1 Introduction

The goal of the project is to create a DQN agent and define a reward system to teach a robotic arm to carry out two primary objectives:

- Have any part of the robot arm touch the object of interest, with at least a 90% accuracy.
- Have only the gripper base of the robot arm touch the object, with at least a 80% accuracy.

The project will be done using Gazebo simulated environment with a C++ plug-in to interface Gazebo Simulated Robot to the DQN agent. in order to achieve the objective few tasks are required to be completed:

- Subscribe to camera and collision topics published by Gazebo.
- Create the DQN Agent and pass all required parameters to it.
- Define position based control function for arm joints.
- Penalize robot gripper hitting the ground.
- Interim Reward/Penalize based on the arm distance to the object.
- Reward based on collision between the arm and the object.
- Tuning the hyperparameters.
- Reward based on collision between the arm's gripper base and the object.

Next sections will explain in details how the required tasks were achieved.

2 Reward Function

Deep Q-Network (DQN) output is usually mapped to a particular action, which, for this project, is the control of each joint for the simulated robotic arm. Control of the joint movements can be through velocity, position, or a mix of both. In case of this project position control approach was selected.

Reward system was designed to train the robot to have any part of the robot arm touch the object of interest in one attempt then have the gripper base of the robot arm touch the object in a second attempt. designed reward system is as described in below 1. Each episode is limited to a certain number of attempts, penalty will be issued if maximum length of the episode was reached without winning. a penalty will be issued and episode will be ended if robot arm touched ground. interim reward or interim penalty will be issued while robot arm is moving based on the distance from the object of interest. when collision happens with object a win reward is issued and episode is ended.

