<u>Using Deep Reinforcement Learning to Teach a Robot Arm Accurate Reaching Movements</u>

Arnold Kompaniyets

Robotic Software Engineering, Term 2

Project 4

Reward Functions:

Objective 1:

In order to properly describe the reward functions comprehensively, critical portions of the ArmPlugin.cpp file will be discussed in sequence. The pre-made ArmPlugin.cpp file provided for this project appeared to be an exact copy of the “jetson-reinforcement” Nvidia folder in GitHub, with just a few variable names changed and multiple sections of code removed. The Udacity project outline attempted to offer various hints and clues as to how the code should be structured, but these were very brief and, quite frankly, rather difficult to understand. As such, the Nvidia code was inspected for help, in order to fully understand the intent of the coder himself. Afterwards, necessary alterations could be made to apply to this specific project. First:

```
/*
/ TODO - Define Reward Parameters
/
*/
```

```
#define REWARD_WIN  100.0f

#define REWARD_LOSS -100.0f
```

The reward win and loss values were assigned a value of 100. This value was decided upon after a bit of experimentation, altering between 20 and 1,000. Overall, the actual value of the reward did not appear to have a significant impact on the performance of the program, but rather its value in ratio to other reward values described below. Therefore, for convenience, the value of 100 was kept. There was no need seen to have one of the reward values lower or higher than the other, so only the sign of the value was changed (i.e. positive or negative).

Next, the reward functions for collision were defined. For the first task of achieving contact with any portion of the robot arm 90% of the time, the function was kept simple. When seeing the portion of the reward function already defined, the method implemented was to compare the name of the collision element provided to that defined in the function (using the "strcmp" function). The predefined line stated to ignore the floor colliding with either the arm or tube, which makes sense, since the two latter objects rest on the floor. In DEBUG mode, it was found that two collision objects were always listed upon collision: collision1 and collision2. When in relation to the arm contacting the tube, the arm was always "collision2" and the tube "collision1". Therefore, the following code was written:

```
if (strcmp(contacts->contact(i).collision1().c_str(), COLLISION_ITEM) == 0)

        {

                rewardHistory = REWARD_WIN;
                newReward  = true;
                endEpisode = true;

                return;
        }
```

The term COLLISION_ITEM was already defined in the ArmPlugin.cpp template file provided, so it was assumed that this was the term ought to be used. The reward amount was set to that specified in REWARD_WIN, afterwards ending the episode.

With the winning reward defined, negative reward values needed to be specified as well. The template ArmPlugin.cpp already defined a variable for the gripper model (which corresponds to the "gripper_middle" link), along with its bounding box (using the GetBoundingBox() function). With "groundContact" defined as well, simply as a float value of 0.05, the resulting function was typed as such:

```
    if(gripBBox.min.z <= groundContact || gripBBox.max.z <= groundContact)
    {

        if(DEBUG){printf("GROUND CONTACT, EOE\n");}

        rewardHistory = REWARD_LOSS;
        newReward     = true;
        endEpisode    = true;
        }
```

In Gazebo, the main reference frame has the z-axis corresponding to vertical height, which is why gripBBox.min.z and gripBBox.max.z were the chosen variables. Both the min and max had to be included here, due to the uncertainty of the position/orientation of the middle gripper link upon ground contact (due to the uncertain motion of the arm itself). Upon contact, the value for REWARD_LOSS was given, and the episode was ended, forcing the robot arm to restart its journey should contact be made with the ground.

Furthermore, the intermediate reward function needed to be defined as well. This function was intended to give the robot arm higher reward with closer distance to the tube. The tube already had a pre-defined bounding box, "propBBox", so the "BoxDistance" function was used to attain the direct distance between the two objects in question, the second object being the middle gripper – this distance was called the "distGoal". Next:

```
if( episodeFrames > 1 )
{
        const float distDelta  = lastGoalDistance - distGoal;
        const float alpha = 0.9;

        // compute the smoothed moving average of the delta of the distance to the goal
        avgGoalDelta  = (avgGoalDelta * alpha) + (distDelta * (1.0f - alpha));
        rewardHistory = avgGoalDelta * 10.0f;
        newReward     = true;
}

lastGoalDistance = distGoal;
```

The average change in distance was computed with the assistance of a pre-defined variable (lastGoalDistance). Following this, as was instructed in the project outline, a smoothing moving average was calculated via the implementation of an alpha value. The alpha variable was defined as a float, and then the formula described in the project outline was implemented. The alpha value itself was initially started at 0.5, but this resulted in far too erratic movement by the robot arm. Moving the value to 0.75 proved ineffective as well, so 0.9 was tried, which proved to be effective enough at smoothing the arm motion without slowing the movement and trajectory significantly. Values above 0.9 (such as 0.95 or 0.99) were tried, but didn't show improvement, so 0.9 was kept. The delta variable was initially used as the reward itself, but it was quickly discovered that altering this reward value had significant impact on the success of the application as a whole. Therefore, the "avgGoalDelta" was multiplied by a chosen value, 10.0 in the case of the first objective.

