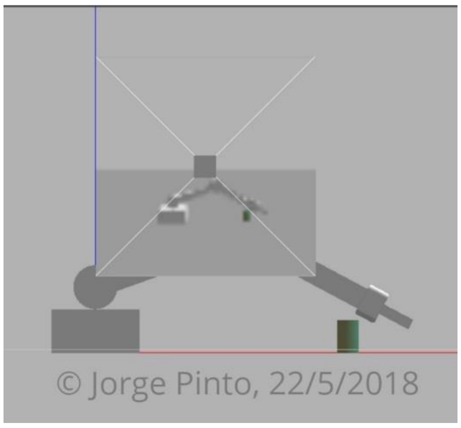


DEEP RL ARM MANIPULATION PROYECT REPORT

This report will provide the reader with the outcome of setting up a simulation for a robotic Arm operated under the reinforcement learning paradigm. The results herein contained will show the response of the Arm Manipulator simulator to a series of hyperparameters. In this order of ideas, the primary objectives to accomplish in this project were:

1. Have any part of the robot arm touch the object of interest, with at least a 90% accuracy for a minimum of 100 runs.
2. Have only the gripper base of the robot arm touch the object, with at least an 80% accuracy for a minimum of 100 runs.

In addition, three challenges were also proposed after completion of the above said clauses:

1. Object's randomization.
2. Increasing the Arm's Reach.
3. Increasing the Arm's Reach with Object Randomization.

For the sake of providing a reference to the contents herein portrayed, the reader is prompted to check the references from Mnih et al (2015) at the end of this document.

1. Reward Functions

To ease the implementation of the functions, three global variables were defined: REWARD_WIN, REWARD_LOSS and REWARD_FACTOR. Such were included in the rewarding process as it was developed in terms of the actions that may occur during the operation of the arm. These were categorized as follows:

- Collisions: for providing positive or negative reward in terms of the instances involved in a detected collision (or contact), namely the arm, gripper, object of interest and the ground.
- Distance: intended to provide rewarding in terms of how close the arm is to the object of interest.

For the first case, positive collisions were defined if either the arm or the gripper touched the object of interest. The rationale is simple: if a collision is detected between arm and object, provide a positive reward (+1); if the collision involves the pair gripper-object, provide the double of the originally defined reward (+2); but if the arm touched the ground (or the episode limit was reached), provide a negative reward (-1). Keep in mind though that negative rewards were not defined as being cumulative in time.

In terms of the second case, rewarding was implemented by defining a smoothing average of the distance's difference to the goal in time:

$$distDelta = lastGoalDistance - distGoal$$

$$avgGoalDelta = (avgGoalDelta * \alpha) + (distDelta * (1.0 - \alpha))$$

$$rewardHistory = avgGoalDelta * REWARD\_FACTOR$$

With α a smoothing factor of 0.1 (current value's contribution accounts for 10% of the total calculated average over the historical data). The final reward was calculated in terms of the average difference of accumulated distance to the goal (i.e. the object of interest). That's is, if distDelta was less than a defined limit (0.25 in this case), a new reward would be calculated through $rewardHistory$. The REWARD_FACTOR shown in the equation was set at a default value of 100 right before the simulation started. On the other hand, position control was left as default for the DQN agent to control the joints.

2. Hyperparameters' settings

The following hyperparameters were modified at the beginning of the ArmPlugin.cpp file:

- Input's width and height: to 64 pixels both.
- Optimizer type: RMSProp for gradient descent optimization algorithm.
- Replay memory: 10000
- Batch size: 32
- Use and size of Long-short term memory: set to FALSE and left at 256 frames.

Although they may seem conservative at a first glance, these values proved to be adequate for the completion of the objectives described previously. On the other hand, the DQN agent included another series of parameters given by definition