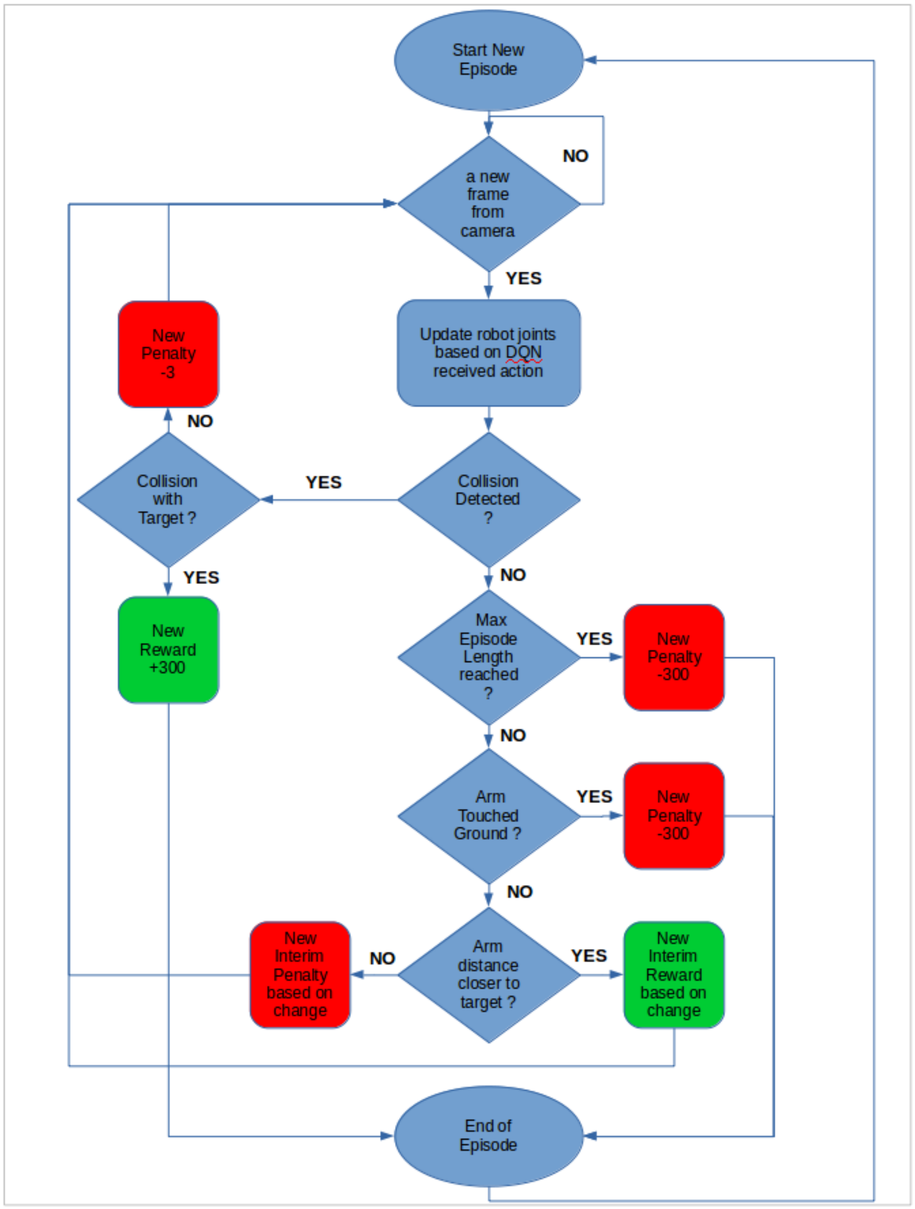


Figure 1: Reward System flowchart

Interim rewards are issued based on a smoothed moving average of the delta of the distance from the robot arm/gripper to the object of interest. It is calculated as following:
Distance Delta = last Distance to Goal - Current Distanced to Goal

Average Goal Delta = (Average Goal Delta * alpha) + (distance Delta * (1.0 - alpha))

Then interim reward is calculated based on the Average Goal Delta.

3 Hyper-parameters

Following parameters were adjusted before it was possible to get any good results from the DQN agent.

- **INPUT_WIDTH x INPUT_HEIGHT** : every camera frame, at every simulation iteration, is fed into the DQN agent then the agent makes a prediction and carries out an appropriate action. Size of the input or the dimension of the camera frame is decided by these two parameters. Bigger frame size will require both more memory and more computing power. 512x512 did not work well for example and caused memory related errors. **64x64** was fine and it did not cause issues or negative impact on accuracy.

- **NUM_ACTIONS** : based on the particular application, number of actions should be decided, in case of this project there are two actions per robot joint, one tto increase either the joint velocity or the joint position and the second is to decrease either the joint velocity or the joint position, total number of joints is degrees of freedom or DOF; so the total number of actions for the robot will be 2 x DOF = **6**.

- **OPTIMIZER** : There are many variations of gradient descent: Adam, RMSprop, Adagrad, etc. all let you set the learning rate. in this project RMSprop and Adam were tested and it produced similar results. **RMSprop** was used to obtain the required results in both Tasks 1 and 2.

- **LEARNING_RATE** : This parameter tells the optimizer how far to move the weights in the direction of the gradient for a batch. If the learning rate is low, then training is more reliable, but optimization will take a lot of time because steps towards the minimum of the loss function are tiny. If the learning rate is high, then training may not converge or even diverge. Weight changes can be so big that the optimizer overshoots the minimum and makes the loss worse. for task 1, 0.3 was worse than 0.2, 0.1 was too slow to converge however with value of **0.2** it was possible to reach the targeted accuracy. for Task 2 0.2 did not work well **0.01** was required to reach the target accuracy.

- **REPLAY_MEMORY** : A cyclic buffer that stores the transitions that the DQN agent observes for later reuse by sampling from it randomly. the transitions that build up a batch are decorrelated. It has been shown that this greatly stabilizes and improves the DQN training procedure. This number was selected to be **10000** which means it will be possible to store 10000/Batch_SIZE or 312 states that can be reused randomly. if number of joints are more and it is required to store more variety of states, then this number can be further increased.

- **BATCH_SIZE** : batch size * number of iterations = number of training examples, the bigger batch size the less number of iterations will be required but it will mean more memory and computing is required. 128, 64, and 32 was tested and it was found that **32** is best fit based on computer performance and available memory. This number will also impact REPLAY_MEMORY; if batch is bigger it will require bigger replay memory to store states.

- **USE_LSTM** : Enabling the (long Short Term Memory LSTM) as part of DQN network will allow training the network by taking into consideration multiple past frames from the camera sensor instead of a single frame. This will much improve the accuracy of learning. In this project LSTM was required and it was **enabled** to reach the required accuracy.

- **LSTM_SIZE** : Size of each LSTM cell, the bigger the size the more computing power will be required. 512 was tested but it was causing memory errors, **256** was better and it achieved the required target.

Following reward related parameters were also adjusted to fine tune the rewarding system:

- **REWARD_WIN** : number of points that will be issued in case of a successful arm/gripper touching the object of interest. in this project value was set to +300 that will be given when target is touched by robot.

