Deep RL Arm Manipulation

Seyfi Gozubuyuk

Abstract—In this project, the aim is to create a DQN agent and define reward functions to teach a robotic arm to touch the object. The project uses jetson-utils and jetson-reinforcement libraries and the Gazebo simulation environment.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Reinforcement Learning, RL, DEEP RL, DQN, jetson.

1 Introduction

The problem is to train a DQN agent to enable a robotic arm to do the following tasks

1) Touch the object with any part of the robot arm, with 90% accuracy.
2) Touch the object with the gripper base of the robot arm, with 80%accuracy.

The jetson jetson-utils and jetson-reinforcement libraries are used to realize this project [1]. The Gazebo is the simulation environment for the robotic arm. There is a tube object, and the robot should learn to reach to this object.
The project defines several reward functions and tunes the parameters of the libraries.

2 Background

The problem is to build a QDN agent. The agent uses image data and reward values as input and outputs the action best action to perform.

2.1 Reinforcement Learning

In reinforcement learning (RL), the software agents learn the best actions in an environment to maximize the reward. The rewards should be defined for the agents. They start learning with trial and error. As they learn, they can take logical actions [2].

2.2 Q-learning

In Q-learning, the agent learns a policy which contains best actions in the different states of the environment [3]. It is a model-free algorithm and does not need to know the every detail of the environment. It aims to build a table to store rewards for each state and action pair. At every time step, the agent observes it state and choose an action. Then it updates the reward value of that state, action pair.

2.3 Deep Reinforcement Learning

Deep reinforcement learning (DRL) is a special kind of reinforcement learning that uses artificial neural networks to train the agent. It replaces the processing sensor measurements step with an end-to-end approach. The network directly receive the sensor measurements and returns the best action to perform [4].

2.4 Deep Q Network

DQN trains an NN for the Q value function. It uses the following techniques to let the network converge faster and more reliably.

- The replay pool takes the average of the behavior distribution over previous states. It is essential for smoothing the learning and handling the noise. It provides the advantage of using every step more than once in the weight update process.
- The target network represents the Q function that is the function to compare when calculating the loss. Since the Q function values are changing during the training, it is required to use another network.

3 Simulations

The simulation environment is in Gazebo. There exist two nodes. The camera node and the collision node. The agent should subscribe to the topics of these nodes. The agent is an instance of the dqnAgent provided by the jetson-reinforcement library.
There were two options for controlling the arm joints. The first one is the velocity control. In the beginning, it was the preferred approach; however, the agent using this approach was not successful. One issue with this controller is, it requires an additional action for not changing any velocities. The other method was the position control. This approach directly changes the position of the joint.

3.1 Rewards

There are three rewards in general. They can be listed as follows.

- Reward for the successful end of episode
- Reward for the unsuccessful end of episode
  - Hit to the ground
  - End of episode without success
  - Touch the object with other parts than gripper (Only for task 2)
- Reward Delta, based on distance for each episode.

There are two parameters for these rewards. They are REWARD_WIN and REWARD_LOSS, and they were set to 100 and -100 respectively, in order to increase the difference

between the end of episode rewards and episode rewards. However, there are multipliers for these parameters.
There are five different distance functions. The first one existed in both tasks, but the others were explored in the second task. The first function, BoxDistance, calculates the distance between the nearest points of two boxes, the gripper and the tube.

