Deep RL Arm Manipulation

Lucas Wohlhart

Index Terms

Robot, Udacity, Reinforcement Learning, Object Manipulation

1 Introduction

The goal of the DeepRL Project is the creation of an artificial agent operating a 3DoF robot manipulator arm to solve the task of touching a target can positioned in front of it. The base of the arm rotates around the z-axis and the two joints linking the serial arm segments to the base share the same rotation axis perpendicular to the z-axis. The agent is tasked with controlling the joints such that it touches the can in front of it based on an image provided by a camera positioned next to the manipulators workspace. The control task at hand therefore falls into the category of visual servoing. The experimental setup is depicted in Fig. 1.

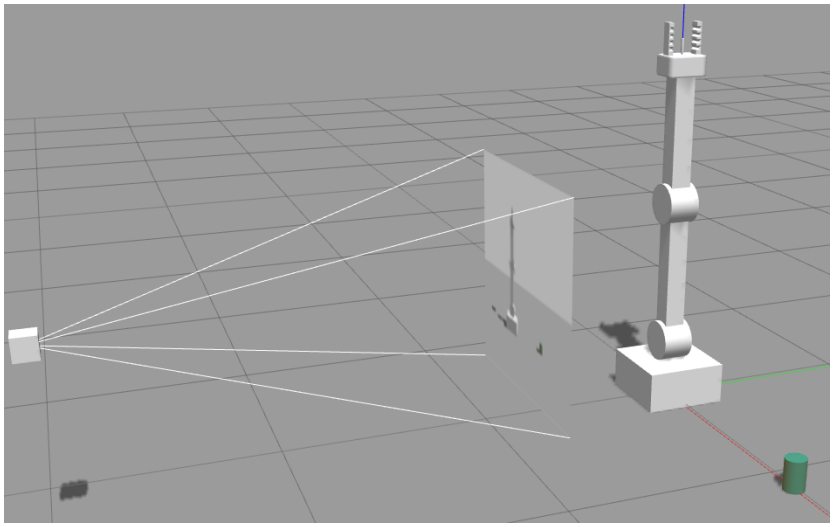

Fig. 1: Robot manipulator setup

In task 1 of the project the entire arm of the manipulator is counted as valid collision target and an accuracy of 90% successful touches over at least 100 attempts has to be achieved.

In task 2 only the base of the parallel jaw gripper attached to the endeffector is allowed to touch the can for a successful run and the accuracy has to exceed 80% out of more than 100 iterations.

For all the experiments the base joint is locked, reducing the task to two degrees of freedom.

The agent could be set up to either operate on joint positions or joint velocities as control parameters. The controlling agent has to choose between increasing or decreasing either of the two joints position/velocity. This yields 4 available actions the agent has to choose from at each time step, based on its policy and the provided input image state.

Through the use of reinforcement learning techniques the problem of training the agent can be solved by repeatedly letting it try to deduce actions based on the image and reward intended behaviour while penalizing erroneous actions.

2 Reward Functions

The mechanisms used to reward the reinforcement learning agent are the following.

- Whenever the agent sucessfully hits the can it receives REWARD_WIN = 100 and the episode ends. In task 2 hitting the can with anything else but the gripper yields REWARD_LOSS = -30.
- If the gripper ever hits the ground or the 100th simulation step is exceeded, the reward is REWARD_LOSS = -30 and the episode is terminated.

- For each simulation step the average motion of the endeffector towards the goal ($avgGoalDelta$) is determined

$$avgGoalDelta \leftarrow \alpha * avgGoalDelta + (1 - \alpha) * distDelta$$

where smoothing factor $\alpha = 0.2$ and $distDelta$ is the difference in endeffector-to-goal-distances of two consecutive steps.
The resulting intermediate reward that is issued every timestep is then comprised of a scaled motion based REWARD_GOAL_APPROACHING = 20 and a very small constant penalty of REWARD_TIME_PENALTY = -0.1.

$$reward = avgGoalDelta * 20 - 0.1$$

This yields positive rewards for average motion of the endeffector towards the goal.

3 HYPERPARAMETERS

The agents created for task 1 and task 2 operated on changing the joint positions with each sampled action.

Table 2 hyperparameters gives provides a summary for the hyperparameters described in this section that are used to train were agent for both tasks:

INPUT_WIDTH	64
INPUT_HEIGHT	64
OPTIMIZER	RMSprop
LEARNING_RATE	0.05
REPLAY_MEMORY	10000
BATCH_SIZE	128
USE_LSTM	false
LSTM_SIZE	#
EPS_START	0.9f
EPS_END	0.01f
EPS_DECAY	200

Fig. 2: Hyperparameters

INPUT_WIDTH x INPUT_HEIGHT

The camera image that acts as state input for the agent is downsized to a 64x64pixels image to reduce state and model complexity while still providing enough information and resolution to sufficiently identify the required elements in the scenery.

