Deep RL Arm Manipulation

Sandra Schuhmacher

Abstract—In this project Deep Q-Learning network is used to teach a serial manipulator to touch an object in front of it by reeinforcement learning. By defining befitting reward functions the robotic arm is trained to two different challenges.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, deep learning.
Repository: https://github.com/yulivee/RoboNDMapMyWorld

1 Introduction

With the advent of machine learning, a lot of task that were hard to solve by computing like image recognition are now feasible to solve by training a network with carefully chosen testdata and making the network figure out a way to solve the solution from the data instead of defining an algorithm. Robotic path planning is a computationally intensive task and in many cases done by hand, although all important information to automate it are usually present within companies who do system integration for robot arms: A model of the robot, machine data (velocity constraints, joint limits etc) and a model of the environment the robot is expected to work in including collisions. This work explores the question whether a robot arm can learn a path by using deep reeinforcement learning to automate the path planning task.

2 Setup

As a simplified experiment a 3 DoF serial manipulator is used with the objective to touch an object in front of it. Two challenges are accomplished in this experiment:

1) Have any part of the robot arm touch the object of interest, with at least a 90% accuracy over at least 100 episodes.
2) Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy over at least 100 episodes.

To achieve those challenges, a Deep Q-Learning Network (DQN) agent is given control over all three actuators of the robotic arm. The arm is a very simple manipulator: It features an actuator at the base and two joints in the arm. The gripper is not able to rotate. The DQN agent can control the actuators either by controling the velocity of the movement or the position of the joints. In the experiment both of those possibilities are explored.

To observe the learning process the agent is connected to a video-feed of a camera which was positioned sideways of it. At every turn the agent has to apply reeinforcement learning to make a decision on the next action and decide for a new movement. To aide the learning process and achieve the desired accuracy the following tasks are implemented:

1) Turn the action of the agent into a movement of the arm - either by increasing/decreasing joint velocity or changing the absolute positions of the joints.
2) Define rewards to encourage beneficial behaviour - apply a negative reward for unwanted actions, a positive reward for touching the object and an intermediate reward for movements in the correct direction.
3) Tune the hyperparameters of the DQN agents neural network to optimize the learning process

3 Reward Functions

For the reeinforcement learning, three types of rewards are defined:

Win — Reward issued in case the robot arm achieves the objective. This reward ends an episode with a success.

Loss — Penalty issued in case the robot arm performs behaviour that ends the episode without achieving the objective. It also ends an episode, but in failure.

Interim — Issued while the robot is moving and episode has not yet ended. Guides the DQN agent to move the robot into the correct direction and is positive or negative depending on the position of the arm.

The proportional size between those rewards significantly impact the learning performance. See chapter Results for details.

3.0.1 *Win Reward*

This positive reward is issued in case the robot arm touches the object. In case of challenge 1, this reward is issued if any part of the robot touches the object, during challenge 2 it is only issued if the gripper touches the object. The chosen value for the Win reward is a magnitude of 1000x larger than the interim rewards issued to put a huge weight on the final winning state and prevent the agent from collecting more interim reward through movement. A higher value also results in quicker learning after the arm achieved its objective a few times.

Challenge	**1**	**2**
Reward Gripper Contact	3000	3000
Reward Arm Contact	3000	-

3.0.2 Loss Reward

This negative reward is issued in the following cases:

1) the robot arm touches the ground
2) the middle joint of the robot arm touches the ground
3) the episode ends by timeout
4) (challenge 2 only) any part of the robots besides the gripper touches the object

The failure reward is different between challenges for some of those events. In challenge 1 the arm is more likely to encounter event 2, because contact with any part of the arm counts as a win. This results in a larger set of winning poses, which include poses where the middle joint touches the ground and the arm touches the object. For that reason, it has a larger penalty. In challenge 2 this pose is not in the set of wins and the robot does not try this pose very often.

The penalty for touching the ground is scaled by the distance from the object to serve as additional guidance for future state decisions - the nearer the arm is to the object when failing, the smaller the penalty. This is consistent for both challenges.