3.2 Hyperparameters

The hyperparameters used in both tasks are given in Table 1. Since the hardware that the project run on has limited resources, it was required to decrease the complexity of the neural network.
Input size, the input width, and the input height parameters were set to 64. Previously tried values were 512, 256, 128 and all of them caused the out of memory error for the GPU, before 100 episodes. Besides, the camera image size was 64x64, and 64 produced satisfactory results.
There were two different optimizers, RMSprop and ADAM, which work similarly. RMSprop tended to produce better results during the project. RMSprop helps the gradient descent to reach the minima faster.
Learning rate parameters that applied during the first task were 0.005, 0.01, 0.02, 0.05. As the reward functions define successfully, the higher learning rate helped much. As a result, the robot failed only in the first episode from 102 episodes.
For the second task, it was much harder to come up with well defining reward functions. Therefore, the robot did not perform well in the first episodes. And this led the agent to be biased. Again lower parameters such as 0.005 and 0.01 were applied. However, they did not provide satisfying results. The agent worked best with a learning rate of 0.02 for the second task.
The batch size value was eight in the project repository, and it wasn't changed for the first task. In the second task, it was increased to 16. Higher batch size help network to learn better; however, raising it would cause computational overhead.
Replay memory parameter left unchanged at 10000. Higher values would speed up the learning process on the other hand increase the memory requirements.
Use of LSTM would enhance the learning as it uses two memories for long-term and short-term. LSTM size decreased to 16 for the first task to raise the computational performance, since using larger LSTM size values would increase the memory requirements.
The number of actions, there were three joints, and the degree of freedom was 3. There were two actions for each joint. Rotate on in one direction or the opposite direction. For the velocity controller, the actions were changing the velocities. Therefore, if velocity were not zero the joint would be still rotating. It was experimented to add another action in which no joints accelerate or decelerate. This new action did not help much but increased the complexity.

3.3 Task 1

The reward for the successful end of episode was set 2 times REWARD_WIN minus the episodeFrames property of the ArmPlugin class. The reason was to produce a value which was always higher than the REWARD_WIN parameter and to penalize the increasing number of steps.
The reward for hit the ground and episode exceeds the maximum number of frames had the value of the REWARD_LOSS parameter. This were good enough to penalize the end of episode without success.
The distance reward that was calculated in each episode had the value of the REWARD_WIN parameter multiplied by the moving average of the delta of the distance to the goal. The other parameter was the alpha which was equal to 0.25. The reason for keeping alpha small is to increase the effect of the last delta values.

TABLE 1
Hyperparameters

Parameter	Task 1	Task2
INPUT_WIDTH	64	64
INPUT_HEIGHT	64	64
OPTIMIZER	RMSprop	RMSprop
LEARNING_RATE	0.05	0.02
REPLAY_MEMORY	10000	10000
BATCH_SIZE	8	16
USE_LSTM	true	true
LSTM_SIZE	16	16
NUM_ACTIONS	6	6

3.4 Task 2

The robot arm from the previous task only aimed to touch the tube no matter which part reaches before. The first thing to try was adding a penalty for robot parts other than the gripper touches the object. However, this does not help the agent to change its behavior. Therefore, this project investigated several reward functions.
The changes applied to the reward functions are as follows.

- Reward for the successful end of episode
  - Collision with arm gripper middle had a reward multiplier of 16 (16 * REWARD_WIN - episodeFrames)
  - Collision with arm gripper base had a reward multiplier of 8 (8 * REWARD_WIN - episodeFrames)
- Reward for the unsuccessful end of episode
  - Hit to the ground has a multiplier 1.8 (1.8 * REWARD_LOSS)
  - End of episode without success (REWARD_LOSS)
  - Touch the object with other parts than gripper (REWARD_LOSS)
- Reward based on distance for each episode.
  - Reward Delta from the previous task divided by 2
  - Add a negative reward of Manhattan distance between the centers of two boxes
  - Add the distance between min z of the gripper and the ground

The success rewards were increased to encourage the robot to learn the task. Since it was better to touch the object with the gripper middle, it has a higher multiplier. And it was still important to reach the goal in early steps.
The penalty for hitting the ground was increased because the robot was inclined to touch the ground with the gripper before moving forward on the x-axis. Raising the threshold caused the robot to move along the x-axis without turning the joint 2. The Reward Delta was causing the same problem; therefore, it was divided with 2.
There was a need for other rewards changing on each frame because the end of episode rewards were not adequate. One of the additional rewards had the value of the distance between the gripper and the tube. This distance was between the centers of the boxes, and it was a Manhattan distance. It has no multipliers. The other reward was based on the minimum z value of the gripper. It was added as a reward if the gripper intersects the hyperplane of the tube extended along the z-axis. Otherwise, as a penalty, if the gripper is close to the ground. If either min.x or max.x value of the gripper was between min.x and max.x of the tube, and the same applied for the y-axis, then the gripper was assumed to intersect the hyperplane. In this case, the REWARD_WIN divided by the minimum z value of the gripper to achieve the reward value. Otherwise, the minimum z value was compared with eight times the ground contact value. In other words, if the minimum z value was less than 0.4, a penalty was added in the same manner.
During the second task, additional cost functions were created. These are as follows.

