Deep Reinforcement Learning for Arm Manipulation

Christian Rabus

Abstract—This report introduces Deep Reinforcement Learning based on a 2 (3) DOF robotic arm. The robotic arm is simulated in Gazebo and a DQN (Deep Q-Learning Network) algorithm is used to train a neural network and send actions to the arm for controlling the joints. The objectives of the underlying project is, to define reward functions for the DQN. First objective is that any part of the robot arm touches the object of interest, with at least a 90% accuracy and the second objective that only the gripper base of the robot arm touches the object, with at least a 80% accuracy.

Index Terms—Gazebo, Robotic Arm, DQN, PyTorch, Deep Q-Learning Network, Neural Networks, Deep Reinforcement Learning.

1 Introduction

This report focuses on Deep Reinforcement Learning, demonstrated using the example of a 2 (3) DOF robotic arm. The main goal of the underlying project is to create a DQN agent and define reward functions to teach the robotic arm to carry out two primary objectives:

- **Task 1**: any part of the robot arm shall touch the object of interest, with at least a 90% accuracy
- **Task 2**: only the gripper base of the robot arm shall touch the object, with at least a 80% accuracy

The robot arm is simulated in Gazebo, as is the object to be touched, a cylindrical tube which is placed upright and within reach of the robot arm. The robot arm has 3 rotational degrees of freedom: The base's revolute joint can swing the mounted arm in both directions by 45 degrees to the left and right. The mounted arm has 2 more revolute joints. The first one (axis of rotation horizontally aligned) is located between the base and an "upper arm"-like element. The second joint acts like an elbow and connects the "upper arm" with the "forearm". A gripper is attached to the "forearm", but without function. However, only 2 of them will be used to achieve the two objectives. The revolute joint of the base is locked so that the bending arm can reach the tube. Beside the robot arm and the tube there is also a camera in the Gazebo world, which is located beside the objects and captures the scene. The Gazebo World and the robot are connected to the DQN (uses the python-based deep learning framework named PyTorch) via a C++ API. The 2D camera images are used to fill the DQN agent with data. The neural network of the DQN is then trained on the basis of this image information and the corresponding rewards from the reward function, so that the correct actions for the robot's joints can be derived, also known as "pixels-to-actions".

2 Reward Functions

The basic idea behind the reward functions is to reward (positive points) or punish (negative points) the agent for the previous action he took. The agent's goal, of course, is to maximize his reward. With this knowledge, the deep neural network of the agent can now be trained so that a better decision is made for the next action - as a result the agent learns with every action. The reward functions must therefore be formulated in such a way that actions of the agent are rewarded (positive points) if they lead to the achievement of the tasks, or are punished (negative reward / negative points) if they lead away.
The actions the agent can perform is to control the joints of the robot arm. These can be controlled either by velocity, position or a mixture of both. With velocity control, either the angular velocity for each joint is increased or decreased individually. Position control, on the other hand, individually increases or decreases the angle of each joint. For this project, position control was used for both tasks.

2.1 Task 1

The following reward functions have been implemented for achieving **Task 1** - "any part of the robot arm shall touch the object of interest, with at least a 90% accuracy":
1. If any part of the robot arm touches the tube then agent gets the **REWARD_WIN of +300** and the running episode ends.
2. If episode runs longer than 100 frames, the running episode ends and agent gets **REWARD_LOSS of -300**.
3. If the Gripper touches the ground, the running episode ends and agent gets **REWARD_LOSS of -300**.
4. In addition to certain events (1.-3.), the agent also receives interim rewards (REWARD_INTERIM = +200) , depending on how he moves. The first step is to calculate the current distance *distGoal* between the gripper and the tube (the goal to hit). Then the following formulas are calculated:

$$distDelta = lastGoalDistance - distGoal \quad (1)$$

$$avgGoalDelta = (avgGoalDelta * \alpha) + (distDelta * (1 - \alpha)) \quad (2)$$

