Project: Deep RL Arm Manipulation

Vladimir Kobal

October, 2018

1 Reward Function

The average of the delta of the distance to the goal is smoothed with the basic exponential smoothing: avgGoalDelta = (avgGoalDelta * ALPHA) + (distDelta * (1 - ALPHA)), where ALPHA = 0.3. Joints are controlled by setting positions.

1.1 First task: the arm touches the object

For the first task, reward function encourages the robot arm to move towards the goal. For ground collisions and episode timeouts it takes into account the distance to the goal at the last iteration. To encourage the agent speeding up the winning sequence the number of episode frames is discounted from the reward. In order to slow down the arm at the end of the winning sequence the last avgGoalDelta value is also discounted from the reward.

```
REWARD_WIN = 10
REWARD_LOSS = -30
```

Event	Reward
Collision between the arm and the object	REWARD_WIN + maxEpisodeLength - episodeFrames - 1000 * avgGoalDelta
Robot hit the ground	REWARD_LOSS - 10 * lastGoalDistance
Episode timeout	REWARD_LOSS - 10 * lastGoalDistance
Interim reward	10 * avgGoalDelta

Table 1: Rewards for the first task

1.2 Second task: the gripper middle touches the object

For the second task with the gripper middle touching the object, the reward function coefficients were changed. avgGoalDelta was excluded from the winning reward as far as the robot arm didn't tend to speed up too much. Square root function was applied to lastGoalDistance to make the influence of the distance to the goal more prominent when the arm is closer to the goal.

```
REWARD_WIN = 10
REWARD_LOSS = -1 -100 * sqrt(lastGoalDistance)
```

Event	Reward
Collision between the gripper and the object	REWARD_WIN + maxEpisodeLength - episodeFrames
Collision between the arm and the object	REWARD_LOSS
Robot hit the ground	REWARD_LOSS - 10
Episode timeout	REWARD_LOSS
Interim reward	80 * avgGoalDelta

Table 2: Rewards for the second task (gripper middle)

1.3 Second task: the gripper base touches the object

For the second task with the gripper base touching the object, the reward function was drastically simplified, and the range of its values was decreased.

```
REWARD_WIN = 0.5
REWARD_LOSS = -lastGoalDistance
```

Event	Reward
Collision between the gripper and the object	REWARD_WIN
Collision between the arm and the object	REWARD_LOSS
Robot hit the ground	REWARD_LOSS
Episode timeout	REWARD_LOSS
Interim reward	avgGoalDelta

Table 3: Rewards for the second task (gripper base)

2 Hyperparameters

Input width and height were decreased down to 64x64 (the size of images, taken by the camera). The number of actions was set to the number of joints multiplied by 2, as every joint can move in two directions. With small replay memory size, the agent can't remember sequences of motions, and large values didn't lead to better results, so this value was set to the default one. RMSprop optimizer does the job for all three cases.

2.1 First task: the arm touches the object

The task is simple, so we don't need exploration at all (ALLOW_RANDOM is set to "false"). LSTM and batch size were increased. When LSTM size is big, the agent tends to refine the winning sequence and remember it well. Experiments with Gamma and learning rate showed that these parameters do not play a crucial role in the agent behavior, so moderate values were chosen.

Hyperparameter	Value
INPUT_CHANNELS	3
ALLOW_RANDOM	false
GAMMA	0.7
EPS_START	-
EPS_END	-
EPS_DECAY	-
INPUT_WIDTH	64
INPUT_HEIGHT	64
NUM_ACTIONS	DOF * 2
OPTIMIZER	RMSprop
LEARNING_RATE	0.01
REPLAY_MEMORY	10000
BATCH_SIZE	32
USE_LSTM	true
LSTM_SIZE	128

Table 4: Hyperparameters for the first task

2.2 Second task: the gripper middle touches the object

For the second task with the gripper middle touching the object, exploration was enabled, and Epsilon end value was decreased to prevent agent exploring an action space after the winning sequence was found. Gamma was decreased in order to pay more attention to the most immediate reward. With slightly increased learning rate agent was giving more consistent results.