- Manhattan distance version of the box distance
- Center distance, finds the centers of the boxes and calculates the euclidean distance between them
- Manhattan distance version of the center distance
- Hit distance, calculates the nearest Manhattan distance for collision assuming the robot will reach from the upper left corner of the tube.

4 Results

The ArmPlugin.cpp file was able to utilize the jetson-reinforcement framework and to teach a robot arm to successfully reach the object. The previous sections describe the reward functions, hyperparameters and this section includes the results and screenshots.

4.1 Task 1

Figures 1, 2, 3, 4, 5, and 6 shows the screenshots of a successful run for Task 1. The console output can be viewed from Task1.txt document under the docs folder of the project repository [5]. Figure 7 show a screenshot. The accuracy summary also exists in the same folder [6]. Figure 8 show a screenshot.

4.2 Task 2

Figures 9, 10, 11, 12, 13, and 14 shows the screenshots of a successful run for Task 2. The console output can be viewed from Task2.txt document under the docs folder of the project repository [7]. Figure 15 show a screenshot. The accuracy summary also exists in the same folder [8]. Figure 16 show a screenshot.

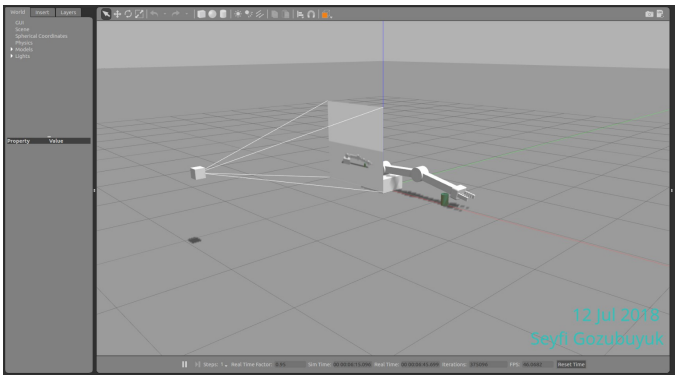

Fig. 1. Task 1 Episode 84 Gazebo

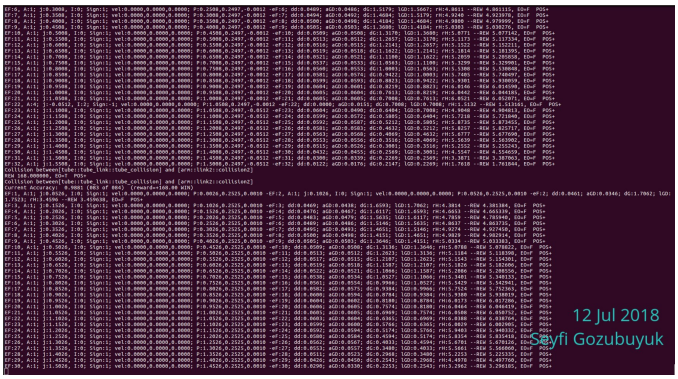