$$reward = avgGoalDelta * REWARD\_INTERIM \quad (3)$$

The value *distDelta* describes the deviation of the distance between gripper and tube to the previous step. If the gripper moves towards the target, this value is positive,

otherwise negative. With *avgGoalDelta* then calculates an average value from the previous deltas (previous value of *avgGoalDelta*) and the current delta *distDelta*, whereby the influence of the two variables is weighted differently with α. The best value for α was determined iteratively during testing. Good results for task 1 can be achieved with $\alpha = 3.0$. Finally, the value of *avgGoalDelta* was multiplied with the **REWARD_INTERIM of +200**, however, the true interim reward is dependent on *avgGoalDelta* and can therefore be negative as well. A growing positive value of *avgGoalDelta* then describes, for example, an arm continuously moving towards the target.

2.2 Task 2

The following reward functions have been implemented for achieving **Task 2** - "only the gripper base of the robot arm shall touch the object, with at least a 80% accuracy":
1. If gripper base (arm::gripperbase::gripper_link) of the robot arm touches the tube then agent gets the **REWARD_WIN of +300** and the running episode ends.
2. If any other part of the robot arm touches the tube then agent gets **REWARD_LOSS of -300** and the running episode ends.
3. If episode runs longer than 100 frames, the running episode ends and agent gets **REWARD_LOSS of -300**.
4. If the Gripper touches the ground, the running episode ends and agent gets **REWARD_LOSS of -300**.
5. In addition to the certain events (1.-4.) that can occur, the agent also receives interim rewards (**REWARD_INTERIM = +200**), depending on how he moves. Here, it is way more complex than for task 1. For the interim reward, the formulas (1) - (3) from task 1 were extended by further formulas. There are 2 ways the agent gets an interim reward.
a) The movements of the robot are small doesn't move. This was determined by,

$$abs(distDelta * 100) <= 0 \tag{4}$$

if (4) was true, then the agent was rewarded as follows (5):

$$reward = reward - 0.5 * REWARD\_INTERIM \tag{5}$$

So the reward he received was even smaller if he got stuck. This kept the robot moving.
b) If the robot was continuous moving, more formulas for the interim reward were implemented. At first, it was found out in the config file, that the distance of the Tube to the Base is set to 1.15m. As the Arm with the Gripper is more or less rotating on an arc with Joint 1 as center point, the distance of the Gripperbase should have approximately the same distance to its center, as the Tube has to the Base. Then, the Gripperbase would probably hit the Tube, which is the goal. So, the agent should get rewarded, if the Gripperbase has the right distance (1.15m) to the Robot Base. Therefore, the distance *distJoint1Grip* between Joint 1 and the Gripperbase was calculated. First the relative ratio of *distJoint1Grip* to the ideal distance of 1.15m is calculated. If *distJoint1Grip* is greater than 1.15m,

$$A = 1.15/distJoint1Grip \tag{6}$$

or else

$$A = distJoint1Grip/1.15 \tag{7}$$

This ratio is therefore at its maximum (A_max = 1) when the Gripper has the ideal distance (1.15m) to Joint 1. For this Joint 2 (elbow joint) must bend. Joint 2 can bend to the left or to the right so that A_max occurs. So there are two possible solutions for Joint 2. At the target point, when the Gripper Base touches the tube, there is only one reasonable solution. That's in the case where Joint 2 bends his forearm to the right. In this scenario, Joint 2 does not fall below a certain height Z either. This minimum distance of the Joint 2 for Z was therefore integrated in another formula. The minimum distance (0.42m) for Z was iteratively determined and approximated. Now the ratio of the distance of the lower edge of Joint 2 to the ground *distJoint2GroundMin* and the ideal distance 0.42m was calculated. If *distJoint2GroundMin* is greater than 0.42m the following was used,