In challenge 2, no part besides the gripper is allowed to touch the object, so there is a high penalty for a collision with the arm. This penalty is more severe than touching the ground to discourage the arm from trying "flat" poses with an obtuse angle between base and gripper, as most of those poses result in failure.

Challenge	1	2
Ground Contact	-2000 * dist	-2000 * dist
Ground Contact w\Mid Joint	-3000	-2000
Episode Timeout	-2000	-2000
Object Contact with Arm	+3000	-2000

The loss reward is also an order of 1000x larger than the interim rewards to discourage the arm from trying those actions.

3.0.3 Interim Reward

The interim reward is a variable amount and can be either a reward or a penalty based on the position of the arm and the last action the arm took. The formula for the reward is the following:

```
REWARD_ALPHA = 0.2;
distDelta  = lastGoalDistance - distGoal;

weightedDelta =
distDelta * ( 1.0f - REWARD_ALPHA);
weightedHist =
avgGoalDelta * REWARD_ALPHA;

goalReward = weightedDelta + weightedHist;

scaledReward = goalReward * 20; // for Ch 1
scaledReward = goalReward;      // for Ch 2
avgGoalDelta = goalReward;

rewardHistory = scaledReward;
lastGoalDistance = distGoal;
```

`distDelta` is the difference between the distance to the object from the last state and the current distance to the object. By minimizing this value the arm can improve its performance, therefore it is scaled up (`weightedDelta`). `avgGoalDelta` is the reward from the last step, scaled down (`weightedHist`). The reason for the scaling is, that the current distance has more impact on the learning than the last reward. In case of Challenge 1, the arm took to long to converge to a winning solution, so the total reward (`goalReward`) was scaled up additionally to speed up the process.

3.1 ML Hyperparameters

The DQN agents hyperparameters are the same for both challenges.

Parameter	Value	default
INPUT_WIDTH	64	512
INPUT_HEIGHT	64	512
OPTIMIZER	Adam	None
LEARNING_RATE	0.02f	0.0f
REPLAY_MEMORY	10000	10000
BATCH_SIZE	64	8
USE_LSTM	true	false
LSTM_SIZE	256	32
EPS_DECAY	100	200

The first modification to the provided default is to modify width and height to 64 to save memory, as this is the size of the camera input feed. At first some tests with the RMSprop optimizer are made, but the Adam optimizer outperformed its results. It seems that the *running average of recent magnitude* of RMSprop did not fit the problem space as well as the *stochastic optimization* of the Adam optimizer. The default learning rate of the optimizier is slightly increased to make the agent learn faster. The `BATCH_SIZE` and `LSTM_SIZE` are increased with values determined by trial and error and perform best at the presented values.

4 Results

The task for challenge 1 was harder to achieve than that for challenge 2, as there were two winning poses for the arm to touch the object - one with the midjoint above the gripper, the other one with the midjoint touching the ground. As no part of the robot was allowed to touch the ground, an additional check if the midjoint made ground contact was implemented. Getting the arm to not try this pose was a major challenge to overcome. Scaling the interim reward posed a good countermeasure. The interim reward scaling provided some surprising results: while expermenting with a value of 30 for scaling (instead of 20 which was settled on) the robot exhibited some kind of analysis paralysis, where it did nothing but quickly jitter between two poses with little difference.

With the final set of parameters, the arm made it to the 90% requirement, but did not seem to improve more than that. The result can be seen in 1

Applying the same set of options to challenge 2 resulted in 3% accuracy, as the arm often under- of overshot the object. As a first measure penalties and rewards were scaled back, as there weren't so many poses to be prevented compared to challenge 1. This lifted the accuracy to 43%. For experimentation the joint control of the robot was changed to position control instead of velocity control which led to