<u>Objective 2:</u>

In order to successfully accomplish the second objective of having only the gripper base touch the green tube, a few additions and changes were done to the reward functions.

The REWARD_WIN and REWARD_LOSS values were left unchanged. However, the contact reward function was expanded upon, as such:

```
                if ((strcmp(contacts->contact(i).collision1().c_str(), COLLISION_ITEM) == 0) &&
(strcmp(contacts->contact(i).collision2().c_str(), COLLISION_POINT) == 0))
                {
                        rewardHistory = REWARD_WIN;
                        newReward  = true;
                        endEpisode = true;

                        return;
                }

                if ((strcmp(contacts->contact(i).collision1().c_str(), COLLISION_ITEM) == 0) &&
(strcmp(contacts->contact(i).collision2().c_str(), COLLISION_POINT) != 0))
                {
                        rewardHistory = REWARD_LOSS;

                        newReward  = true;
                        endEpisode = false;

                        return;
                }
```

The conditional statement for contact with the tube was modified to include not only the "collision1" element, but also specify to award the REWARD_WIN only when the "collison2" object is the gripper base (defined by the pre-defined COLLISON_POINT element). The opposing situation, therefore, was also defined (if the arm touches the tube, but not with the gripper base), this time returning the REWARD_LOSS value. The major point to note, though, is that while initially the episode was ended upon touching the tube with a different part of the arm, this proved to make it incredibly difficult for the robot to learn the desired movement. Therefore, the "endEpisode" value was left at false, to allow the robot arm to learn from the error and still continue towards its objective. Within just a few runs, the robot arm would only touch the tube with the gripper base, avoiding any other part of the robot arm altogether.

Next, the reward functions for ground contact were modified, as such:

```
        // get the bounding box for the gripper
        const math::Box& gripBBox = gripper->GetBoundingBox();
        const math::Box& jointBBox = joint->GetBoundingBox();
        const math::Box& griprBBox = gripperbase->GetBoundingBox();
        const float groundContact = 0.01f;

        /*
        / TODO - set appropriate Reward for robot hitting the ground.
        /
        */


        if(gripBBox.min.z <= groundContact || gripBBox.max.z <= groundContact)
        {

                if(DEBUG){printf("GROUND CONTACT, EOE\n");}

                rewardHistory = REWARD_LOSS;
                newReward     = true;
                endEpisode    = true;
        }

/*
/       if(jointBBox.min.z <= groundContact || jointBBox.max.z <= groundContact)
/       {
/
/               if(DEBUG){printf("GROUND CONTACT, EOE\n");}
/                   rewardHistory = REWARD_LOSS;
/                   newReward     = true;
/                   endEpisode    = true;
/         }
*/
```

Attaining the desired movement from the robot arm for this more specific objective proved to be a much larger challenge. First, the robot arm would register as hitting the ground far too frequently while attempting to zone in on the trajectory angle. This kept the arm from learning appropriately, so the "groundContact" variable was lowered to 0.01, in order to allow the middle gripper unit to be closer to the ground plane before failing the task.

Additionally, a rather frequent problem in training was observed with the middle arm joint (joint2) being lowered to the ground instead of the gripper base. In order to try and avoid this issue, new variables were defined for the joint link (in the main ArmPlugin file and the header file), ending up with attaining the bounding box for this particular link. The same pattern for reward was initiated for this link, forcing the robot to end the episode upon ground contact. However, this proved to not work as expected, either placing the middle joint far to the left, or worse, still ending in the joint hitting the ground and ending a given episode before sufficient learning could be done. Therefore, it was decided to leave this component of code out and focus on the other reward parameters instead.

The final reward component altered was that of the intermediate returns, coded as such:

```
          const float distGoal = BoxDistance(griprBBox, propBBox);
```

```
				const float jointGoal = BoxDistance(jointBBox, propBBox);

				if(DEBUG){printf("distance('%s', '%s') = %f\n", gripper->GetName().c_str(), 
prop->model->GetName().c_str(), distGoal);}


				if( episodeFrames > 1 )
				{
					const float distDelta  = lastGoalDistance - distGoal;
					const float alpha = 0.9;

					// compute the smoothed moving average of the delta of the 
distance to the goal
					avgGoalDelta  = (avgGoalDelta * alpha) + (distDelta * (1.0f - 
alpha));
					rewardHistory = avgGoalDelta * 9.0f;
					newReward     = true;
				}

				lastGoalDistance = distGoal;
```

First, the object used in calculating direct distance to the green tube was changed from the middle gripper to the gripper base itself, since that was the desired part to contact the tube. Afterwards, the same procedure was followed to calculate the average change in overall distance. The number used to multiply this value was experimented with significantly (from 0.5 to 100.0), but a value of 9.0 seemed to produce the best results. As a side note, in order to tempt the robot arm into achieving the desired pose and route of movement, a secondary reward function was made, using the direct distance from the second joint to the green tube; instead though, the reward was reversed, meaning that a greater reward was given the further away joint2 was from the green tube. This attempt to further keep joint2 from hitting the ground did not prove successful (as it kept the arm itself too far to the left), so it was removed from the final code.