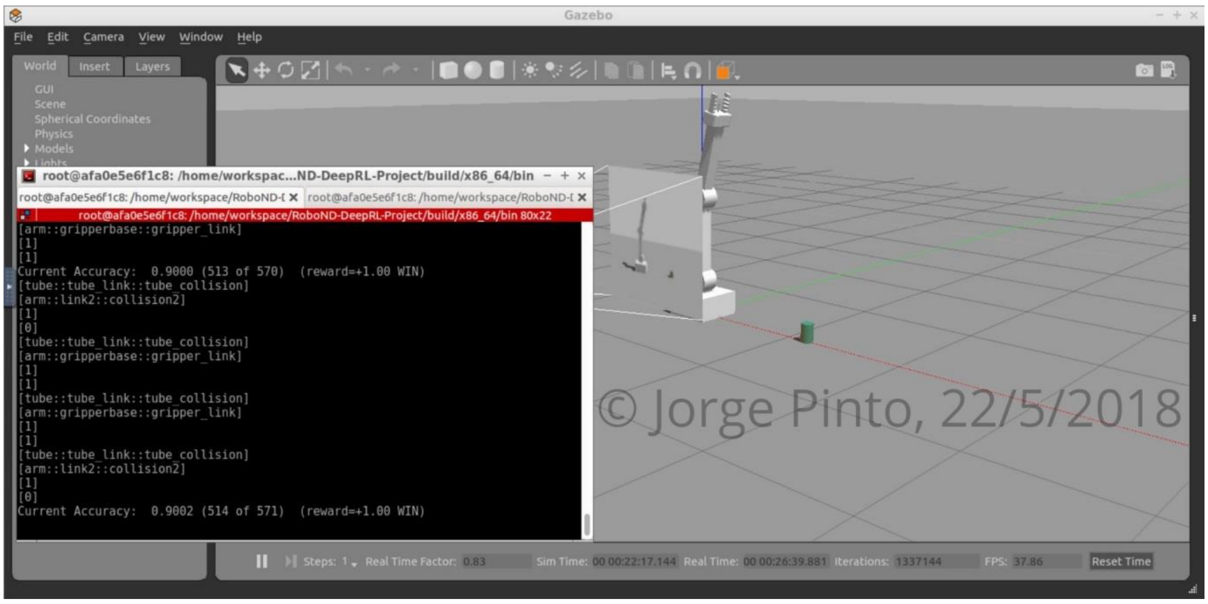
$$Q(s_t, a_t) \leftarrow (1-\alpha) * Q(s_{t-1}, a_{t-1}) + \alpha * (r_t + \gamma * maxQ(s_{t+1}, a))$$

- Learning rate α: different from the smoothing factor defined above. It was established at 0.1 to provide a higher weight to the old state value rather than the future.
- Discount factor γ: To higher the value of rewards received earlier in time. Set at 0.9 before starting the simulation. Set as per default.

Other parameters as the optimizer type and episode definitions were left as per default: RMSProp, EPS_START at 0.9, EPS_END at 0.01 and EPS_DECAY at 200 respectively.

3. Objectives Analysis

Results for the first objective can be seen in Figure 1. In this case scenario, more than 500 episodes were required to achieve the 90% accuracy threshold, as it was expected due to the simple definition of the rewarding function.



```
root@afa0e5e6f1c8: /home/workspac...ND-DeepRL-Project/build/x86_64/bin
root@afa0e5e6f1c8: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x22
[arm::gripperbase::gripper_link]
[1]
[1]
Current Accuracy:  0.9000 (513 of 570)  (reward=+1.00 WIN)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.9002 (514 of 571)  (reward=+1.00 WIN)
```

Figure 1: First Objective accuracy for +1.0 value of positive reward

For the second objective, Figure 2 shows an accuracy higher than 80% in less of registered episodes. The difference resides in the double rewarding defined for every time the gripper touched the object. In either first or second scenario, all mentioned parameters were kept constant. Nonetheless, for the posed challenges different values for positive reward were set.

