DEEP REINFORCEMENT LEARNING ARM PROJECT

POORNIMA L NATHAN

TABLE OF CONTENTS

Contents

Abstract

Deep Reinforcement Learning involves letting an agent learn what is right and wrong based on the reward/penalty that is returned because of its actions. Owing to the goal orientation involved in this learning, algorithms of this kind are expected to perform better in real time environments where uncertainty is higher than controlled test conditions. The Deep Q Learning algorithm is famous due to the fact that its chooses the best action based on the current environment, determined by its Q value. This project involves two parts:

1. Have any part of the robot arm touch the object without hitting the ground
2. Have only the gripper touch the object without hitting the ground

Here reinforcement learning is implemented by assigning a reward when the robot attains the objective in the corresponding run and a penalty if it touches the ground, stays still or takes too long to move.

Project Setup

The below picture shows the setup. The robotic arm is simulated in Gazebo with the object in front of it. The Deep Q Learning algorithm is setup to attain the objectives. Also, PyTorch is also set up to provide feedback to the arm. The scope of the first objective is broader than the second one. Hence the reward and penalty were increased while running the second objective. Meanwhile there are also intermediary rewards provided while the arm moves towards the object and penalties are handed out when it hits the ground or if any other part of the arm touches the object. Although these are different, the final reward/penalty for that episode is handed over at a final action or at the end of an episode. This final reward/penalty needs to be brought forward to the next episode as part of learning how close the arm was able to reach the object. Hence the smoothed moving average is calculated for the arm to correlate between its actions and the gradual feedback. Also, the parameter alpha is set closer to 0 since the more recent actions are to be taken into considerations rather than the entire set. This simulates the forget or remember part of the agent. Also, the rewards/penalties are handed over based on the outcome, and not how fast this was achieved. So the reward function has minimal considerations for the velocity.

Hyperparameters

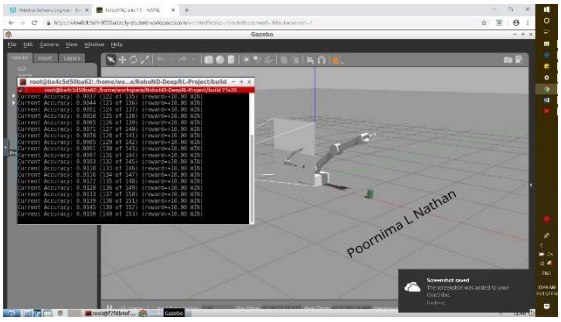
The following parameters were taken mainly into consideration for achieve the objectives:

INPUT_WIDTH and INPUT_HEIGHT – Both these defined the amount of pixels that go into the camera image. This environment is fairly simple, thus eliminating the need for a larger set of values. Hence this was reduced to 64. Also, setting both parameters to the same value enabled the capture of a square image, thus reducing the complexity of matrix operations. To correspond to this logic, the BATCH_SIZE was also reduced to 64.

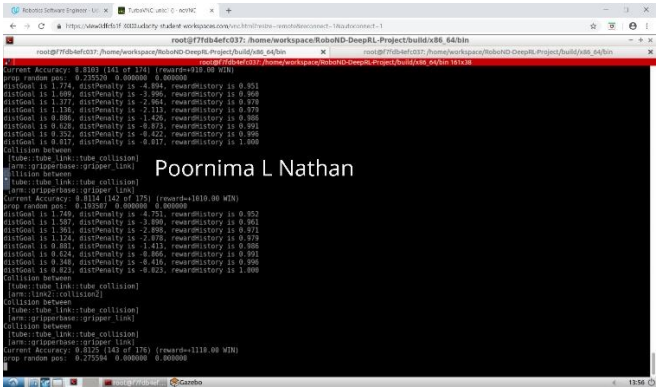
OPTIMIZER – After trying out 3 optimizers – Adam, RMSProp and SGD, I found Adam optimizer to work much better to my satisfaction.

LEARNING_RATE – Tweaking this parameter was a trial and error method. I found values like 1.0 too high and ones like 0.0025 too low. Finally, the learning curve evened out at 0.1, which was found to be optimal.

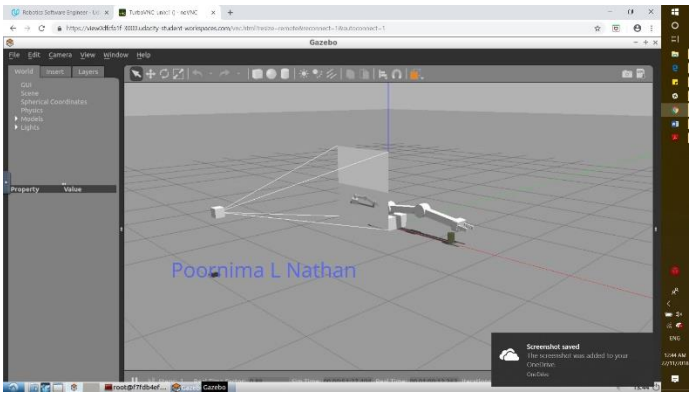
USE_LSTM – This was set to true to train the arm better. Also, the LSTM_SIZE was set to 256 since the dataset was not expected to be huge or complicated.



Task 1: Any part of the arm touching the object and accuracy > 90%



Task 2: Accuracy Details



Task 2: Any part of the arm touching the object and accuracy > 80%

Results

The second and more challenging objective required me to reduce the learning rate to 0.01. This caused the

convergence to occur a little later than the first objective. Although both the objectives were met, the second one took a bit longer due to the sharper focus of the object. The first one took 84 episodes with 91.43% accuracy and 176 episodes with 81.25% accuracy. The second took a longer since the arm seemed to be stuck in a local minimum. But once that was figured out, achieving the desired accuracy didn't take long.

VELOCITY_CONTROL – This parameter is used to define if the arm joints are to be controlled using velocity or by positions. The first used velocity control, while the second one used position control. One of the reasons is that the arm joints in the first task did not require that much of accuracy, whereas the second one need only the gripper to make contact, warranting the use of position control.

Improvements and Future Work

Working on the optimizer is the first step to improve performance better. Also, I noticed that the arm banging on the object enough to send it flying away is also considered touch. This behaviour can be tweaked to make the arm touch the object a little less softly. This project has many uses at the industrial level as well, like the robotic production line or an arm in a research lab that needs to handle hazardous or delicate items.