In regards to joint control, velocity control was kept at "false" for both objectives. With respect to code, each of the two methods was written the same:

```
float velocity = vel[action/2] + actionVelDelta * ((action % 2 == 0) ? 1.0f : -1.0f);

float joint = ref[action/2] + actionJointDelta * ((action % 2 == 0) ? 1.0f : -1.0f);
```

With "vel" and "ref" referring to each possible joint, the indexing was chosen as simply the action number divided by 2. Since integer division in C++ only returns the integer portion of the division, the six possible actions would always return their respective joints if divided by 2 (0/2 and 1/2 would return the index 0, 2/2 and 3/2 would return the index 1, and 4/2 with 5/2 would return the index 2). In order to state whether the action is positive or negative, a question conditional statement was used. Taking the modulus division of the action by 2, the conditional statement can decide if the action is even or odd. If even, the action delta value would be multiplied by 1.0 and if odd, the value would be multiplied by -1.0. After coding both methods, velocity control was attempted, but produced no improved functioning of the arm. As such, the normal joint control was left.

Hyper-parameters:

For both objectives, the hyper-parameters were kept as such:

```
#define INPUT_WIDTH   64
#define INPUT_HEIGHT  64
#define OPTIMIZER "RMSprop"
#define LEARNING_RATE 0.1f
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 64
#define USE_LSTM true
#define LSTM_SIZE 256
```

In the Gazebo world file, the camera object was inspected, and the image width and height were both coded as “64”. With this being the size of the Gazebo camera image, the deep RL network’s input width and height were taken to be the same, changing the above two variables as such. Next, the optimizer was chosen as “RMSprop”, following directly from examples presented previously in the deep RL lessons. The learning rate was altered various times, ranging from 0.5 to 0.001, but 0.1 appeared to work the best.

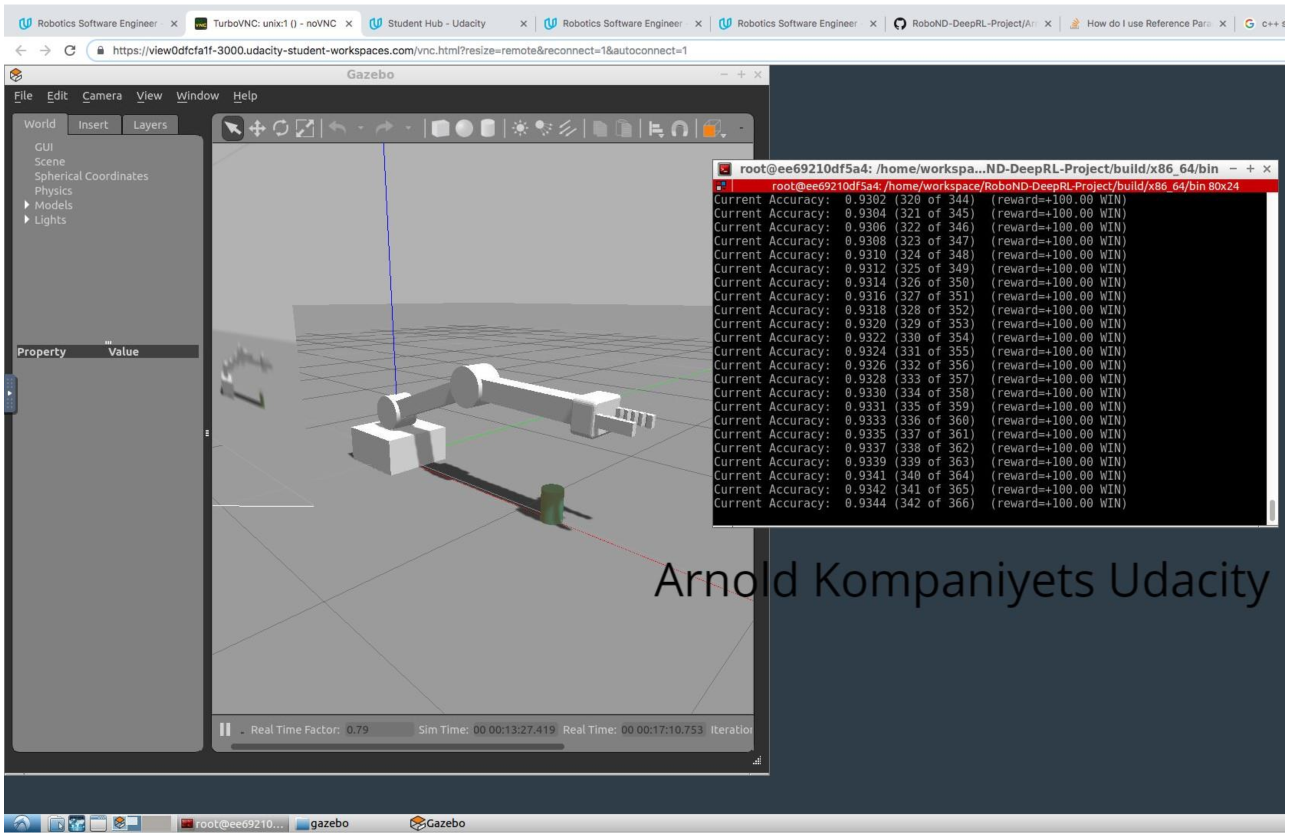
Continuing, the replay memory was kept at 10,000, and LSTM was turned on. Batch size and LSTM size were varied extensively, and it was discovered that the greatest impact was seen not purely from the absolute value of the numbers, but their relation to each other. The program seemed to work best when the LSTM value was greater than the batch size, by at least a few factors. Batch size was experimented with, from 10 to 256, with LSTM being altered from 50 to 512. The lower numbers seemed to produce the most jagged movements by the robot arm, while the higher numbers produced smooth movements, although movements too fast and inaccurate for proper learning. As such, a middle ground was chosen in making the batch size 64 and the LSTM size four times larger.