$$B = 0.42/distJoint2GroundMin \tag{8}$$

or else,

$$B = distJoint2GroundMin/0.42 - 1 \tag{9}$$

For (9) the ratio B becomes even always negative. This is done to underly that Joint 2 never shall fall be lower than 0.42m. In addition to the previously mentioned values, a further ratio was calculated. This is the ratio *distGoal* to the maximum possible distance *maxDistGripper*. *maxDistGripper* is the maximum distance the Gripperbase can reach from the Tube. It was approximatively determined to *maxDistGripper*=2.63m. Therefore,

$$C = 1 - distGoal/maxDistGripper \tag{10}$$

becomes 1, when the Gripperbase touches the Tube. Otherwise it gets 0 if the Gripper moves away. With (3), (6), (7), (8), (9) and (10) the overall function for the Interim Reward is now as follows,

$$rewardFinal = reward * A * B * C \tag{11}$$

Good results for task 2 can be achieved with $\alpha = 9.0$ and **REWARD_INTERIM = +200**. The agent is therefore only rewarded positively if the Gripper Base moves towards the tube on a certain radius and with a certain joint setting for Joint 2.

3 HYPERPARAMETERS

In addition to good reward functions, it is also important to optimize the hyper parameters for the DQN agent. The tuning of the hyper parameters was accompanied by the finding of the reward functions. The following hyper parameters have been adjusted:

- **INPUT_WIDTH, INPUT_HEIGHT**: These two values represent the input size of the 2D image for the DQN agent. The default value is set to 512x512, but was iteratively reduced to increase the processing speed. For both tasks, the size was reduced to 64x64.
- **OPTIMIZER**: within the PyTorch Framework, different optimizing algorithms for the backpropagation are available like Adadelta, Adagrad, Adam, SparseAdam, Adamax, ASGD, LBFGS, RMSprop, Rprop and SGD. However, not all algorithms were

tested, but by trial and error, the RMSprop algorithm led to sufficient results.

- **LEARNING_RATE**: The learning rate is responsible for how much the weights change during optimization. It was initially set to 0.01 for both tasks. Unfortunately, it took very long (already for task 1) to see improvement of the arm's movements. So the runs have been aborted before reaching the objectives. Therefore the learning rate was increased. With 0.1 as final learning rate, the goals could be achieved.
- **REPLAY_MEMORY**: This memory stores the transitions that the agent observes and allows to reuse this data. In a later step, it is possible to take randomly samples from it to train the network and for decorrelation. The default value of 10000 was already sufficient for task 1. For task 2, the value was doubled to 20000.
- **BATCH_SIZE**: The batch size is the number of samples which are taken (randomly) from the replay memory to train and optimize the neural network. The bigger the batch size, the longer the training takes but as well the more accurate the estimate of the gradient will be. This is especially important for task 2, where only a small area is allowed to touch the Tube. For task 1, a batch size of 32 was sufficient, whereas for taks 2 a batch size of 512 was selected.
- **USE_LSTM**: LSTM (Long Short Term Memory) keeps track of the long-term memory and the short-term memory. The advantage of it is, that when training the network multiple past frames from the camera sensor are taking into consideration instead of a single frame. So this process is set to true for both tasks.
- **LSTM_SIZE**: It is the number of frames which are taken into consideration. For task 1 size was 256 and for 2, 512 frames are used.

TABLE 1
Hyperparameters

Parameter	Task 1	Task 2
INPUT_WIDTH	64	64
INPUT_HEIGHT	64	64
OPTIMIZER	RMSprop	RMSprop
LEARNING_RATE	0.1	0.1
REPLAY_MEMORY	10000	20000
BATCH_SIZE	32	512
USE_LSTM	true	true
LSTM_SIZE	256	512

4 Results

4.1 Task 1

The following Fig. 1 shows the results from Task 1: "any part of the robot arm shall touch the object of interest, with at least a 90% accuracy".

4.2 Task 2

The following Fig. 2 shows the results from Task 2: "only the gripper base of the robot arm shall touch the object, with at least a 80% accuracy". In Fig. 2 there are also a lot of other Debug statements shown. These debugging information has been deleted from the final code of ArmPlugin.cpp.