OPTIMIZER

During tuning of the hyperparameters Adam as well as RMSprop were evaluated and appeared to yield essentially the same results. RMSprop was chosen for the final training of both task 1 and task 2.

LEARNING_RATE

The step size the optimizer takes after evaluating the gradient descent vector is scaled by the LEARNING_RATE. It gives control over the tradeoff between speed of convergence of the learning algorithm convergence stability. After launching several training cycles with learning rates of 0.2, 0.1, 0.05 or 0.01 it became clear that eventhough higher learning rates also occasionally were able to learn good behaviour, the final rate of 0.05 repeatedly yielded successful agents.

REPLAY_MEMORY

The size of the experience pool the algorithm retains to sample from for each policy update step is determined by the REPLAY_MEMORY parameter. Setting this size to 10000, as done to achieve the results for task 1 and task 2, means the agent stores its past 10000 experiences in a first in first out manner and randomly recalls a batch of BATCH_SIZE form this pool to improve its policy.

USE_LSTM & LSTM_SIZE

After training several agents using an LSTM layer of 16, 64 and 128 nodes it was observed that many of these agents quickly ran into overfitting to some suboptimal policy, while some achieved relatively good accuracy. Omitting the use of the LSTM layer produced much more stable learning results and therefore USE_LSTM was set to false for the final agents. I hypothesize that having an LSTM layer to retain temporal information might be beneficial for the task at hand but this layer seems to be comparably difficult to train appropriately.

EPS

The EPS parameters, controlling the rate of exploration vs exploitation were changed to start at EPS_START 0.9 and exponentially decay over EPS_DECAY 200 steps to EPS_END 0.01. This means the agent starts out with 90% of actions being randomly chosen but once the policy starts to converge the remaining small amount of just 1% exploration allow the controller to exploit the learned policy.

4 Results

The video available at https://youtu.be/n06wDGm7ll4 show the training of the agent for task 1 where it's allowed to touch the can with any part of the arm to get full reward. The final accuracy of the agent reaches 92.06% as shown in Fig 3.