The one other parameter that was changed pertained to the "maxEpisodeLength". It was observed that a value of 100 was too short in the early training phase to allow the robot arm enough time learn desired movements. Therefore, it was doubled to 200.

<u>Results:</u>

<u>Objective 1:</u>

The following screenshot was attained when having the robot arm learn to touch the green tube with any of its parts:


Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Models
Lights
Property Value
Real Time Factor: 0.79
Sim Time: 00 00:13:27.419
Real Time: 00 00:17:10.753
Arnold Kompaniyets Udacity

```
root@ee69210df5a4: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.9302 (320 of 344)  (reward=+100.00 WIN)
Current Accuracy:  0.9304 (321 of 345)  (reward=+100.00 WIN)
Current Accuracy:  0.9306 (322 of 346)  (reward=+100.00 WIN)
Current Accuracy:  0.9308 (323 of 347)  (reward=+100.00 WIN)
Current Accuracy:  0.9310 (324 of 348)  (reward=+100.00 WIN)
Current Accuracy:  0.9312 (325 of 349)  (reward=+100.00 WIN)
Current Accuracy:  0.9314 (326 of 350)  (reward=+100.00 WIN)
Current Accuracy:  0.9316 (327 of 351)  (reward=+100.00 WIN)
Current Accuracy:  0.9318 (328 of 352)  (reward=+100.00 WIN)
Current Accuracy:  0.9320 (329 of 353)  (reward=+100.00 WIN)
Current Accuracy:  0.9322 (330 of 354)  (reward=+100.00 WIN)
Current Accuracy:  0.9324 (331 of 355)  (reward=+100.00 WIN)
Current Accuracy:  0.9326 (332 of 356)  (reward=+100.00 WIN)
Current Accuracy:  0.9328 (333 of 357)  (reward=+100.00 WIN)
Current Accuracy:  0.9330 (334 of 358)  (reward=+100.00 WIN)
Current Accuracy:  0.9331 (335 of 359)  (reward=+100.00 WIN)
Current Accuracy:  0.9333 (336 of 360)  (reward=+100.00 WIN)
Current Accuracy:  0.9335 (337 of 361)  (reward=+100.00 WIN)
Current Accuracy:  0.9337 (338 of 362)  (reward=+100.00 WIN)
Current Accuracy:  0.9339 (339 of 363)  (reward=+100.00 WIN)
Current Accuracy:  0.9341 (340 of 364)  (reward=+100.00 WIN)
Current Accuracy:  0.9342 (341 of 365)  (reward=+100.00 WIN)
Current Accuracy:  0.9344 (342 of 366)  (reward=+100.00 WIN)
```