Hyperparameter	Value
INPUT_CHANNELS	3
ALLOW_RANDOM	true
GAMMA	0.4
EPS_START	0.9
EPS_END	0.01
EPS_DECAY	200
INPUT_WIDTH	64
INPUT_HEIGHT	64
NUM_ACTIONS	DOF * 2
OPTIMIZER	RMSprop
LEARNING_RATE	0.05
REPLAY_MEMORY	10000
BATCH_SIZE	32
USE_LSTM	true
LSTM_SIZE	128

Table 5: Hyperparameters for the second task (gripper middle)

2.3 Second task: the gripper base touches the object

For the second task with the gripper base touching the object, exploration was radically prolonged, but the probability of random actions was decreased. With LSTM disabled the successful task resolution became much more reproducible. When LSTM was enabled, the arm movements was quicker and smoother, but the agent was tending to repeat non-optimal action sequences.

Hyperparameter	Value
INPUT_CHANNELS	3
ALLOW_RANDOM	true
GAMMA	0.9
EPS_START	0.5
EPS_END	0.0001
EPS_DECAY	1000
INPUT_WIDTH	64
INPUT_HEIGHT	64
NUM_ACTIONS	DOF * 2
OPTIMIZER	RMSprop
LEARNING_RATE	0.01
REPLAY_MEMORY	10000
BATCH_SIZE	256
USE_LSTM	false
LSTM_SIZE	-

Table 6: Hyperparameters for the second task (gripper base)

3 Results

For both objectives, the DQN agent's performance was quite good. The expected accuracy was achieved in less than 100 steps (Fig. 1, 3). The main problem was solving the second task with enabled LSTM. Not only the expected accuracy was achieved in more than 100 steps (Fig. 2), but successful runs were not steadily reproducible.

4 Future work

Further improvements could be made to the winning sequence speed and to the agent's ability to reproduce successful runs. It can be achieved by fine-tuning reward function and parameters. It also worth trying to use velocity based control, as it can lead to quicker and more precise movements of the robot arm. Though, with such control, it will be more difficult for an agent to find a solution.

