Deep RL Arm Manipulation

Siddharth Kothiyal

Abstract—As babies, humans learn how to perform a task by seeing others and trying to mimic their actions. Learning is not a simple process, for learning something as basic as standing up, we take countless attempts, trying different ways to push ourselves up, and find the perfect balance. In the process, we fall multiple times, but with each fall we gain some experience.

Humans have tried to come up with algorithms to teach a move to a robot. But we soon realized that we need to transfer the ability of gaining experience to robots, the ability to learn from our mistakes (punish when we do something wrong) and get rewarded for performing an action properly. This project aims to teach a robotic arm how to touch/grab an object on the ground, without touching the ground.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Mapping.

1 Introduction

THIS project aims to teach a robotic arm how to touch an object on the ground, without the arm touching the ground it self. This feat is achieved by using Deep Q-Networks, a type of Deep Reinforcement Learning technique. Where the agent is rewarded for every time it successfully completes the given task and gets a negative reward, when it fails to do so.

The project is divided into two parts, touching the object with any part of the arm, and touching the object with just the gripper_base. For both the parts, almost similar hyperparameters and reward functions are used.

2 Reward Function

In each attempt, the robotic arm starts from a defined initial position, the arm is given 100 moves to move around and find the object and touch it. If it successfully does it, the learning agent is rewarded with REWARD_WIN (+500) score, but if the arm is unsuccessful in completing the task within 100 moves, the arm will be reset to its initial position, and the next attempt will start. At the same time, the learning agent is rewarded with a negative score REWARD_LOSS (-500) score. If in the process, the robotic arm touches the ground, it loses and is awarded the negative reward of REWARD_LOSS (-500), and the scene is reset.

During each attempt, the learning agent is also rewarded for each move it makes, this is known as the interim reward, this is used to guide the arm in the right direction. This interim reward is calculated on the basis of how much closer the arm has gotten to the object as compared to its movement in the last move. To do so, the distance between the object and the arm is found, and it is compared to how much it moved closer in the last attempt. The exact formula is given below:

The distance calculation is then multiplied to a REWARD_INTER (+400) score, to normalize the distance calculation and make it significant in from of the REWARD_WIN and REWARD_LOSS score.

Reward function used for both the cases are same.

```
if(!checkGroundContact)
{
    const float distGoal = BoxDistance(gripBBox,propBBox);

    if(DEBUG){printf("distance('%s', '%s') = %f\n", gripper->GetName().c_str(), prop->model->GetName().c_str(), distGoal);}


    if( episodeFrames > 1 )
    {
        const float distDelta  = (lastGoalDistance - distGoal)/distGoal;
        float alpha = 0.2;
        // compute the smoothed moving average of the delta of the distance to the goal
        avgGoalDelta  = (avgGoalDelta * alpha) + (distDelta * (1.0f - alpha));
        rewardHistory = avgGoalDelta * REWARD_INTER;
        //printf("distDelta %f, avgGoalDelta %f, rewardHistory %f \n", distDelta, avgGoalDelta, rewardHistory);
        newReward     = true;
    }

    lastGoalDistance = distGoal;
}
```

Fig. 1. Function to calculate interim rewards

3 Hyperparameters

The input size for the DQN were decreased to 64x64, as both the object and arm were fitting inside those image parameters. RMSprop and Adam were experimented as the optimizers for the DQN, and RMSprop seemed to be performing better. LSTM was enabled and the size of LSTM was increased so that the DQN agent is able to remember the sequence of moves and its corresponding results. BATCH_SIZE was also increased to 64

Varying the learning rate did not show much improvement in the overall training time.

```
[deepRL]  use_cuda:       True
[deepRL]  use_lstm:       1
[deepRL]  lstm_size:      128
[deepRL]  input_width:    64
[deepRL]  input_height:   64
[deepRL]  input_channels: 3
[deepRL]  num_actions:    6
[deepRL]  optimizer:      RMSprop
[deepRL]  learning rate:  0.2
[deepRL]  replay_memory:  10000
[deepRL]  batch_size:     64
[deepRL]  gamma:          0.9
[deepRL]  epsilon_start:  0.9
[deepRL]  epsilon_end:    0.05
[deepRL]  epsilon_decay:  200.0
[deepRL]  allow_random:   1
[deepRL]  debug_mode:     0
[deepRL]  creating DQN model instance
[deepRL]  DRQN::__init__()
[deepRL]  LSTM (hx, cx) size = 128
[deepRL]  DQN model instance created
[deepRL]  DQN script done init
[cuda]  cudaAllocMapped 49152 bytes, CPU 0x2049a0000 GPU 0x2049a0000
[deepRL]  pyTorch THCState  0x28E75C10
[cuda]  cudaAllocMapped 12288 bytes, CPU 0x204aa0000 GPU 0x204aa0000
```