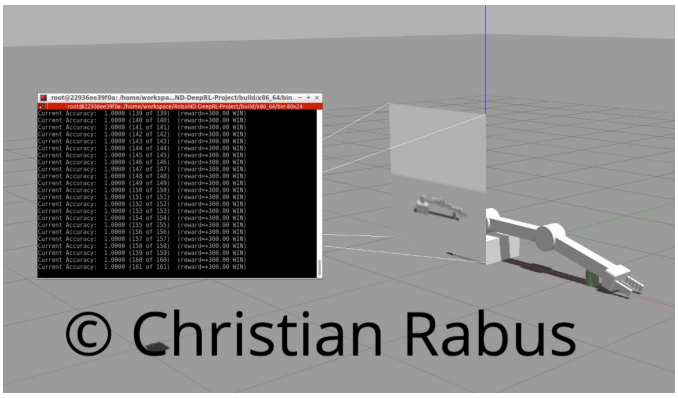

Fig. 1. Results from Task 1 show 100% accuracy

Fig. 2. Results from Task 2 shows 80.46% accuracy

5 Discussion

5.1 Task 1

After the appropriate Reward functions were implemented and the respective hyper parameters were set, the robot arm continuously moved to the desired position already after a few learning steps. Fig. 1 shows a well performed execution, whereas every try was successful. This result was not reproducible for every run with this high accuracy of 100%. In some cases, the robot arm either collided first with the ground (see 2.1.3) or the episode was ended because it was taking too long (see 2.1.2). In most cases, however, an accuracy of more than the required 90% was always significantly achieved. The implementation is therefore very well implemented for task 1.

5.2 Task 2

It was much more challenging to find the right reward functions and tuning of the hyperparameters for task 2 than task 1. In the first attempts it was tried to use the same reward functions as from task 1 and to limit the collision condition to the Gripper Base only. So the reward functions 2.2.1, 2.2.2, 2.2.3, 2.2.4 and 2.1.4 have been used. However, this did not lead to the desired result even after several adjustment attempts of the hyperparameters. Therefore, 2.1.4 was extended and led to the functions in 2.2.5. Due to the high memory and processing consumption caused by the

different hyperparameters, the execution for 1 run also takes longer because significantly more GPU resources are required. In the end, the minimum requirement of 80% could at least be narrowly reached. The result, however, first appears after about 260 episodes, with the curve flattening out at the end. However, the result was usually reproducible for several runs. Due to the long time it takes to reach the minimum requirement, the execution was aborted as soon as the 80% was reached. If you let the execution run longer, the accuracy might increase up to 82%-83%, but not further. The error that occurs again and again, even after many episodes (200 or more), is that the gripper first collides with the base (arm::gripperbase::gripper_link) on the tube, and then with the middle section (arm::gripper_middle::middle_collision). A REWARD_LOSS is returned from the reward function and is not counted as REWARD_WIN, even though the execution looks successful when watching it.

6 Future Work

Task 1 already delivers very good results. An optimization is therefore not necessary here. However, this task makes no sense in reality. The aim should be for the gripper to approach the tube so that it is between the two gripper arms. Then the robot can close them and lift the object. Task 2 therefore makes more sense when observed. Here, however, the collision task would have to be changed to the middle part (arm::gripper_middle::middle_collision). This means that a REWARD_WIN should only be returned if the gripper middle part (arm::gripper_middle::middle_collision) collides with the tube. The values from 2.2.5 would have to be adjusted accordingly. The values can certainly be improved by a more detailed evaluation. Furthermore, better functions could be found to describe the circular movement of the gripper.
The next step would probably be to implement further tasks: Unlocking the 3rd degree of freedom so that the base can swivel with a randomized positioning of the tube around the robot arm.
The implementation of the DQN algorithm on the Jetson TX2 in combination with a real robot arm would also be interesting.