- **REWARD_LOSS** : number of points that will be panelized in case robot touched the ground or exceeded the allowed number of iterations per episode. in this project -300 points will be deducted when the robot touch the ground or exceed the allowed limit of iterations per episode. in case of Task 2 10% of the -300 (or -30) will also be deducted if collision of the robot with the target object was not with robot gripper base.

- **REWARD_MULT** : multiplier used to control the amount of points given in each interim reward or penalty based in distance from object of interest. in this project multiplier was chosen to be 200 since the delta of distance was ranging from 0.4 to 1.4 so the rewards was ranging from 80 to 280. This can be changed based on the distance of the target from the robot arm.

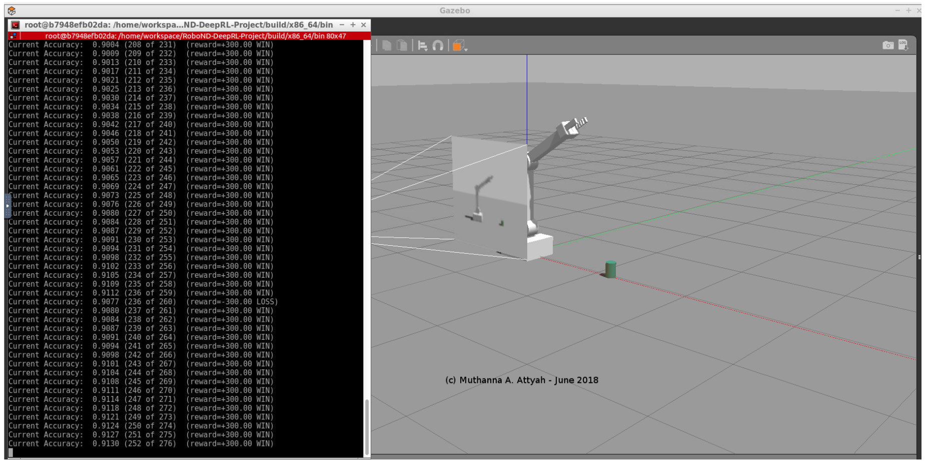
  Considering the fact that this project tasks are Episodic; the above rewards parameters was chosen so that goal of touching the target object can be framed as the maximization of (expected) cumulative reward.

- **alpha** : smoothing factor to control average distance. alpha was decided to be 0.3 based on what was mentioned in project details.

Parameters INPUT_CHANNELS=3, ALLOW_RANDOM=true, DEBUG_DQN=false, GAMMA=0.9f, EPS_START=0.9f, EPS_END=0.05f, and EPS_DECAY=200 were provided as part of the project code and it was left unchanged.

