Reinforcement Learning to teach Robots

Navin Rahim, navin.rahim@gmail.com

Abstract—This paper discusses the deep reinforcement approach for teaching a robot to touch a can. The experiment was done as two tasks. First task was to teach the robot to touch the can with any part of its body. Second task was to teach it to touch the can with only the base of the gripper. The robot learned over time and performed both the tasks with the required accuracy.

Index Terms—Reinforcement Learning, Deep RL

1 Introduction

Teaching robots to perform tasks is a difficult job. The process of performing the task has to be programmed with many layer of complexity based on the task in hand. This takes a lot of effort and time to bring the robot to perform tasks perfectly.

Humans learn tasks by repeatedly doing them, understanding how the environment changes based on the action and tuning the way the tasks are done. This approach can be extended to robots as well. This approach is called Reinforcement Learning(RL).

With lots of data, deep learning can be attached to RL and robots can learn from its experiences and rewards. This makes developing robots faster and efficient as well.

This project focuses on implementing a deep RL method to teach a robot touch a can - with any part of its body and with the base of the gripper alone.

2 Background

The idea behind RL is to learn from experiences rather than explicit programming. A robot/agent performs senses the environment, performs an action and based on the action it took, the environment provides a reward to the robot. The objective of the agent is to maximize the cumulative reward. This makes the robot move in a positive direction where it learns how to maximize the reward and thus performing the task given correctly.

A policy determines what action a robot takes based on the environment. Thus, the job of the agent is to identify the optimal policy that maximizes the cumulative reward it obtains.

An episode is the end of a learning cycle. It can happen when the task is performed correctly or when something wrong happens. In each episode, the robot evaluates the rewards it obtained and fine tunes its policy to perform better in the future. There are also tasks that never end. Such a task is called a continuing task, and the robot has to refine its policy at every step.

2.1 Q Learning

Q learning is a reinforcement learning technique. It involves a Q table to keep track of actions, states and q-values, which are the expected rewards that can be obtained for the given action and state. Initially, the q-table contains random/zero values and the agent takes random actions. The Q-learning algorithm involves updating the q-values in an iterative manner based on the expected rewards that can be obtained in the future if a particular action is taken in a particular state.

The Q Learning technique requires various measurements from the environment as input to determine the state it is in. Real world problems involves many such measurements making Q Learning unsuitable for real world problems in general.

2.2 DQN

Deep Q Network involves using a deep neural network instead of a Q-table. Here, an image of the environment could be given as input to the RL algorithm, thus removing many of the complexities involved in the normal Q learning approach. By using the image or stack of image as input, the network outputs which action to perform. Initially, random actions are allowed to explore the action space of the agent. As time goes, the agent learns from the experiences and the model starts to give accurate actions as output.

Experience replay is a technique that improves the accuracy of the DQN. It involves storing the images that the agent saw before, so that they can be utilized again and again for learning. This also helps in dealing with environment states that occur rarely, as once it happens, they are stored and the network could learn from it again.

3 Model Training

3.1 Robot and task

The environment used for this project was simulated in Gazebo. It has a robot with 3 DoF. The robot has a gripper attached to it. A object was spawned in a fied location. Task 1 was to make the robot touch the object with any of its parts and task 2 was to make the robot touch the object using the base of the gripper.

The robot could be controlled by either increasing/decreasing the velocity of joints or directly calculating the position of the joints.

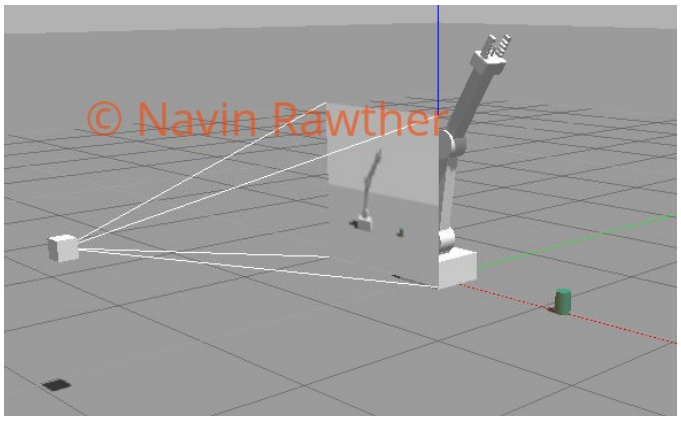


Fig. 1: Robot and object in Gazebo

3.2 Rewards

The agent had to be given rewards for the actions it perform as a feedback so that the agent can learn to perform the task well.