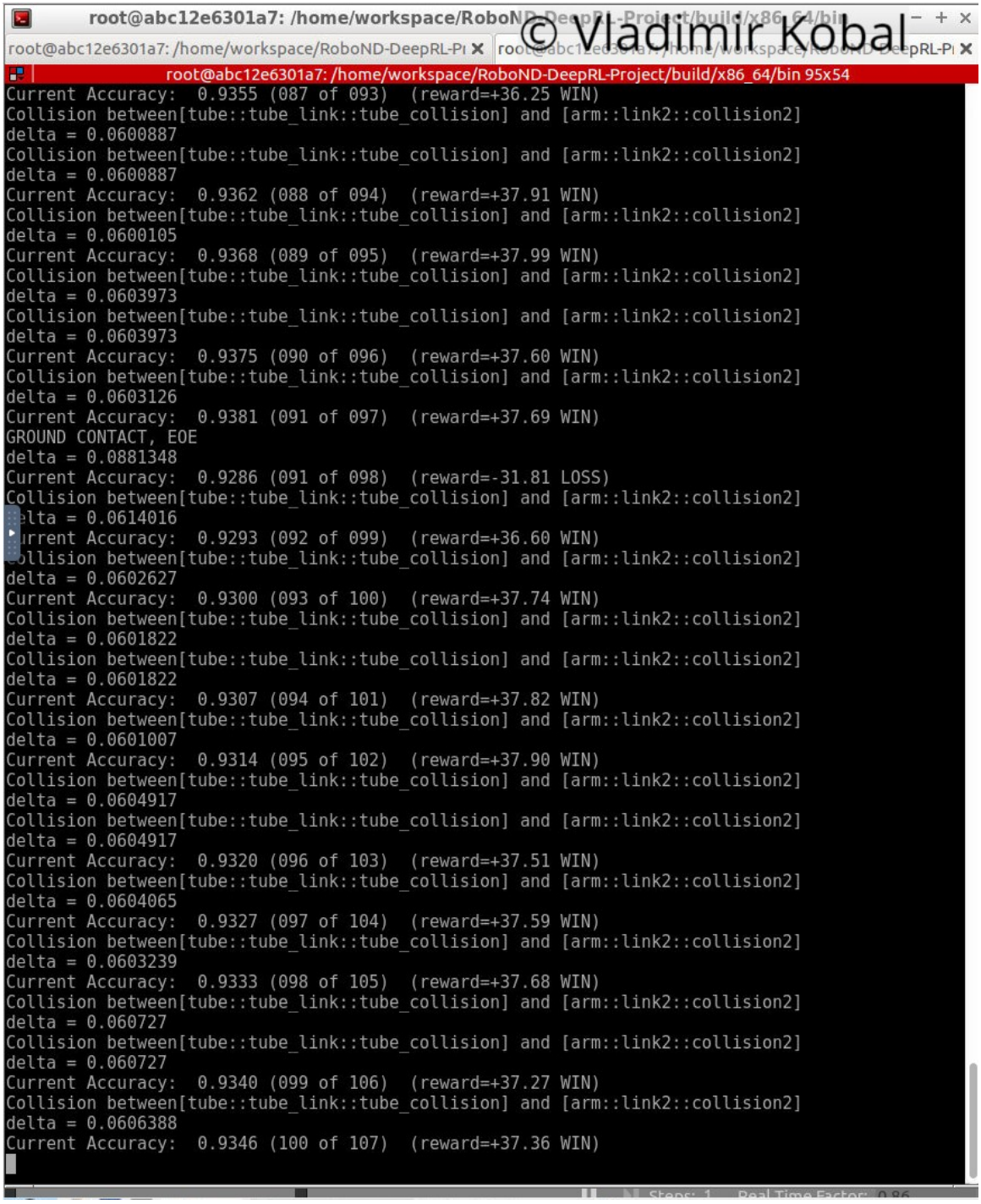

Figure 1: The accuracy of the arm touching the object

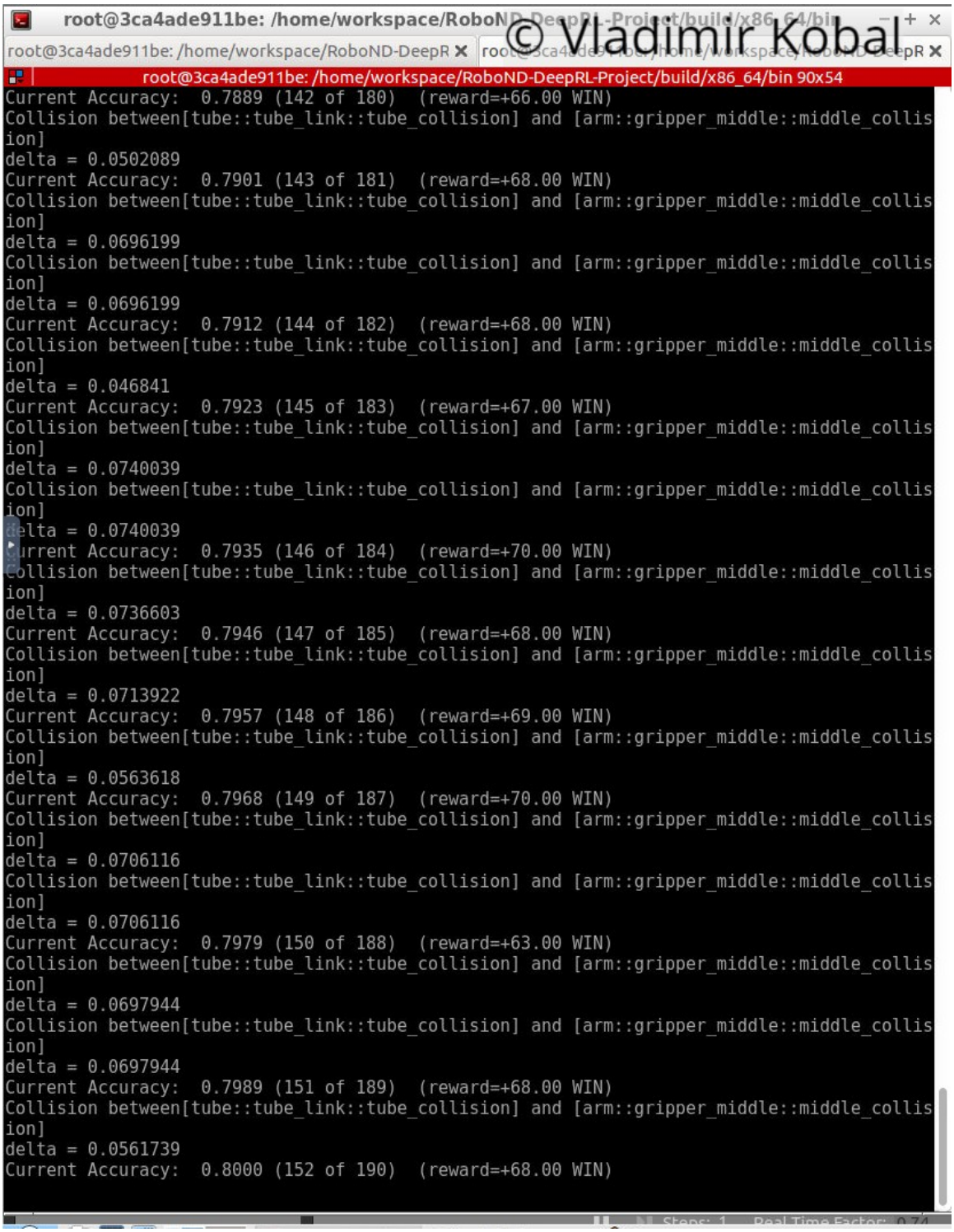