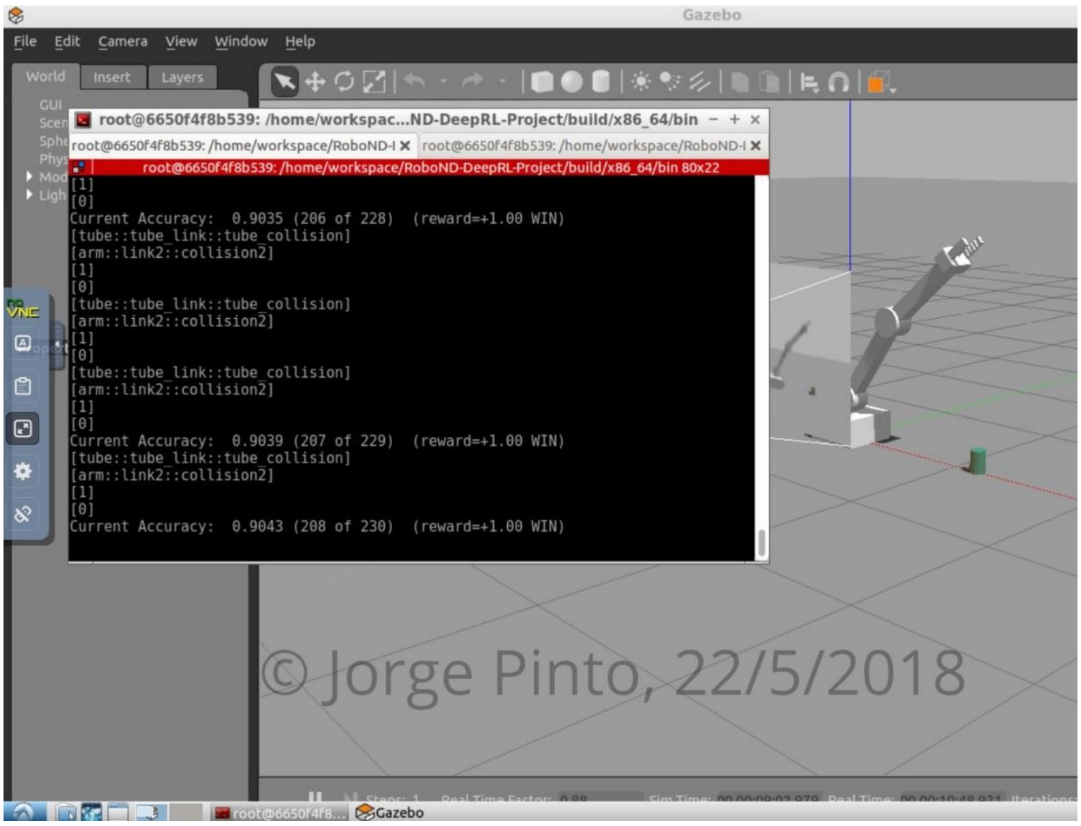
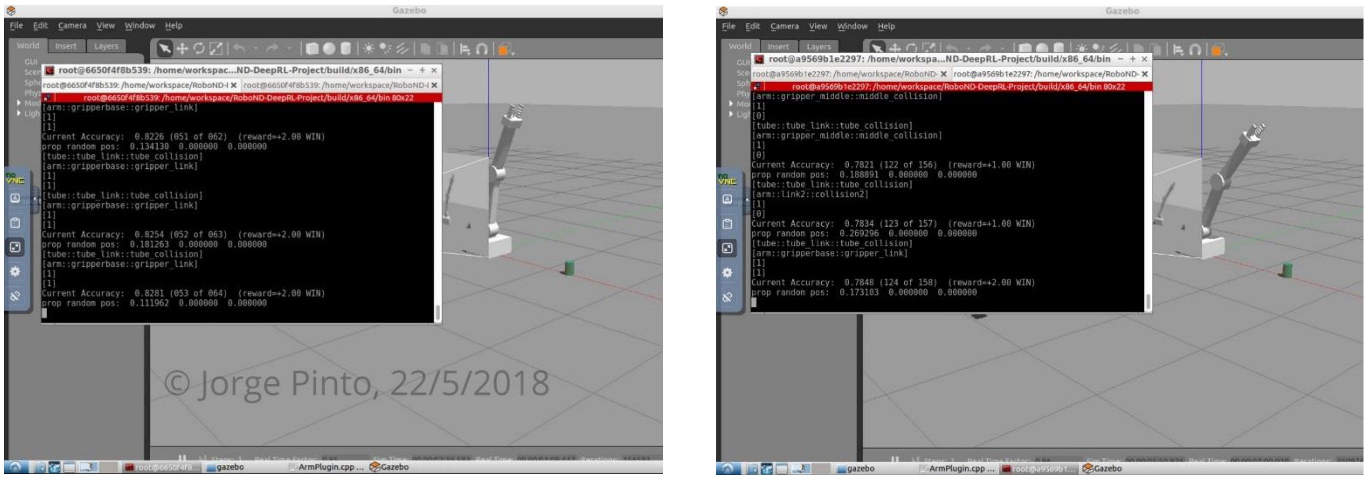

```
[1]
[0]
Current Accuracy:  0.9035 (206 of 228)  (reward=+1.00 WIN)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.9039 (207 of 229)  (reward=+1.00 WIN)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.9043 (208 of 230)  (reward=+1.00 WIN)
```

