rate was lowered to 0.01, all other hyperparamaters were left unchanged, however the training process took much longer over 700 episodes just surpassing the required accuracy.

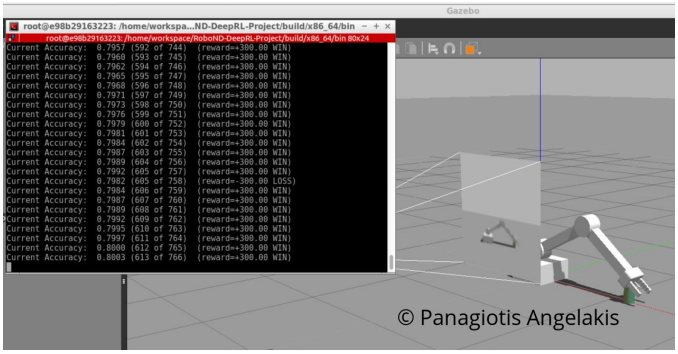

Fig. 3. Result of gripper touching the prop

5 Future Work

One approach to improve our results is to graph the accuracy progress for every change in paramaters and then identify the maximum accuracy that can be achieved and how many iterations it needed. This however is a brute force approach not always applicable.

Our arm is a simple arm with only 3 DOF and no rotating base, a more complicated problem will be to train a 6 or 7 DOF arm, which are more common in industry.

Furthermore what if our arm was on a mobile base preferably an omni-directional one? In this scenario we can't use an external camera to capture the frames required for training to pick up objects. An onboard camera must be used either on the base it'self or the end effector. This is a more complicated problem because we have actions of movement of the base and also the frames coming from the camera are not ″static″ in space.