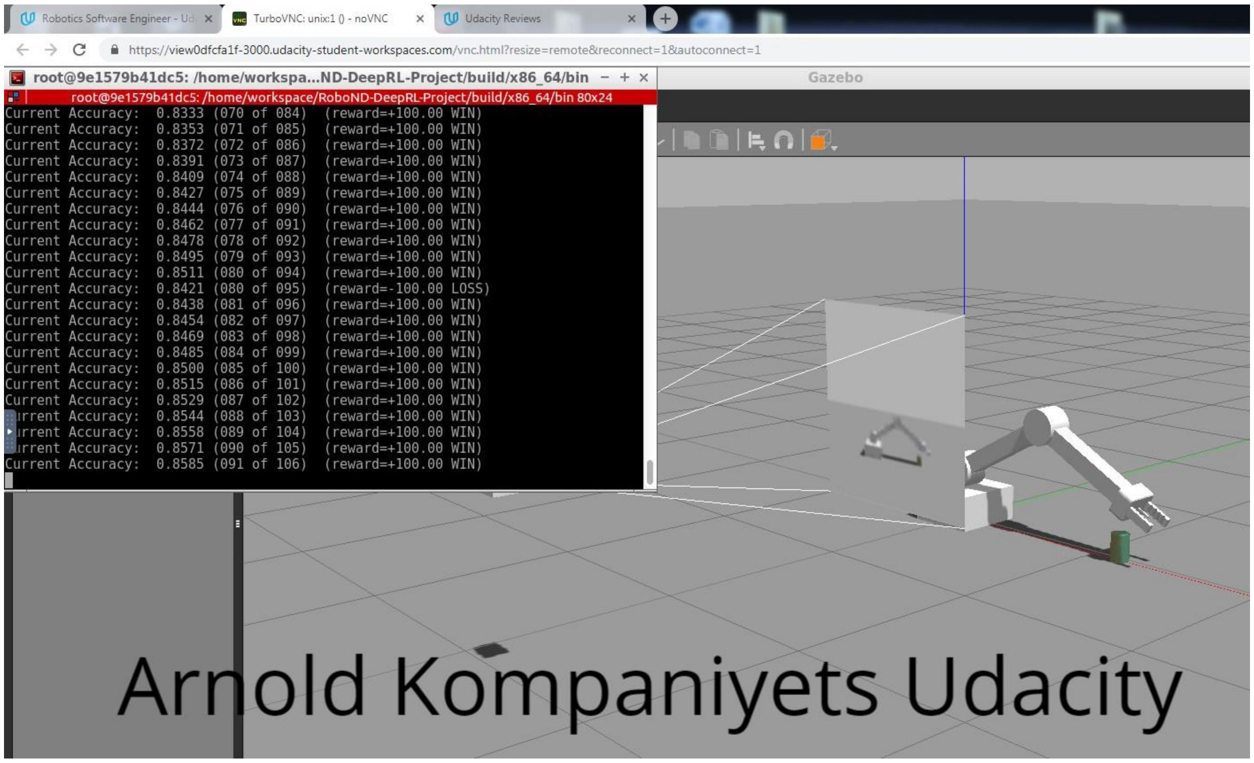
Within 80 or so runs, an accuracy level of 90%was successfully achieved, and this was maintained over the course of 366 total runs, with an accuracy of over 93% recorded.

In analyzing the learning network, the robot arm seemed to prefer to learn the simplest movement needed to achieve the reward. Therefore, the arm chose to move mainly only the first joint, directly lowering the arm onto the green tube. This resulted in mainly the second part of the arm being the point of contact.

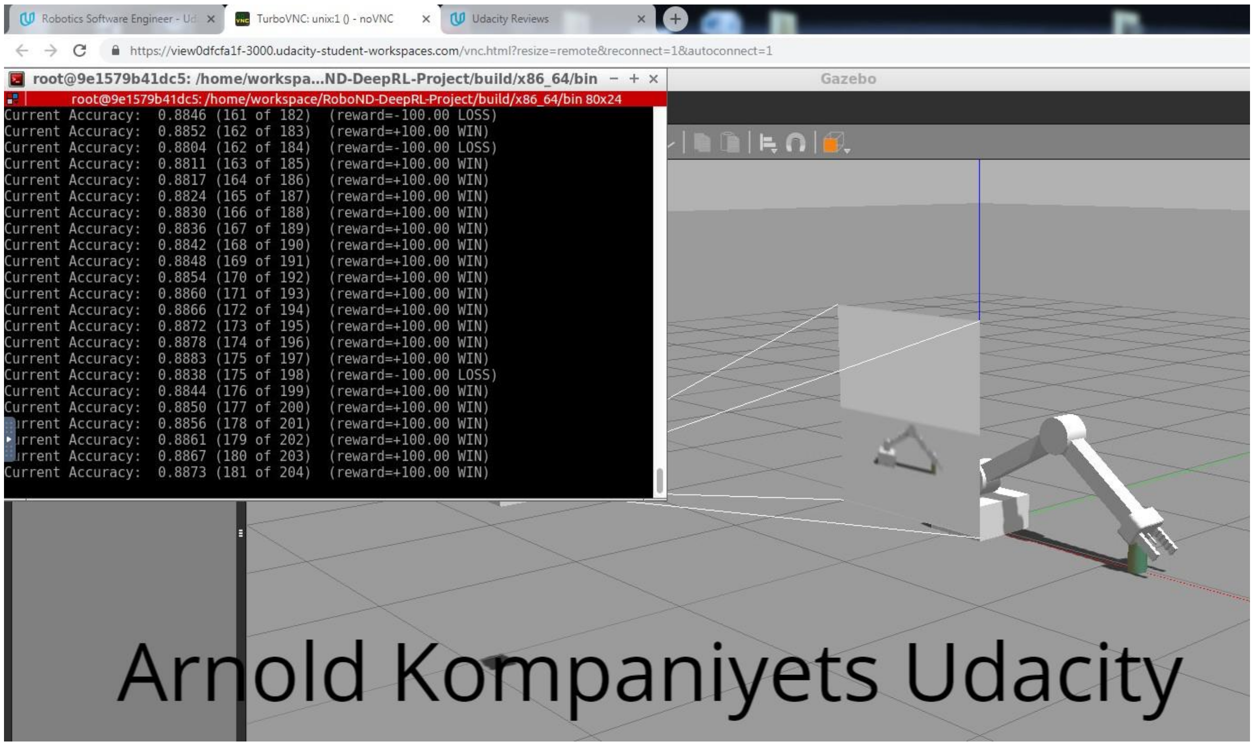
Given the reward and hyper-parameters above, the robot arm would replicate the desired movement quite quickly, settling on its desired path within 50 or so runs. The simplicity of the movement necessary to achieve the task seemed to be the primary component that allowed the network to learn quickly and keep training variability to a minimum.