Figure 2 Second Objective accuracy for +1.0 value of positive reward

4. Challenges Completion

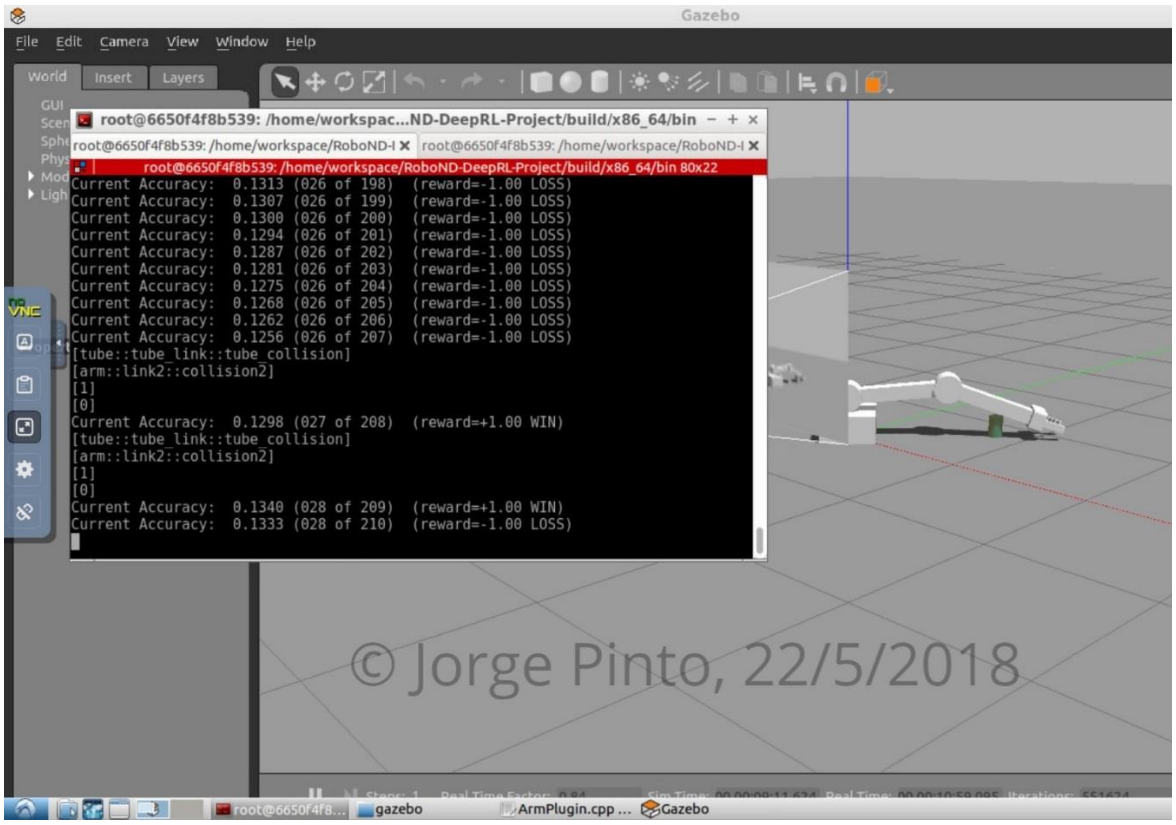
For the object's randomization challenge in one dimension, the results gathered are shown in Figure 3 for the original values of positive reward. At the beginning, accuracy picked up quickly to above 80%, but as iterations increased, it got reduced to 79% at most (which is good keeping in mind the random position of the object).


```
[arm::gripperbase::gripper_link]
[1]
[1]
Current Accuracy:  0.8226 (051 of 062)  (reward=+2.00 WIN)
prop random pos:  0.134130  0.000000  0.000000
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
Current Accuracy:  0.8254 (052 of 063)  (reward=+2.00 WIN)
prop random pos:  0.181263  0.000000  0.000000
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
Current Accuracy:  0.8281 (053 of 064)  (reward=+2.00 WIN)
prop random pos:  0.111962  0.000000  0.000000
```

```
[arm::gripper_middle::middle_collision]
[1]
[0]
[tube::tube_link::tube_collision]
[arm::gripper_middle::middle_collision]
[1]
[0]
Current Accuracy:  0.7821 (122 of 156)  (reward=+1.00 WIN)
prop random pos:  0.188891  0.000000  0.000000
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.7834 (123 of 157)  (reward=+1.00 WIN)
prop random pos:  0.269296  0.000000  0.000000
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
Current Accuracy:  0.7848 (124 of 158)  (reward=+2.00 WIN)
prop random pos:  0.173103  0.000000  0.000000
```