Fig. 2. Task 1 Episode 84 Console

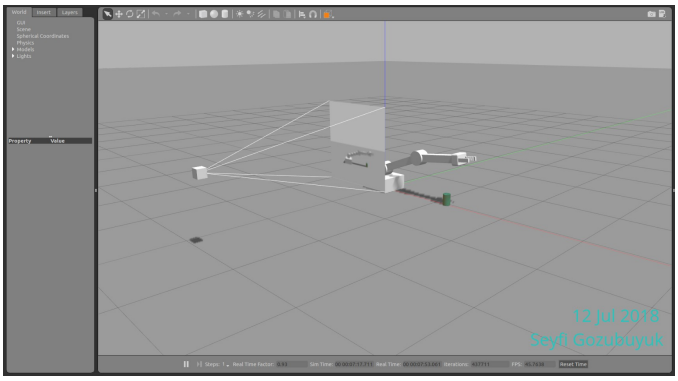

Fig. 3. Task 1 Episode 99 Gazebo

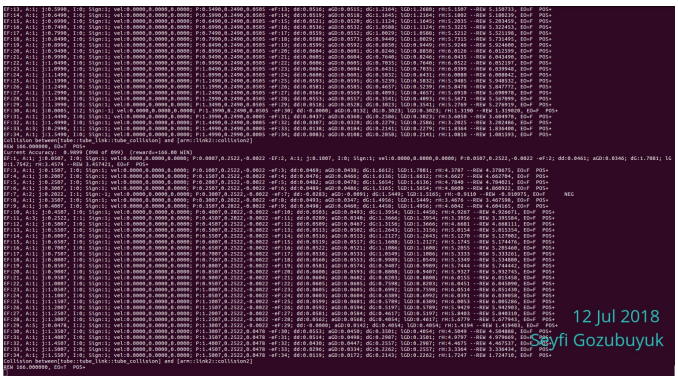

Fig. 4. Task 1 Episode 99 Console

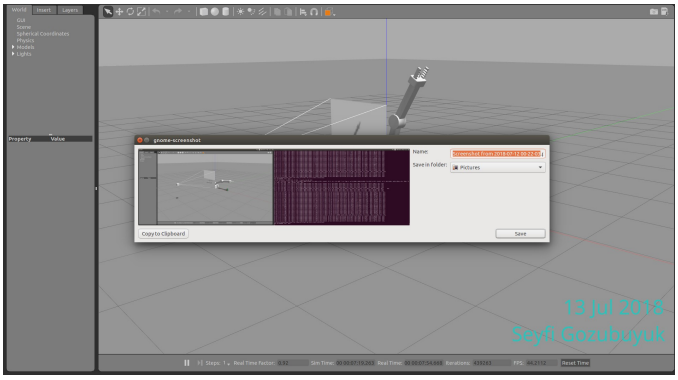

Fig. 5. Task 1 Episode 100 Gazebo

Fig. 6. Task 1 Episode 100 Console

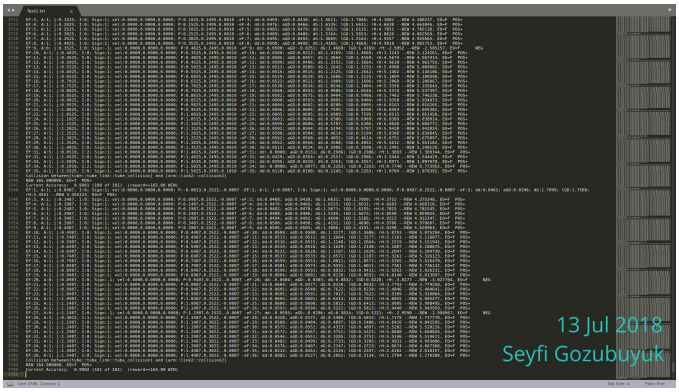

Fig. 7. Task 1 End Console Output

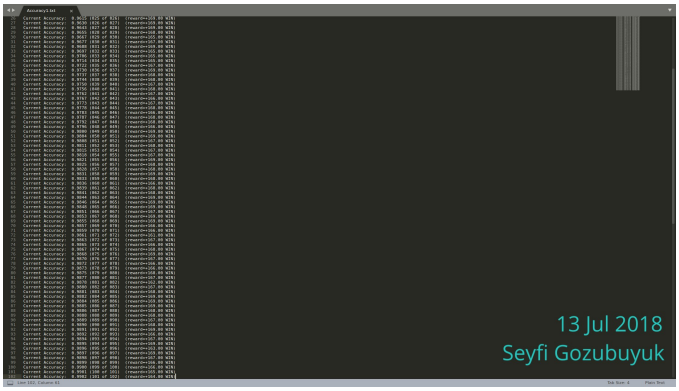

Fig. 8. Task 1 Accuracy Summary

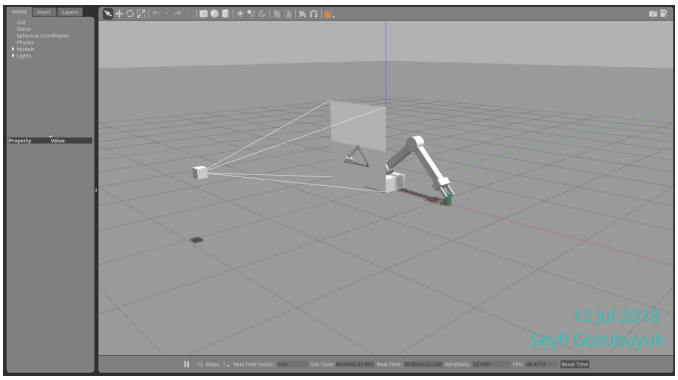

Fig. 9. Task 2 Episode 55 Gazebo

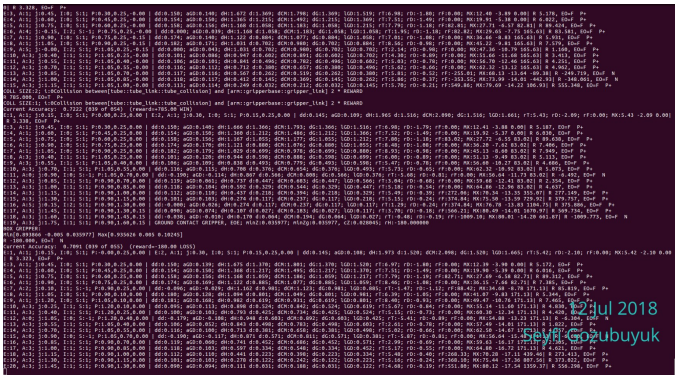

Fig. 10. Task 2 Episode 55 Console

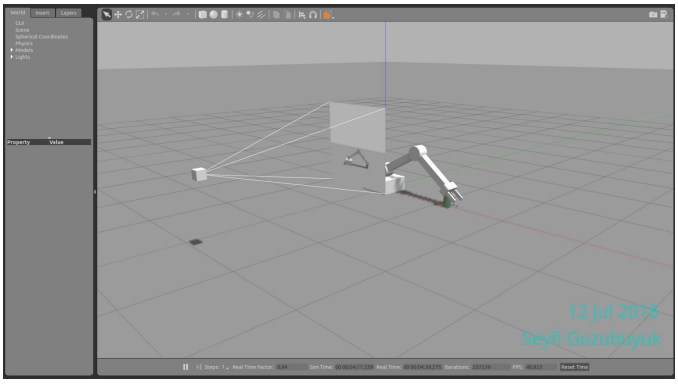

Fig. 11. Task 2 Episode 94 Gazebo

