Deep RL Arm Manipulation

Yisi Zhang

Abstract—The deep reinforcement learning project for arm manipulation aims to reliably touch the object with any part or the gripper of the robotic arm. A camera sensor captures the side view of the robotic arm with the object, from which the images are fed into the Deep Q-Network (DQN) for learning. The robotic arm has a degree of freedom (DOF) of 3, with possible up and down movements for each joint. Different rewards are assigned to different outcomes of the movements. Within reasonable amount of trials, the robot learns to touch the object with $>$90% accuracy using any part of the arm and with $>$80% accuracy using the gripper.

Index Terms—Deep RL, Robotic arm, C++ API, Gazebo, LSTM.

1 Introduction

Deep reinforcement learning is used to teach robots to learn tasks in 2D and 3D worlds. It passes the image data to a deep neural network to produce a Q value for every possible action. The images are first passed through a couple of convolutional layers to extract the spatial relationships and a brief temporal properties across the small stack of frames. They are then processed by a fully-connected hidden layer and a fully connected output layer to produce the action values. In this way, the representation of the state is encoded in the deep network.

2 Methods and Parameters

The reward functions are implemented in the ArmPlugin.cpp file. To realize the control task, the image of the arm and the object as well as the collision information are needed to update the neural network and to assign proper rewards to the actions. Thus, the camera data and the collision message from Gazebo are subscribed.

There are two control modes through which the state of the robotic arm is updated: velocity control and position control. In this project, position control was used in both tasks. The number of possible actions is twice the degree of freedom (DOF). As there are three non-fixed joints, the DOF is 3 and thus the number of actions is 6. There is a velocity and a position value for each of the 3 joints. The even numbers of the action correspond to an increase of velocityposition by Delta amount, while the odd numbers correspond to a decrease by Delta amount.

Collision is checked by detecting if contact #1 is the COLLISION_ITEM, which is the tube. If this is the case, the first objective, which is to touch the object by any part of the arm, is achieved. For the first task, the reward of this achievement (by updating the rewardHistory variable) is set to 1. To accomplish the second task, the collision is further checked whether the second contact is the COLLISION_POINT, which is the gripper. If the object is touched by the gripper, a reward of 2.5 is granted; if it is touched by other parts of the arm, a penalty of -0.5 is given to the rewardHistory. If the gripper instead hits the ground, a penalty of -1 is assigned to the rewardHistory. Ground collision is detected by whether the bottom of the gripBBox is less than a threshold height of 0.05. When a collision is detected, the newReward and endEpisode variables are set to true. In addition, an interim reward is given according to the change of the distance to the goal made by each action. If the action results in an decrease in the distance between the gripper and the object, the rewardHistory is updated with REWARD_WIN * 2 (=0.2); if the action moves the gripper farther from the goal, a penalty of REWARD_LOSS (=-0.1) multiplied by the distance is given to the rewardHistory. The distance is computed as smoothed moving average of the change of distance to the goal with the smoothing factor set to 0.75. The distance to the goal is computed by the BoxDistance function between the gripBBox and the propBBox. If the arm state is stuck, i.e., the change of the delta of the distance to the goal changes less than a threshold (of 0.05), the rewardHistory cumulates the penalty every iterate by an amount of REWARD_LOSS. If no collision is detected within 100 frames, a REWARD_LOSS is given to the rewardHistory and the episode ends.

The choices of the hyperparameters for the two objectives are shown in Tabel 1.

TABLE 1
Hyperparameters

	Objective 1	Objective 2
INPUT_WIDTH	64	64
INPUT_HEIGHT	64	64
OPTIMIZER	“Adam”	“Adam”
LEARNING_RATE	0.1	0.01
REPLAY_MEMORY	1000	10000
BATCH_SIZE	512	512
USE_LSTM	true	true
LSTM_SIZE	256	256
REWARD_WIN	0.1	0.1
REWARD_LOSS	-0.1	-0.1
alpha	0.75	0.75

The size of the images was set by trial and error. The algorithm performed poorly with 32x32 image size. Two optimizers were tried: ”RMSProp” and ”Adam”. The latter performed significantly better than the former. A larger learning rate of 0.1 for the first objective converged much faster; while this value was too large to achieve the second objective as the learning did not converge well with good fidelity. The REPLAY_MEMORY, which defines the number of actions to take before the learning phase, does not require a large number for the first task as the probability for any part of the arm to hit the ground is higher than just the gripper. Thus, effective learning could occur more frequently in the first case and the replay memory size was accordingly determined. The setup of the BATCH_SIZE was through guessing. The long-short-term memory (LSTM) was used to allow for long term dependencies in both tasks and it is crucial. The size of the LSTM was set to 256.

3 Results

1) Objective #1: Any part of the robot arm touches the object with at least an accuracy of 90%.
   We demonstrate that the robotic arm is trained to touch the object with >90% accuracy within approximately 500 iterations(Fig. 1, 2).
2) Objective #2: Only the gripper base of the robot arm touches the object with at least an accuracy of 80
   We demonstrate that the robotic arm gripper is trained to touch the object with >80% accuracy within approximately 150 iterations(Fig. 3, 4).