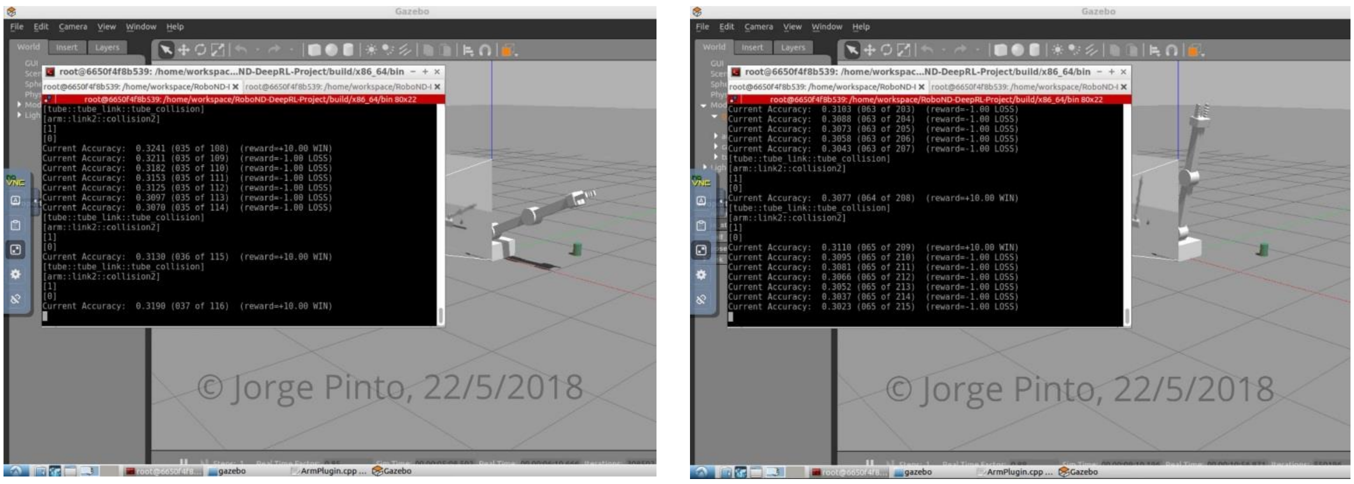
Figure 3 First Challenge: Preliminary VS Final Accuracy

On the other hand, for the second challenge, the object's location was modified to $(x, y) = (0.75, 0.75)$ and was kept fixed during the entire simulation. The revolute base of the arm was set to freely rotate around its own vertical axis for the arm to reach the position. As positive reward value was kept constant, preliminary results provided a poor accuracy of 13% at most, even after 200 iterations (as shown in Figure 4). Increasing the positive reward to +10.0 did improve the accuracy to 40% around the first 40 iterations and reaching an average value of 30% after 100 and 200 iterations respectively (see Figure 5).



```
root@6650f4f8b539: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x22
Current Accuracy:  0.1313 (026 of 198)  (reward=-1.00 LOSS)
Current Accuracy:  0.1307 (026 of 199)  (reward=-1.00 LOSS)
Current Accuracy:  0.1300 (026 of 200)  (reward=-1.00 LOSS)
Current Accuracy:  0.1294 (026 of 201)  (reward=-1.00 LOSS)
Current Accuracy:  0.1287 (026 of 202)  (reward=-1.00 LOSS)
Current Accuracy:  0.1281 (026 of 203)  (reward=-1.00 LOSS)
Current Accuracy:  0.1275 (026 of 204)  (reward=-1.00 LOSS)
Current Accuracy:  0.1268 (026 of 205)  (reward=-1.00 LOSS)
Current Accuracy:  0.1262 (026 of 206)  (reward=-1.00 LOSS)
Current Accuracy:  0.1256 (026 of 207)  (reward=-1.00 LOSS)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.1298 (027 of 208)  (reward=+1.00 WIN)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.1340 (028 of 209)  (reward=+1.00 WIN)
Current Accuracy:  0.1333 (028 of 210)  (reward=-1.00 LOSS)
```