Figure 2: The accuracy of the gripper middle touching the object

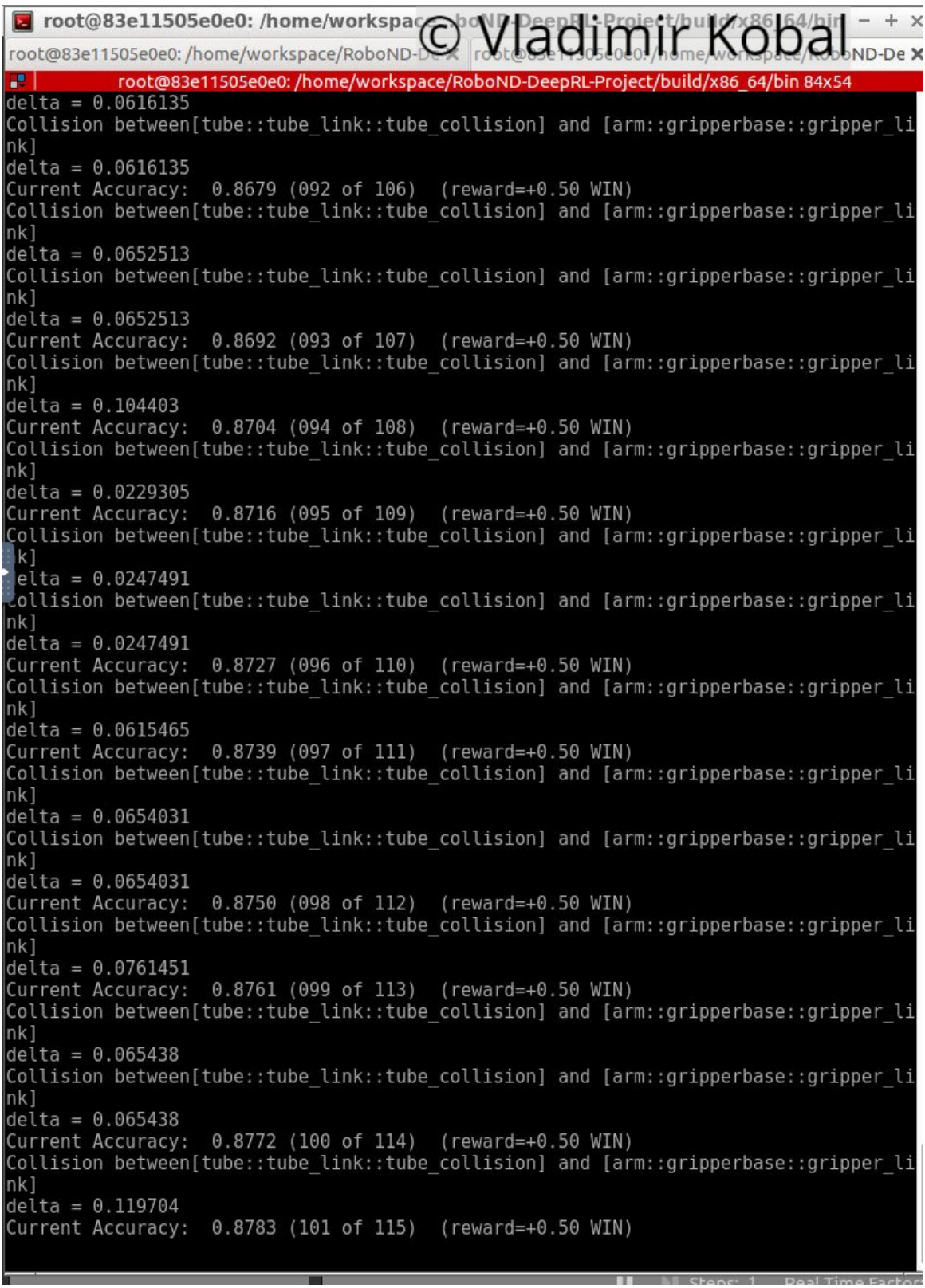

Figure 3: The accuracy of the gripper base touching the object