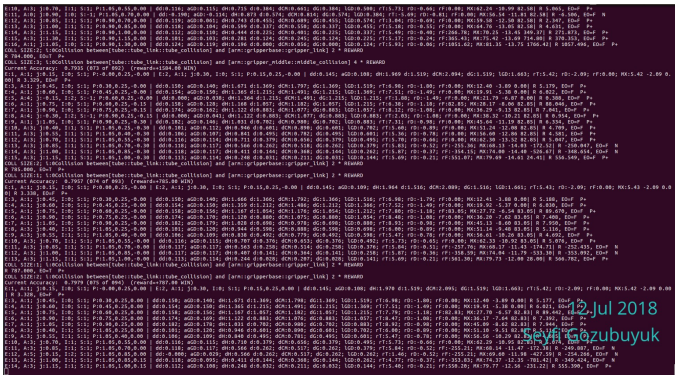

Fig. 12. Task 2 Episode 94 Console

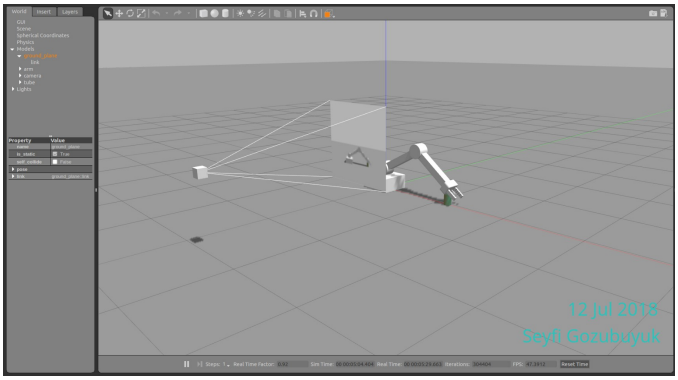

Fig. 13. Task 2 Episode 113 Gazebo

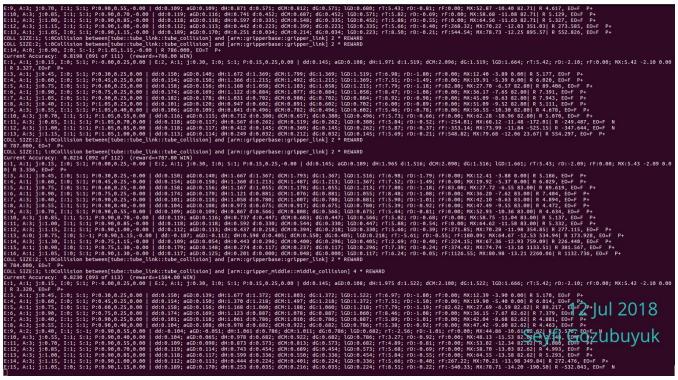

Fig. 14. Task 2 Episode 113 Console

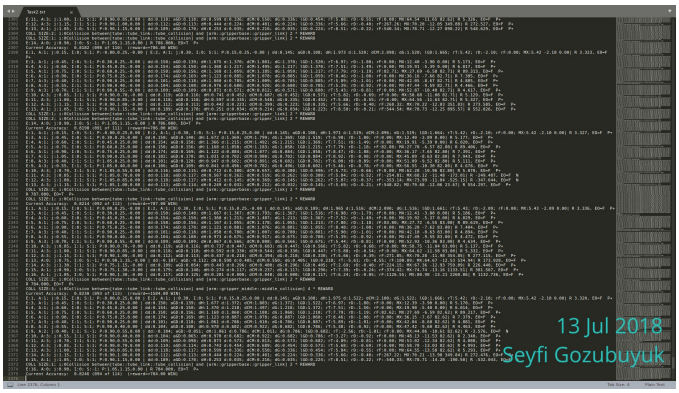

Fig. 15. Task 2 End Console Output

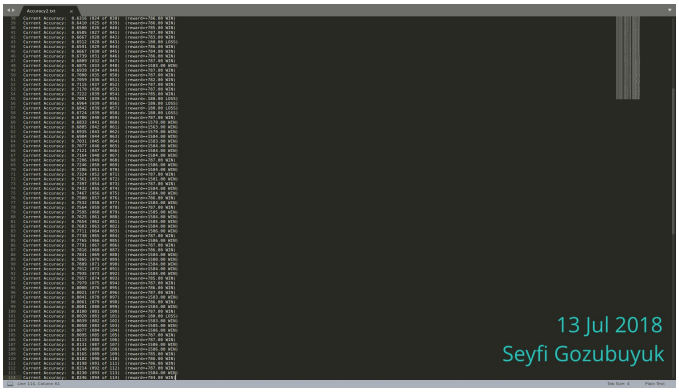

Fig. 16. Task 2 Accuracy Summary

4.3 Technical Comparison

The accuracy graph for both of the tasks are given in the Figures 17, 18. Red line and points show the Reward value. For Task 2, the values are divided by 10 on the graph. The Yellow line and points show 100 for win and 0 for loss. Blue line shows the accuracy as the percentage value.