Objective 2:

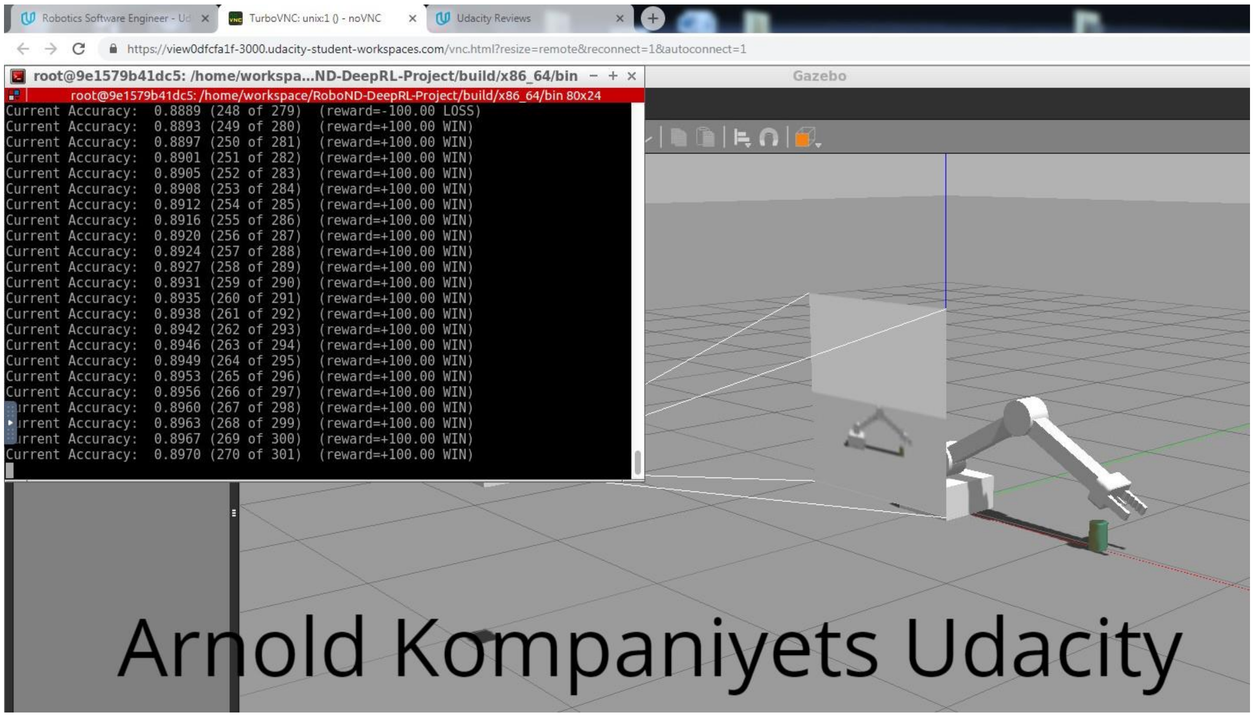
When training the arm to touch the green tube with only the gripper base, the following results were achieved:


Robotics Software Engineer - Ud
TurboVNC: unix:1 () - noVNC
Udacity Reviews
https://view0dfcfa1f-3000.udacity-student-workspaces.com/vnc.html?resize=remote&reconnect=1&autoconnect=1
Gazebo
Arnold Kompaniyets Udacity

```
root@9e1579b41dc5: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@9e1579b41dc5: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.8333 (070 of 084)  (reward=+100.00 WIN)
Current Accuracy:  0.8353 (071 of 085)  (reward=+100.00 WIN)
Current Accuracy:  0.8372 (072 of 086)  (reward=+100.00 WIN)
Current Accuracy:  0.8391 (073 of 087)  (reward=+100.00 WIN)
Current Accuracy:  0.8409 (074 of 088)  (reward=+100.00 WIN)
Current Accuracy:  0.8427 (075 of 089)  (reward=+100.00 WIN)
Current Accuracy:  0.8444 (076 of 090)  (reward=+100.00 WIN)
Current Accuracy:  0.8462 (077 of 091)  (reward=+100.00 WIN)
Current Accuracy:  0.8478 (078 of 092)  (reward=+100.00 WIN)
Current Accuracy:  0.8495 (079 of 093)  (reward=+100.00 WIN)
Current Accuracy:  0.8511 (080 of 094)  (reward=+100.00 WIN)
Current Accuracy:  0.8421 (080 of 095)  (reward=-100.00 LOSS)
Current Accuracy:  0.8438 (081 of 096)  (reward=+100.00 WIN)
Current Accuracy:  0.8454 (082 of 097)  (reward=+100.00 WIN)
Current Accuracy:  0.8469 (083 of 098)  (reward=+100.00 WIN)
Current Accuracy:  0.8485 (084 of 099)  (reward=+100.00 WIN)
Current Accuracy:  0.8500 (085 of 100)  (reward=+100.00 WIN)
Current Accuracy:  0.8515 (086 of 101)  (reward=+100.00 WIN)
Current Accuracy:  0.8529 (087 of 102)  (reward=+100.00 WIN)
urrent Accuracy:  0.8544 (088 of 103)  (reward=+100.00 WIN)
urrent Accuracy:  0.8558 (089 of 104)  (reward=+100.00 WIN)
urrent Accuracy:  0.8571 (090 of 105)  (reward=+100.00 WIN)
Current Accuracy:  0.8585 (091 of 106)  (reward=+100.00 WIN)
```