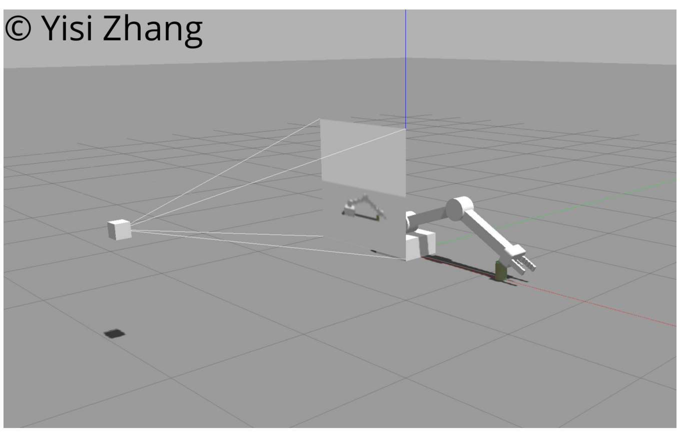

Fig. 1. Screen shot of the arm touching the object

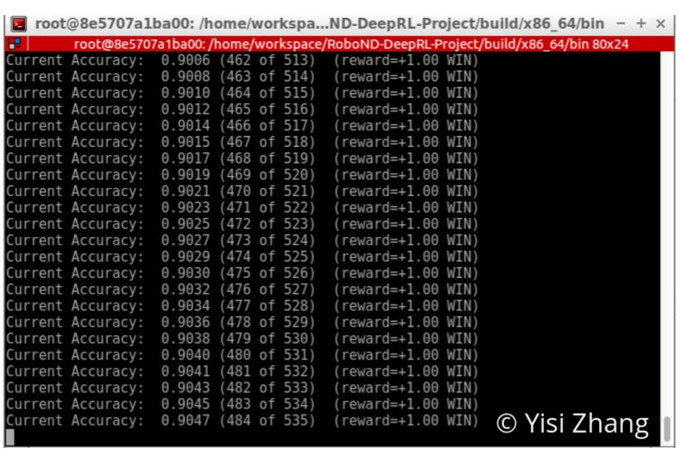

Fig. 2. Accuracy achievement of Objective 1

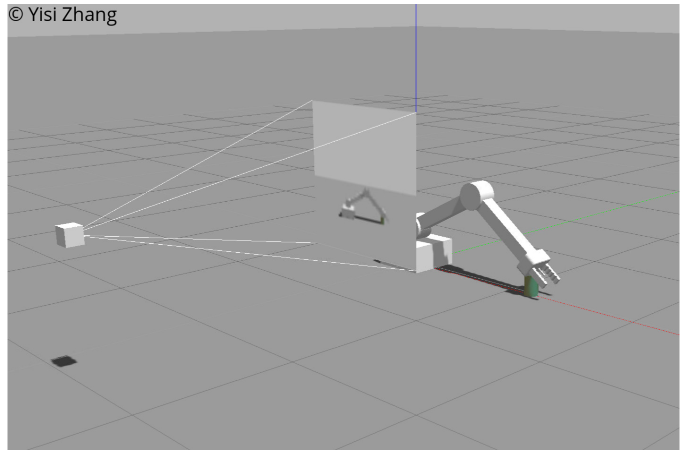

Fig. 3. Screen shot of the gripper touching the object

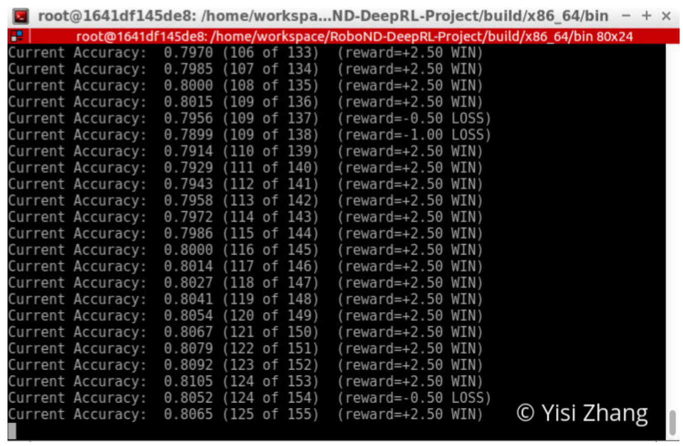

Fig. 4. Accuracy achievement of Objective 2

4 Discussion

It took a large number of iterations for the robotic arm to accomplish the first objective. The reason is that it was stuck at a local minimum when it reached around 60% accuracy. This is because as some part of the arm hits or is close to hit the goal, the gripper reaches the ground first and causes the failure. This subtlety took longer to train. One way to avoid this is to add penalization for the closeness of the lowest part of the arm to the ground if it deviates from the goal. Despite the lack of this constraint, the robot still learned to avoid such circumstances given enough iterations, demonstrating the powerfulness of the RL algorithm. The second objective did not have this problem because the gripper is usually the lowest part of the arm and it is trained to reach the goal instead of the ground.

Given more time, it is worth experimenting with the parameters including the learning rate, the batch size, the differential rewards and the smoothing factor alpha to further optimize the learning.