3.2.1 Task 1

The rewards for task 1 are as follows:

- +10 for robot hitting the object
- -10 for robot hitting the ground
- Interim reward based on the smoothed moving average of the distance to the goal
- -10 for exceeding allowed number of frames in an episode

3.2.2 Task 2

The rewards for task 2 are as follows:

- +20 for robot hitting the object with gripper base. -20 for robot hitting the object with any other part of the robot.
- -20 for robot hitting the ground
- Interim reward based on the smoothed moving average of the distance to the goal. Also a time penalty of 0.5 was added for each episode.
- -20 for exceeding allowed number of frames in an episode

3.3 Hyperparameters

The following are the hyperparameters for each of the tasks

3.3.1 Task 1

Most of the hyperparameters for task 1 was chosen arbitrarily and worked in the first attempt itself.

- VELOCITY_CONTROL : true
- INPUT_HEIGHT : 64
- INPUT_WIDTH : 64 - these values were reduced from 512 as network parameters could be reduced that makes learning faster
- OPTIMIZER : RMSprop - This optimizer gave the desired result for task 1 in the first try
- LEARNING_RATE : 0.1
- BATCH_SIZE : 32
- USE_LSTM : true - This was used to learn from multiple past frames
- LSTM_SIZE : 256

3.3.2 Task 2

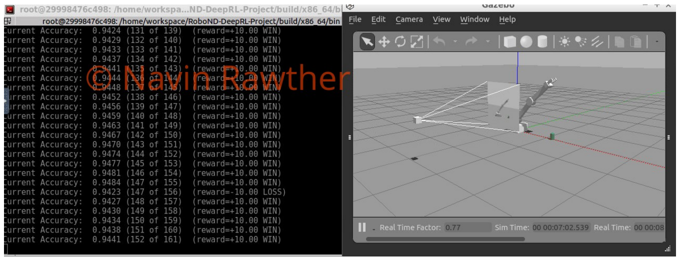
Below are the values that were tuned for task 2. Other hyperparameters were kept as same as that of task 1.

- VELOCITY_CONTROL : false - Position control was used for task 2 as velocity control with different hyperparameters did not seem to converge while position control started giving positive results straight away.
- OPTIMIZER : Adam - RMSProp kept having majority of negative results for a long duration. Adam optimizer started giving positive results and network seemed to be learning faster.

4 Results

4.1 Task 1

The target accuracy for task 1 was at least 90%. This was obtained in the 104th episode.