4 Results

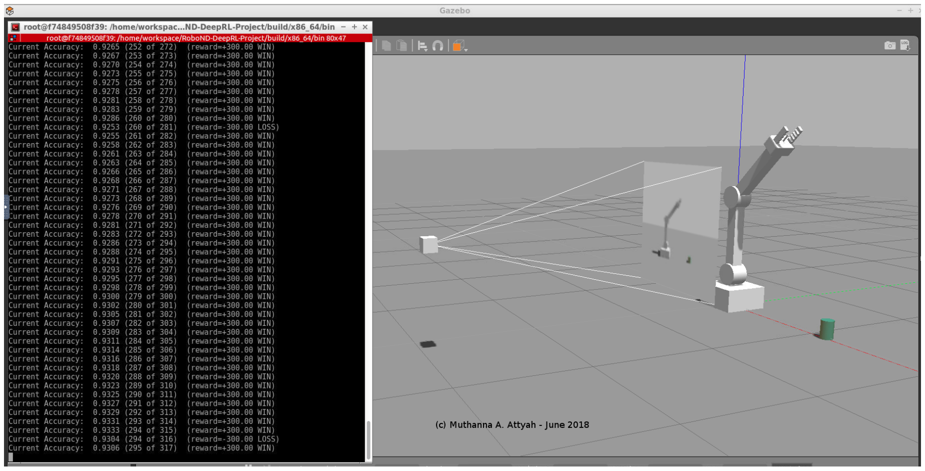
Before fine tuning the hyper-parameters and setting the right reward system; DQN network was taking longer time to reach the objective, accuracy was very low, and in many cases system was stopping with memory related errors. However after the fine tuning as described in above section it was possible to cross 92% accuracy within less than 300 cycles and the accuracy was still increasing. The same great performance was achievable in every run which confirms that system was well tuned and stable as shown in 2 and 3 figures.



```
root@b7948efb02da: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@b7948efb02da: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x47
Current Accuracy:  0.9004 (208 of 231)  (reward=+300.00 WIN)
Current Accuracy:  0.9009 (209 of 232)  (reward=+300.00 WIN)
Current Accuracy:  0.9013 (210 of 233)  (reward=+300.00 WIN)
Current Accuracy:  0.9017 (211 of 234)  (reward=+300.00 WIN)
Current Accuracy:  0.9021 (212 of 235)  (reward=+300.00 WIN)
Current Accuracy:  0.9025 (213 of 236)  (reward=+300.00 WIN)
Current Accuracy:  0.9030 (214 of 237)  (reward=+300.00 WIN)
Current Accuracy:  0.9034 (215 of 238)  (reward=+300.00 WIN)
Current Accuracy:  0.9038 (216 of 239)  (reward=+300.00 WIN)
Current Accuracy:  0.9042 (217 of 240)  (reward=+300.00 WIN)
Current Accuracy:  0.9046 (218 of 241)  (reward=+300.00 WIN)
Current Accuracy:  0.9050 (219 of 242)  (reward=+300.00 WIN)
Current Accuracy:  0.9053 (220 of 243)  (reward=+300.00 WIN)
Current Accuracy:  0.9057 (221 of 244)  (reward=+300.00 WIN)
Current Accuracy:  0.9061 (222 of 245)  (reward=+300.00 WIN)
Current Accuracy:  0.9065 (223 of 246)  (reward=+300.00 WIN)
Current Accuracy:  0.9069 (224 of 247)  (reward=+300.00 WIN)
Current Accuracy:  0.9073 (225 of 248)  (reward=+300.00 WIN)
Current Accuracy:  0.9076 (226 of 249)  (reward=+300.00 WIN)
Current Accuracy:  0.9080 (227 of 250)  (reward=+300.00 WIN)
Current Accuracy:  0.9084 (228 of 251)  (reward=+300.00 WIN)
Current Accuracy:  0.9087 (229 of 252)  (reward=+300.00 WIN)
Current Accuracy:  0.9091 (230 of 253)  (reward=+300.00 WIN)
Current Accuracy:  0.9094 (231 of 254)  (reward=+300.00 WIN)
Current Accuracy:  0.9098 (232 of 255)  (reward=+300.00 WIN)
Current Accuracy:  0.9102 (233 of 256)  (reward=+300.00 WIN)
Current Accuracy:  0.9105 (234 of 257)  (reward=+300.00 WIN)
Current Accuracy:  0.9109 (235 of 258)  (reward=+300.00 WIN)
Current Accuracy:  0.9112 (236 of 259)  (reward=+300.00 WIN)
Current Accuracy:  0.9077 (236 of 260)  (reward=-300.00 LOSS)
Current Accuracy:  0.9080 (237 of 261)  (reward=+300.00 WIN)
Current Accuracy:  0.9084 (238 of 262)  (reward=+300.00 WIN)
Current Accuracy:  0.9087 (239 of 263)  (reward=+300.00 WIN)
Current Accuracy:  0.9091 (240 of 264)  (reward=+300.00 WIN)
Current Accuracy:  0.9094 (241 of 265)  (reward=+300.00 WIN)
Current Accuracy:  0.9098 (242 of 266)  (reward=+300.00 WIN)
Current Accuracy:  0.9101 (243 of 267)  (reward=+300.00 WIN)
Current Accuracy:  0.9104 (244 of 268)  (reward=+300.00 WIN)
Current Accuracy:  0.9108 (245 of 269)  (reward=+300.00 WIN)
Current Accuracy:  0.9111 (246 of 270)  (reward=+300.00 WIN)
Current Accuracy:  0.9114 (247 of 271)  (reward=+300.00 WIN)
Current Accuracy:  0.9118 (248 of 272)  (reward=+300.00 WIN)
Current Accuracy:  0.9121 (249 of 273)  (reward=+300.00 WIN)
Current Accuracy:  0.9124 (250 of 274)  (reward=+300.00 WIN)
Current Accuracy:  0.9127 (251 of 275)  (reward=+300.00 WIN)
Current Accuracy:  0.9130 (252 of 276)  (reward=+300.00 WIN)
```