Robotics Software Engineer - Ud
TurboVNC: unix:1 () - noVNC
Udacity Reviews
https://view0dfcfa1f-3000.udacity-student-workspaces.com/vnc.html?resize=remote&reconnect=1&autoconnect=1
Gazebo
Arnold Kompaniyets Udacity

```
root@9e1579b41dc5: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@9e1579b41dc5: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.8846 (161 of 182)  (reward=-100.00 LOSS)
Current Accuracy:  0.8852 (162 of 183)  (reward=+100.00 WIN)
Current Accuracy:  0.8804 (162 of 184)  (reward=-100.00 LOSS)
Current Accuracy:  0.8811 (163 of 185)  (reward=+100.00 WIN)
Current Accuracy:  0.8817 (164 of 186)  (reward=+100.00 WIN)
Current Accuracy:  0.8824 (165 of 187)  (reward=+100.00 WIN)
Current Accuracy:  0.8830 (166 of 188)  (reward=+100.00 WIN)
Current Accuracy:  0.8836 (167 of 189)  (reward=+100.00 WIN)
Current Accuracy:  0.8842 (168 of 190)  (reward=+100.00 WIN)
Current Accuracy:  0.8848 (169 of 191)  (reward=+100.00 WIN)
Current Accuracy:  0.8854 (170 of 192)  (reward=+100.00 WIN)
Current Accuracy:  0.8860 (171 of 193)  (reward=+100.00 WIN)
Current Accuracy:  0.8866 (172 of 194)  (reward=+100.00 WIN)
Current Accuracy:  0.8872 (173 of 195)  (reward=+100.00 WIN)
Current Accuracy:  0.8878 (174 of 196)  (reward=+100.00 WIN)
Current Accuracy:  0.8883 (175 of 197)  (reward=+100.00 WIN)
Current Accuracy:  0.8838 (175 of 198)  (reward=-100.00 LOSS)
Current Accuracy:  0.8844 (176 of 199)  (reward=+100.00 WIN)
Current Accuracy:  0.8850 (177 of 200)  (reward=+100.00 WIN)
urrent Accuracy:  0.8856 (178 of 201)  (reward=+100.00 WIN)
urrent Accuracy:  0.8861 (179 of 202)  (reward=+100.00 WIN)
urrent Accuracy:  0.8867 (180 of 203)  (reward=+100.00 WIN)
Current Accuracy:  0.8873 (181 of 204)  (reward=+100.00 WIN)
```


Robotics Software Engineer - Ud
TurboVNC: unix:1 () - noVNC
Udacity Reviews
https://view0dfcfa1f-3000.udacity-student-workspaces.com/vnc.html?resize=remote&reconnect=1&autoconnect=1
Gazebo
Arnold Kompaniyets Udacity

```
root@9e1579b41dc5: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@9e1579b41dc5: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.8889 (248 of 279)  (reward=-100.00 LOSS)
Current Accuracy:  0.8893 (249 of 280)  (reward=+100.00 WIN)
Current Accuracy:  0.8897 (250 of 281)  (reward=+100.00 WIN)
Current Accuracy:  0.8901 (251 of 282)  (reward=+100.00 WIN)
Current Accuracy:  0.8905 (252 of 283)  (reward=+100.00 WIN)
Current Accuracy:  0.8908 (253 of 284)  (reward=+100.00 WIN)
Current Accuracy:  0.8912 (254 of 285)  (reward=+100.00 WIN)
Current Accuracy:  0.8916 (255 of 286)  (reward=+100.00 WIN)
Current Accuracy:  0.8920 (256 of 287)  (reward=+100.00 WIN)
Current Accuracy:  0.8924 (257 of 288)  (reward=+100.00 WIN)
Current Accuracy:  0.8927 (258 of 289)  (reward=+100.00 WIN)
Current Accuracy:  0.8931 (259 of 290)  (reward=+100.00 WIN)
Current Accuracy:  0.8935 (260 of 291)  (reward=+100.00 WIN)
Current Accuracy:  0.8938 (261 of 292)  (reward=+100.00 WIN)
Current Accuracy:  0.8942 (262 of 293)  (reward=+100.00 WIN)
Current Accuracy:  0.8946 (263 of 294)  (reward=+100.00 WIN)
Current Accuracy:  0.8949 (264 of 295)  (reward=+100.00 WIN)
Current Accuracy:  0.8953 (265 of 296)  (reward=+100.00 WIN)
Current Accuracy:  0.8956 (266 of 297)  (reward=+100.00 WIN)
urrent Accuracy:  0.8960 (267 of 298)  (reward=+100.00 WIN)
urrent Accuracy:  0.8963 (268 of 299)  (reward=+100.00 WIN)
urrent Accuracy:  0.8967 (269 of 300)  (reward=+100.00 WIN)
Current Accuracy:  0.8970 (270 of 301)  (reward=+100.00 WIN)
```