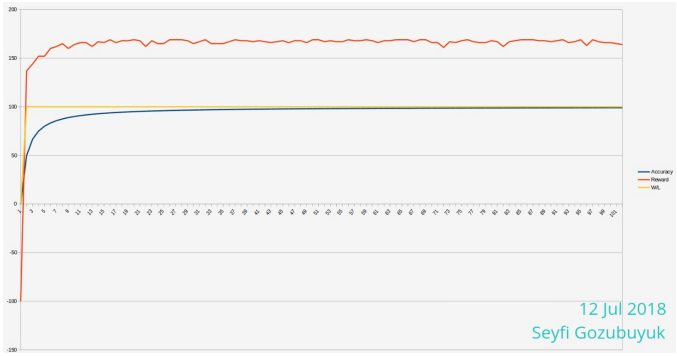

Fig. 17. Task 1 Result Graph

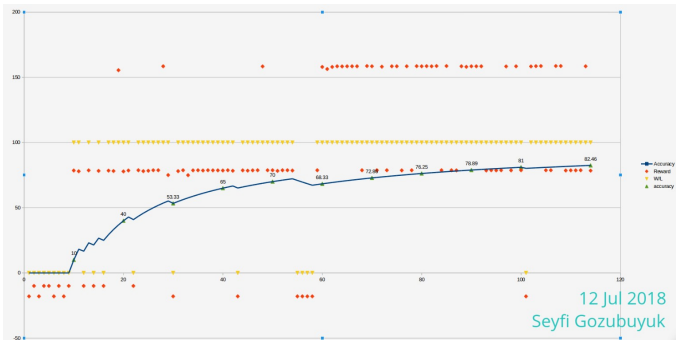

Fig. 18. Task 2 Result Graph

5 Discussion

The algorithm was able to teach the agent for the objective, and the robot arm was able to reach the object which was always in the same location. However, there were limitations on the agent's progress. The results were not good enough for a problem where the object location changes randomly.

The first task has an accuracy rate of 99.02%. The robot only failed in the first one of the 102 episodes. This performance is well enough; however, the outcomes should be transferred to harder problems.

The second task lasted 114 episodes. The robot arm reached to target from 94 of them. The accuracy was 82.46%. The success rate was improving with each trial. Testing for more episodes would probably result in higher accuracy, but the run was terminated to prevent out of memory error. The accuracy graph shows that the agent has only one failure in the last 56 episodes, which will reach to an accuracy of 98.21%.

On the other hand, the robot had failed in all of the first nine episodes. This shows that the reward functions are not good enough. The agent failed four episodes consecutively between the episodes 56 and 59. All of them caused by

hitting the ground. Consequently, there is room for improving the ground contact gripper failure. As mentioned in the rewards section, increasing the penalty for gripper touching the ground decreased the fraction of successful episodes. The maximum frames reached in an episode was 71, and it again resulted in ground contact.
Table 2 shows the episode results for Task 2 distributed over reasons.

TABLE 2
Task 2 Results

Reason	Loss	Win
100 Frames	0	0
Hit Ground	11	0
Link Touch	9	0
Gripper Base	0	58
Gripper Middle	0	36
Total	20	94

Besides improving the reward functions, enhancing the capacity of the neural network would help to achieve better results. Increasing the batch size, replay memory, LSTM size parameters are promising to obtain more successful outcomes.

6 Conclusion / Future work

The parameters and the rewards can be improved. Besides, the algorithm may enable the robot to grasp the object. The agent should be trained to access objects that located randomly. First, only on the x-axis, then both on the x-axis and y-axis.
Testing the algorithm on the Jetson TX2 and comparing the results with the current results is a waiting task for the future work. Upon receiving successful results, the algorithms in the project can be used on a real robot

6.1 Hardware Deployment

The project was deployed in the simulation environment. The machine has an I7 CPU, 8GB RAM, and 650M GPU. The configuration was able to run the project, and with more powerful CPU and GPU, it will be possible to increase the complexity of the environment. Using a Jetson TX2 would enhance the performance of this project.

References

[1] dusty-nv, "jetson-reinforcement," 2018. [Online; accessed 12-July-2018].
[2] Reinforcement learning, "Reinforcement learning — Wikipedia, the free encyclopedia," 2018. [Online; accessed 12-July-2018].
[3] Q-learning, "Q-learning — Wikipedia, the free encyclopedia," 2018. [Online; accessed 12-July-2018].
[4] Deep Reinforcement learning, "Deep reinforcement learning — Wikipedia, the free encyclopedia," 2018. [Online; accessed 12-July-2018].
[5] Seyfi Gozubuyuk, "Task 1 console output," 2018. [Online; accessed 12-July-2018].
[6] Seyfi Gozubuyuk, "Task 1 accuracy summary," 2018. [Online; accessed 12-July-2018].
[7] Seyfi Gozubuyuk, "Task 2 console output," 2018. [Online; accessed 12-July-2018].
[8] Seyfi Gozubuyuk, "Task 2 accuracy summary," 2018. [Online; accessed 12-July-2018].