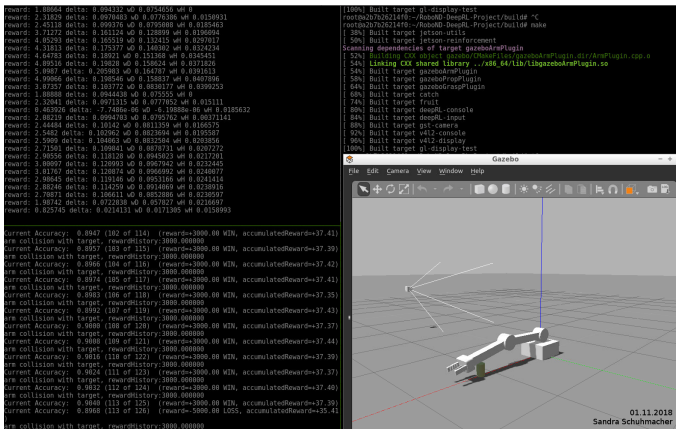

Fig. 1. Robot Arm touching Object in Challenge 1

86% accuracy at first try. The reduced state space of the position control and the less influence of the set of actions that brought it to its position could be reasons for the significant improvements. The final results of challenge 2 can be seen in 2 and 3.

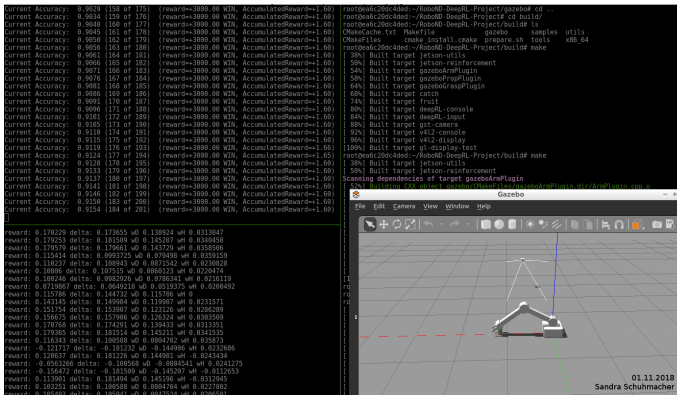

Fig. 2. Robot Arm touching Object in Challenge 2

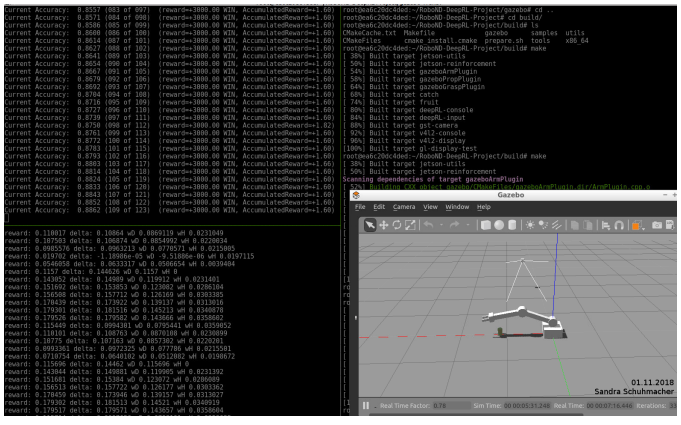

Fig. 3. Robot Arm touching Object in Challenge 2

In both cases the arm did a lot of exploration in the beginning and after around 30 episodes started to settle on a winning move with rare outliers of different attempts.

5 Discussion and Future Work

The tuning of the ML hyperparameters and the reward functions took considerable time to figure out correctly, and the parameters did not look the same for those two settings, even though this was a very simple robot arm and the difference between the challenges was not very For the initially presented scenario of removing path planning for complex robots in custom robot cell environments, this seems an infeasable method, as the hyperparameter tuning and definition of the reward functions probably takes longer then planning the path by hand. Also, there is a lot of randomness in the resulting path the robot figures out, which does not befit an industry dependent on high precision, reproducabilty and optimal machine utilization.

There is probably a use to reeinforcement learning in path optimization, where the robot can explore along the planned path for e.g. a less power-consuming path or a quicker way between those points. Another field interesting for reeinforcement learning could be to performed bin-picking, where the gripper of a robot has to grip an object that lies in a random position in a pile. This is a narrow problem that is hard to achieve with traditional optimization algorithms. This is a step that could be explored with a camera and the Jetson TX2 hardware.