Figure 4 Second Challenge: Preliminary accuracy for +1.0 value of positive reward



```
root@6650f4f8b539: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x22
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.3241 (035 of 108)  (reward=+10.00 WIN)
Current Accuracy:  0.3211 (035 of 109)  (reward=-1.00 LOSS)
Current Accuracy:  0.3182 (035 of 110)  (reward=-1.00 LOSS)
Current Accuracy:  0.3153 (035 of 111)  (reward=-1.00 LOSS)
Current Accuracy:  0.3125 (035 of 112)  (reward=-1.00 LOSS)
Current Accuracy:  0.3097 (035 of 113)  (reward=-1.00 LOSS)
Current Accuracy:  0.3070 (035 of 114)  (reward=-1.00 LOSS)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.3130 (036 of 115)  (reward=+10.00 WIN)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.3190 (037 of 116)  (reward=+10.00 WIN)
```

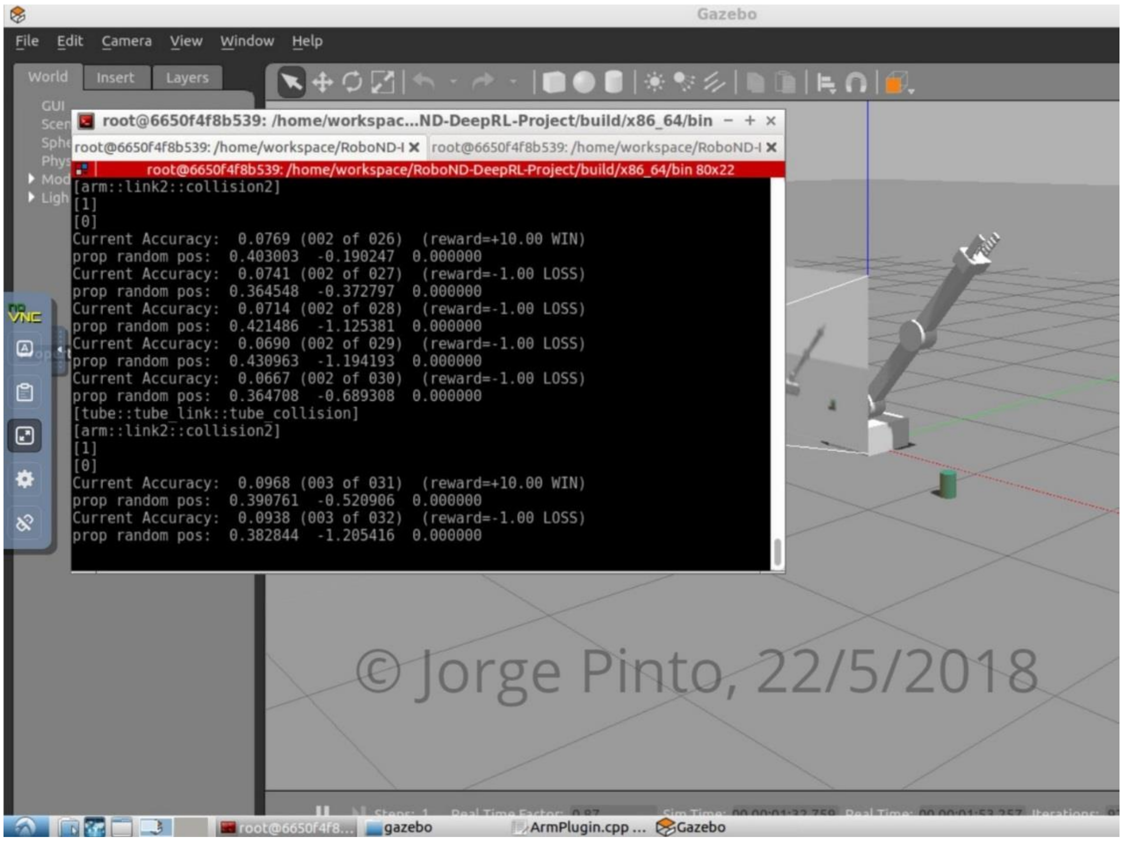
```
root@6650f4f8b539: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x22
Current Accuracy:  0.3103 (063 of 203)  (reward=-1.00 LOSS)
Current Accuracy:  0.3088 (063 of 204)  (reward=-1.00 LOSS)
Current Accuracy:  0.3073 (063 of 205)  (reward=-1.00 LOSS)
Current Accuracy:  0.3058 (063 of 206)  (reward=-1.00 LOSS)
Current Accuracy:  0.3043 (063 of 207)  (reward=-1.00 LOSS)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.3077 (064 of 208)  (reward=+10.00 WIN)
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.3110 (065 of 209)  (reward=+10.00 WIN)
Current Accuracy:  0.3095 (065 of 210)  (reward=-1.00 LOSS)
Current Accuracy:  0.3081 (065 of 211)  (reward=-1.00 LOSS)
Current Accuracy:  0.3066 (065 of 212)  (reward=-1.00 LOSS)
Current Accuracy:  0.3052 (065 of 213)  (reward=-1.00 LOSS)
Current Accuracy:  0.3037 (065 of 214)  (reward=-1.00 LOSS)
Current Accuracy:  0.3023 (065 of 215)  (reward=-1.00 LOSS)
```