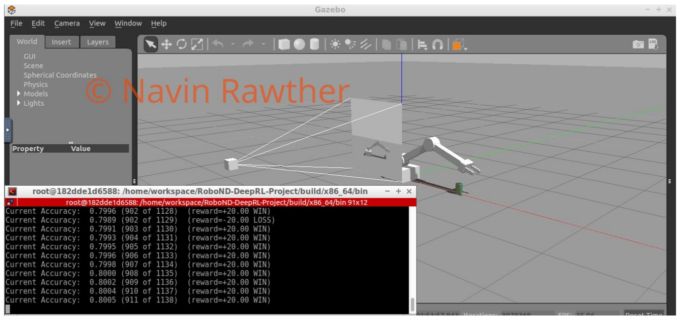


```
root@29998476c498: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin
Current Accuracy:  0.9424 (131 of 139)  (reward=+10.00 WIN)
Current Accuracy:  0.9429 (132 of 140)  (reward=+10.00 WIN)
Current Accuracy:  0.9433 (133 of 141)  (reward=+10.00 WIN)
Current Accuracy:  0.9437 (134 of 142)  (reward=+10.00 WIN)
Current Accuracy:  0.9441 (135 of 143)  (reward=+10.00 WIN)
Current Accuracy:  0.9444 (136 of 144)  (reward=+10.00 WIN)
Current Accuracy:  0.9448 (137 of 145)  (reward=+10.00 WIN)
Current Accuracy:  0.9452 (138 of 146)  (reward=+10.00 WIN)
Current Accuracy:  0.9456 (139 of 147)  (reward=+10.00 WIN)
Current Accuracy:  0.9459 (140 of 148)  (reward=+10.00 WIN)
Current Accuracy:  0.9463 (141 of 149)  (reward=+10.00 WIN)
Current Accuracy:  0.9467 (142 of 150)  (reward=+10.00 WIN)
Current Accuracy:  0.9470 (143 of 151)  (reward=+10.00 WIN)
Current Accuracy:  0.9474 (144 of 152)  (reward=+10.00 WIN)
Current Accuracy:  0.9477 (145 of 153)  (reward=+10.00 WIN)
Current Accuracy:  0.9481 (146 of 154)  (reward=+10.00 WIN)
Current Accuracy:  0.9484 (147 of 155)  (reward=+10.00 WIN)
Current Accuracy:  0.9423 (147 of 156)  (reward=-10.00 LOSS)
Current Accuracy:  0.9427 (148 of 157)  (reward=+10.00 WIN)
Current Accuracy:  0.9430 (149 of 158)  (reward=+10.00 WIN)
Current Accuracy:  0.9434 (150 of 159)  (reward=+10.00 WIN)
Current Accuracy:  0.9438 (151 of 160)  (reward=+10.00 WIN)
Current Accuracy:  0.9441 (152 of 161)  (reward=+10.00 WIN)
```

Fig. 2: Task 1 result

4.2 Task 2

The target accuracy for task 1 was at least 80%. This was obtained in the 1135th episode.



```
root@182dde1d6588: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 91x12
Current Accuracy:  0.7996 (902 of 1128)  (reward=+20.00 WIN)
Current Accuracy:  0.7989 (902 of 1129)  (reward=-20.00 LOSS)
Current Accuracy:  0.7991 (903 of 1130)  (reward=+20.00 WIN)
Current Accuracy:  0.7993 (904 of 1131)  (reward=+20.00 WIN)
Current Accuracy:  0.7995 (905 of 1132)  (reward=+20.00 WIN)
Current Accuracy:  0.7996 (906 of 1133)  (reward=+20.00 WIN)
Current Accuracy:  0.7998 (907 of 1134)  (reward=+20.00 WIN)
Current Accuracy:  0.8000 (908 of 1135)  (reward=+20.00 WIN)
Current Accuracy:  0.8002 (909 of 1136)  (reward=+20.00 WIN)
Current Accuracy:  0.8004 (910 of 1137)  (reward=+20.00 WIN)
Current Accuracy:  0.8005 (911 of 1138)  (reward=+20.00 WIN)
```

Fig. 3: Task 2 result

5 Discussion

The training for task 1 was relatively easy. The chosen hyperparameters provided good results. The 90% mark was reached in the 104th episode and kept increasing.

The training for task 2 involved a lot of hyperparameter tuning. Although the results seemed positive initially, the robot started to just wait near the object causing end of episode or touch the ground. It seemed that the robot was happy in collecting the interim rewards. Thus rewards were also modified. Also a time penalty was added. This greatly helped in getting more positive results frequently.

6 Future work

The obtained results could further be improved. For task 1, the accuracy kept increasing. Thus, training it for more episodes could get a high accurate model. For task 2, it took a lot of episodes to reach the 80% mark. The model and reward system could be further fine tuned for faster convergence. Also, longer training will produce high accuracy.

The object could be moved randomly and the robot could be trained for touching it. Later, this model could be tuned to be used for different tasks such as pick and place of objects.