Fig. 2. Hyperparameters for the DQN

4 Results

The process started with varying the hyperparameters, after several iterations of trial and error, the best result was the arm touching the object only twice or thrice in first 100 attempts. The agent seemed to had found a way, in which it was able to maximize the rewards, through interim rewards, without actually touching the object. It would always end up undershooting or overshooting the object.

After randomization in the spawning position of the object was included, the results started showing exponential jump, even with parameters not tuned properly, the arm was able to touch the object about 50-60 times in first 100 attempts. Hyperparameters were then tuned accordingly, to maximize the results.

Changing randomization limits also showed a very interesting pattern. When the minimum and maximum limits of the object spawning were changed the accuracy and the corresponding training time would change accordingly. The training time would be least when the object spawns were in the sweet spot, ”0.05 to 0.40” along x-axis, and ”0.05 to -0.05” along y-axis (this was not varied a lot since the base was locked and not able to rotate). Increasing the spawn limits would increase training time, and sometimes even lead to the arm completely failing to touch the object.

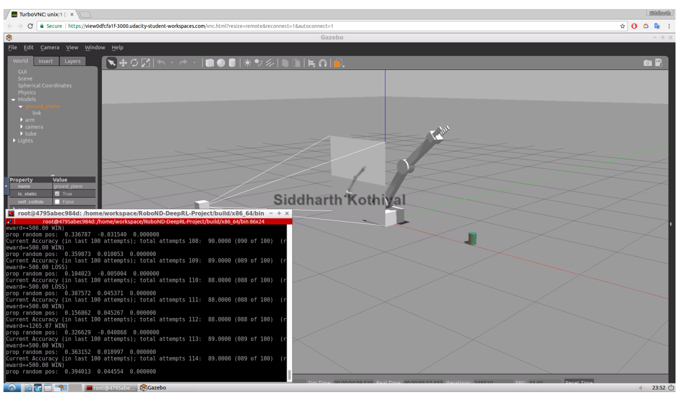

Fig. 3. Touching objet with Gripper base

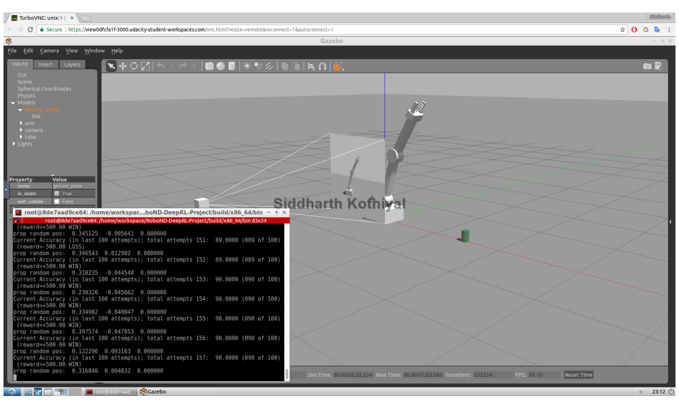

Fig. 4. Touching object with Arm

5 Conclusion / Future work

Enabling Velocity_control was promising, but the overall time taken for the arm to touch the object for the first time increased significantly. But the movements of the arm showed that it could be a much more versatile solution and would be able to grasp objects in a wider range.

Another interesting approach could be removing the lockbase, so that even the base is able to rotate on its axis. This would allow the arm to move in y-axis and hence y-axis randomization parameters could also be explored.

The way interim rewards are calculated could be changed for the case when the whole arm was learning how to touch the object. Instead of using the same distance function, where distance from gripper is used, minimum distance from any part of the arm could be used.