Figure 5 Second Challenge: Accuracy at +10.0 value of positive reward

Finally, the third challenge was built on top of the two previous ones, thus letting the arm to freely rotate around its vertical axis and allowing the object to randomly change its position in the (x,y) plane. Keep in mind though that limits were set to both coordinates so the object was always reachable by the arm. Reward value was kept at +10.0 from the earlier challenge.

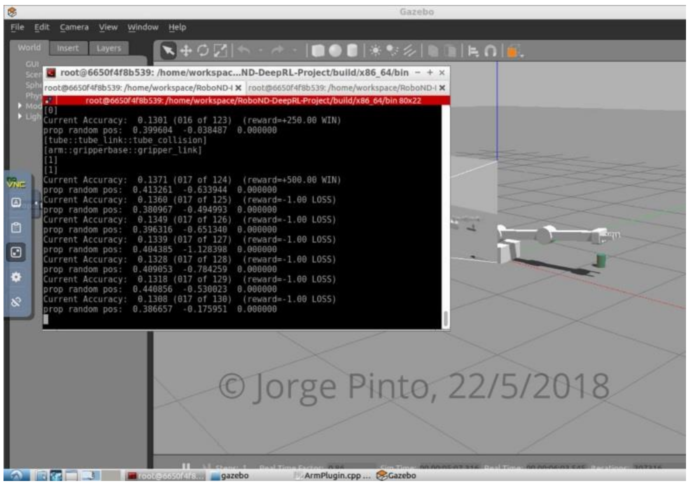
Initially, reported accuracy values were poor: below 5% for first 30 iterations as seen in the figures. The simulation was therefore stopped as no further improvement was expected after the 100 iterations (see Figure 6). Thus, while keeping the same rewarding strategy, the positive value was raised to +250.0 therefore making it +500.0 if the gripper touched the object of interest. Results can be observed in next figure for both over 100 and 300 iterations.

Clearly, raising the awarded reward did improve accuracy reaching a maximum of 23% in 390 iterations (see Figure 7). Nonetheless, it is far below the 79% of the first challenge where some randomness was included in the experience. That been said, for non-deterministic environments it becomes more difficult for any robot to process (memorize) all the possible cases that may present, even with a replay management set to 10000 frames.

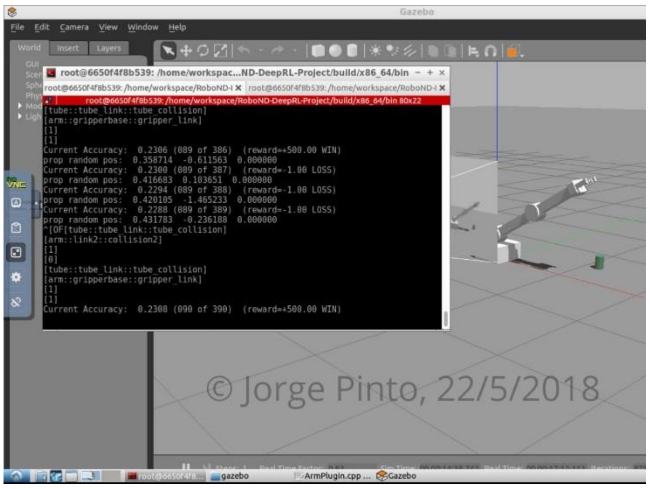


```
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.0769 (002 of 026)  (reward=+10.00 WIN)
prop random pos:  0.403003  -0.190247  0.000000
Current Accuracy:  0.0741 (002 of 027)  (reward=-1.00 LOSS)
prop random pos:  0.364548  -0.372797  0.000000
Current Accuracy:  0.0714 (002 of 028)  (reward=-1.00 LOSS)
prop random pos:  0.421486  -1.125381  0.000000
Current Accuracy:  0.0690 (002 of 029)  (reward=-1.00 LOSS)
prop random pos:  0.430963  -1.194193  0.000000
Current Accuracy:  0.0667 (002 of 030)  (reward=-1.00 LOSS)
prop random pos:  0.364708  -0.689308  0.000000
[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
Current Accuracy:  0.0968 (003 of 031)  (reward=+10.00 WIN)
prop random pos:  0.390761  -0.520906  0.000000
Current Accuracy:  0.0938 (003 of 032)  (reward=-1.00 LOSS)
prop random pos:  0.382844  -1.205416  0.000000
```