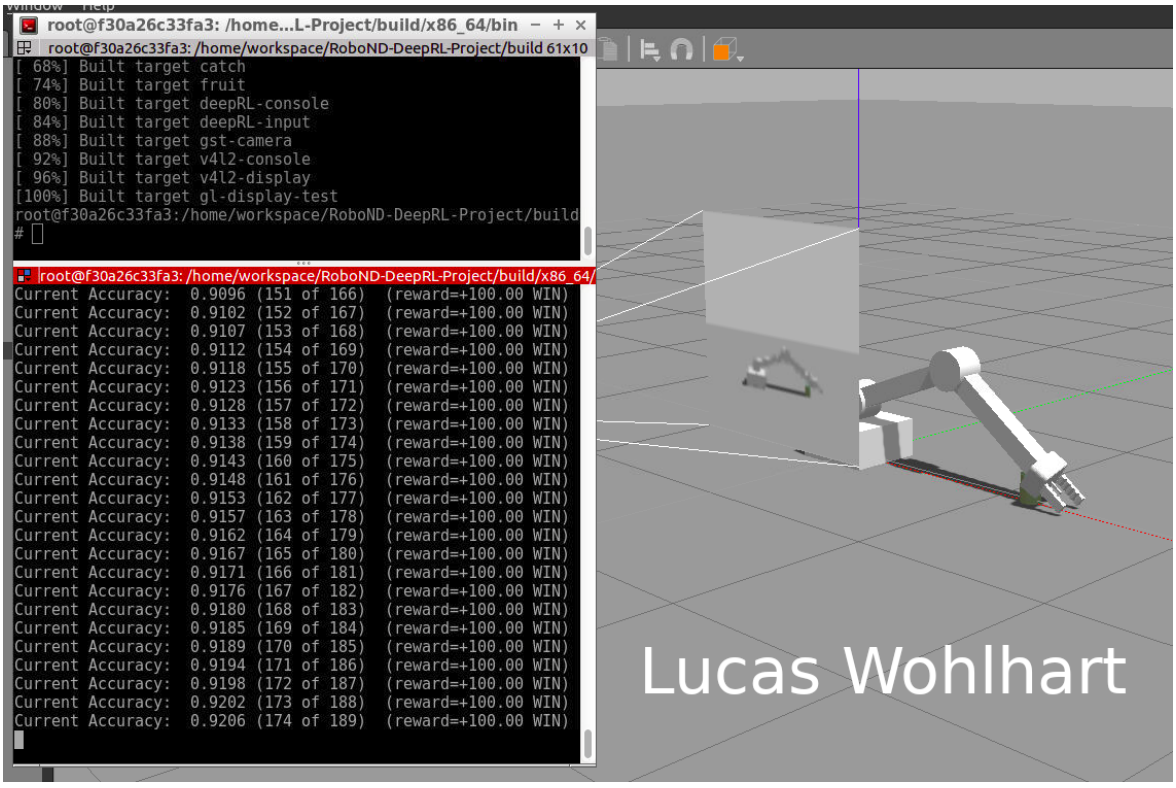

Fig. 3: Task 1 final accuracy

The video https://youtu.be/frCadMNOyA0 shows some parts of the training of the agent for task 2. At first the agent struggles to find the most rewarding goal at all. Once it reaches this goal it is capable of optimizing for this behaviours. Every now and then the already good policy seems to deteriorate a bit for a small period of time. It starts to consecutively miss the target for a short time span of 3 or 4 episodes eventhough the residual exploration probability is down to approximately 1%.

The final accuracy of the agent reaches 82.97% as shown in Fig 4.

5 Future Work

As always, hyperparameter search is a tedious process but once a reasonable configuration is found the creation of intelligent agents by simply reinforcing intended behaviour is a truely amazing approach.

Randomizing the target object position as well as the initial pose of the manipulator would be a very interesting next extension to this experiment. Also unlocking the base-joint of the arm or even mounting the manipulator on a mobile base (such as a differential drive robot) to ultimately train a mobile manipulator with reinforcement learning could yield very promising results for flexbile industry scenarios.

Overall the agent was able to perform both required tasks. However, it would have been wise to first just train the agent and evaluate this model, regarding the task requirements, without residual action noise. This would allow the use of much more exploration noise during training to truely cover the entire state space and achieve more than just one or two successful motion patterns.

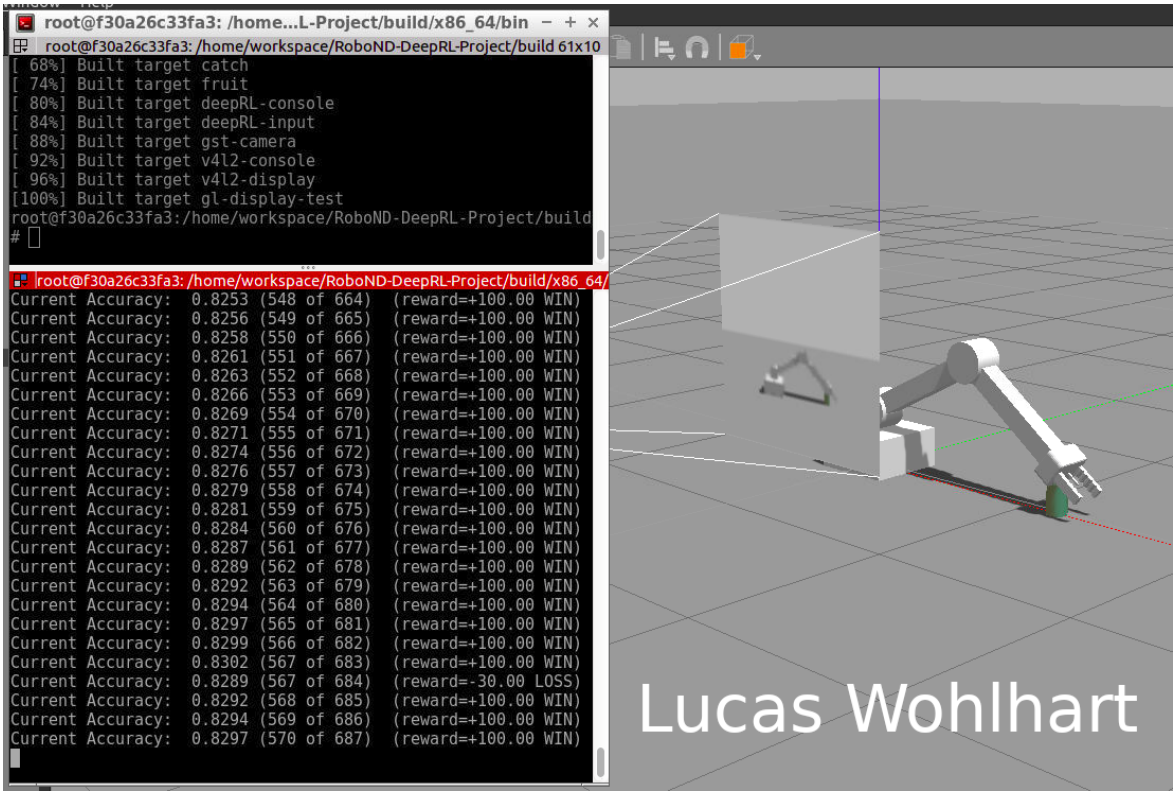

Fig. 4: Task 2 final accuracy