Within about 60 runs, the arm was able to achieve 80% accuracy, leading to 85% by run 100. Within 200 runs, this was raised to just under 89%, up to nearly 90% by run 300.

Training the robot arm to accomplish this objective proved to be far more tedious and variable. Now requiring the movement of multiple joints to achieve the proper reward value, the robot arm had much more possibilities for arm position and movement. As such, the robot arm success seemed to vary wildly on which positions the arm would experiment with first. Some of these positions would never result in a win, while others only sporadically. Changing the intermediate reward multiplier had the greatest effect on the arm performance, with multiple values showing capability of attaining the desired 80% success minimum. However, the severe lack of reproducibility led to further experimentation, with a reward multiplier of 9.0 showing the most consistent DQN success in learning the task.

Conclusion/ Future Work:

All in all, this project did an excellent job of illustrating how only a small change in the complexity of a task can have a profound effect on the success and variability of a DQN agent. With the regular Q-algorithm learning, extra joints and possible movements simply resulted in a new row of table values, increasing the complexity not quite linearly, but close to it. With deep RL, adding extra complexity appeared to make the task exponentially more difficult for the DQN agent, since the agent had to learn all possible venues of action prior to initiating a desired response. This particular point made it quite easy to see why training a DQN setup on easy 2-D problems was necessary, because an agent must be in immaculate condition to begin tackling more complex 3-D environments. The greater amount of freedom allowed for a DQN agent in a 3-D space inevitably results in more varied training cycles, leading to potentially extensive variability in performance from one DQN agent to the next – that is, unless the reward parameters are exquisitely tuned.

This last point of mention was rather interesting to discover. While reading the course material leading up to the project, DQN agents were described as having learning cycles and behaviors akin to human learning. This is an incredibly enticing statement, but not exactly one that turned to fruition in this project. Depending on what seemed quite minor changes in the reward functionality (such as changing the intermediate reward multiplier from 10 to 12), the robot arm would go through a cycle simply failing each episode with the same action. Another minor alteration in the reward multiplier (from 10 to 9), and suddenly the robot arm would begin to learn from its mistakes and go towards the correct motion. This need to so diligently control the underlying reward parameters of the DQN agent made the process feel much less akin to human learning and more like simply replacing one equation for another. The epitome of this, within this project, came when experimenting with extra code to both penalize the robot for hitting the ground with the second joint and provide extra reward for keeping said joint further away from the green tube (in an effort to motivate the agent to adopt the correct motion and pose). Besides this not quite working as well as expected, the larger reason for removing all this extra code was due to it all seeming rather counter-intuitive with the project goal. After all, the DQN agent is supposed to learn desired behaviors from only simply-defined goals, and yet here was further and further structured code to exactly define robot behavior.

Keeping all this in mind, thoughts went further on comparing the DQN agents to the human mind. It is certainly true that the human mind also operates on a reward basis (in its case through the implementation of various neuropeptides), and barring a few odd exceptions, this tends to work quite well in the process of behavioral learning. Therefore, attempting to have this in robotic learning seems like a fantastic idea; however, we are still one step removed from attaining a more equitable comparison. With the human mind, the reward functions are not made consciously, but are constantly altered in response to results and desired goals. The mind continues to receive environmental observations and reward values, in turn continually processing for optimal next actions, but at the same time there is a higher order neural network in charge of optimizing the reward values themselves, so as to align better with the consciously-derived goals. To have a DQN agent capable of doing this would be remarkable, because not only would it be able to optimize desired action, but also optimize the intermediate reward values to fall in line better with the initial goal set. This, of course, requires quite a bit more thinking to try and put forth a model of a higher order neural network to optimize reward functions based on a provided goal.

Nonetheless, robot learning is continually moving towards greater and greater similarity to the human mind, which should hopefully allow a more thorough and welcomed integration of robotics into the day-to-day functioning of the average person. Currently, image data is able to produce behavioral learning that can begin to look human-like, so hopefully, using sensory fusion to expand deep RL capabilities (such as how the human brain uses visual, but also auditory and tactile input data) will lead to even superior learning capabilities.