Figure 6 Third Challenge initially reported accuracy for +10.0 of positive reward



```
[0]
Current Accuracy:  0.1301 (016 of 123)  (reward=+250.00 WIN)
prop random pos:  0.399604  -0.038487  0.000000
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
Current Accuracy:  0.1371 (017 of 124)  (reward=+500.00 WIN)
prop random pos:  0.413261  -0.633944  0.000000
Current Accuracy:  0.1360 (017 of 125)  (reward=-1.00 LOSS)
prop random pos:  0.380967  -0.494993  0.000000
Current Accuracy:  0.1349 (017 of 126)  (reward=-1.00 LOSS)
prop random pos:  0.396316  -0.651340  0.000000
Current Accuracy:  0.1339 (017 of 127)  (reward=-1.00 LOSS)
prop random pos:  0.404385  -1.128398  0.000000
Current Accuracy:  0.1328 (017 of 128)  (reward=-1.00 LOSS)
prop random pos:  0.409053  -0.784259  0.000000
Current Accuracy:  0.1318 (017 of 129)  (reward=-1.00 LOSS)
prop random pos:  0.440856  -0.530023  0.000000
Current Accuracy:  0.1308 (017 of 130)  (reward=-1.00 LOSS)
prop random pos:  0.386657  -0.175951  0.000000
```



```
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
Current Accuracy:  0.2306 (089 of 386)  (reward=+500.00 WIN)
prop random pos:  0.358714  -0.611563  0.000000
Current Accuracy:  0.2300 (089 of 387)  (reward=-1.00 LOSS)
prop random pos:  0.416683  0.103651  0.000000
Current Accuracy:  0.2294 (089 of 388)  (reward=-1.00 LOSS)
prop random pos:  0.420105  -1.465233  0.000000
Current Accuracy:  0.2288 (089 of 389)  (reward=-1.00 LOSS)
prop random pos:  0.431783  -0.236188  0.000000
^[OF[tube::tube_link::tube_collision]
[arm::link2::collision2]
[1]
[0]
[tube::tube_link::tube_collision]
[arm::gripperbase::gripper_link]
[1]
[1]
Current Accuracy:  0.2308 (090 of 390)  (reward=+500.00 WIN)
```

Figure 7 Reported accuracy 130 VS 390 iterations (+250.0 positive reward)

Would the accuracy reach a higher value if iterations were left to run above and beyond say 500 iterations? It is hard to say. At the time the simulation stopped, accuracy had already oscillated between 18 % and 23% in several occasions and the same applies to the second challenge experience. At this point, possibly a better strategy could have improved results even further for these two cases.

5. Future Work

LSTM was tried in all cases to check differences it obtained results. However, it didn't provide an accuracy higher than 2% with the implemented rewarding strategy. Short term memory just didn't work at the long run as no results are registered whatsoever. As a matter of fact, the arm had a difficult time reaching the object of interests if this label set to true. It would be very interesting to further test the reach of such a paradigm in the future to get a better understanding of it.

DQN performance analysis is another point of interest. Gamma and episodes duration was kept at default given the accuracies it provided at the beginning. It is worth mentioning that it is unknown if changing such values would provide different results. For example, the value 0.9 intended to provide a better chance for the future state Q to be a success. Once again, whether lowering this value or using another optimizer process would provide another result is left for future work.

As a final note, the rewarding strategy implemented in this project was simple and provided positive rewards as the robot reached and contacted the object of interest (velocity control of the joint was tested in both first and second objectives tasks, but the joints showed less controlled movements). The rewarding factor included in the calculated distance of the arm to the goal also played an important role in positioning the gripper close the object. Nonetheless, adaptive strategies, something that makes the robot to not only learn the actions to take but also how to reward them, would make a difference if implemented in this project.

6. References

- Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529.
- Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).