Figure 2: Results of Robot Arm touching the object

During the test once the final tuned parameters were set, there was no need to re-tune the parameters after changing the objective from ”arm touching the object” to ”gripper base touching the object”. only Learning_Rate was changed to 0.01 and the DQN performance was excellent and stable to handle both cases.



```
root@f74849508f39: /home/workspac...ND-DeepRL-Project/build/x86_64/bin
root@f74849508f39: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x47
Current Accuracy:  0.9265 (252 of 272)  (reward=+300.00 WIN)
Current Accuracy:  0.9267 (253 of 273)  (reward=+300.00 WIN)
Current Accuracy:  0.9270 (254 of 274)  (reward=+300.00 WIN)
Current Accuracy:  0.9273 (255 of 275)  (reward=+300.00 WIN)
Current Accuracy:  0.9275 (256 of 276)  (reward=+300.00 WIN)
Current Accuracy:  0.9278 (257 of 277)  (reward=+300.00 WIN)
Current Accuracy:  0.9281 (258 of 278)  (reward=+300.00 WIN)
Current Accuracy:  0.9283 (259 of 279)  (reward=+300.00 WIN)
Current Accuracy:  0.9286 (260 of 280)  (reward=+300.00 WIN)
Current Accuracy:  0.9253 (260 of 281)  (reward=-300.00 LOSS)
Current Accuracy:  0.9255 (261 of 282)  (reward=+300.00 WIN)
Current Accuracy:  0.9258 (262 of 283)  (reward=+300.00 WIN)
Current Accuracy:  0.9261 (263 of 284)  (reward=+300.00 WIN)
Current Accuracy:  0.9263 (264 of 285)  (reward=+300.00 WIN)
Current Accuracy:  0.9266 (265 of 286)  (reward=+300.00 WIN)
Current Accuracy:  0.9268 (266 of 287)  (reward=+300.00 WIN)
Current Accuracy:  0.9271 (267 of 288)  (reward=+300.00 WIN)
Current Accuracy:  0.9273 (268 of 289)  (reward=+300.00 WIN)
Current Accuracy:  0.9276 (269 of 290)  (reward=+300.00 WIN)
Current Accuracy:  0.9278 (270 of 291)  (reward=+300.00 WIN)
Current Accuracy:  0.9281 (271 of 292)  (reward=+300.00 WIN)
Current Accuracy:  0.9283 (272 of 293)  (reward=+300.00 WIN)
Current Accuracy:  0.9286 (273 of 294)  (reward=+300.00 WIN)
Current Accuracy:  0.9288 (274 of 295)  (reward=+300.00 WIN)
Current Accuracy:  0.9291 (275 of 296)  (reward=+300.00 WIN)
Current Accuracy:  0.9293 (276 of 297)  (reward=+300.00 WIN)
Current Accuracy:  0.9295 (277 of 298)  (reward=+300.00 WIN)
Current Accuracy:  0.9298 (278 of 299)  (reward=+300.00 WIN)
Current Accuracy:  0.9300 (279 of 300)  (reward=+300.00 WIN)
Current Accuracy:  0.9302 (280 of 301)  (reward=+300.00 WIN)
Current Accuracy:  0.9305 (281 of 302)  (reward=+300.00 WIN)
Current Accuracy:  0.9307 (282 of 303)  (reward=+300.00 WIN)
Current Accuracy:  0.9309 (283 of 304)  (reward=+300.00 WIN)
Current Accuracy:  0.9311 (284 of 305)  (reward=+300.00 WIN)
Current Accuracy:  0.9314 (285 of 306)  (reward=+300.00 WIN)
Current Accuracy:  0.9316 (286 of 307)  (reward=+300.00 WIN)
Current Accuracy:  0.9318 (287 of 308)  (reward=+300.00 WIN)
Current Accuracy:  0.9320 (288 of 309)  (reward=+300.00 WIN)
Current Accuracy:  0.9323 (289 of 310)  (reward=+300.00 WIN)
Current Accuracy:  0.9325 (290 of 311)  (reward=+300.00 WIN)
Current Accuracy:  0.9327 (291 of 312)  (reward=+300.00 WIN)
Current Accuracy:  0.9329 (292 of 313)  (reward=+300.00 WIN)
Current Accuracy:  0.9331 (293 of 314)  (reward=+300.00 WIN)
Current Accuracy:  0.9333 (294 of 315)  (reward=+300.00 WIN)
Current Accuracy:  0.9304 (294 of 316)  (reward=-300.00 LOSS)
Current Accuracy:  0.9306 (295 of 317)  (reward=+300.00 WIN)
```

Figure 3: Results of Robot Gripper touching the object

5 Future Work

After fine tuning the hyper-parameters results was quite encouraging, however still there is a room for improvement. One suggested approach is to graph the accuracy progress for every change in parameters then identify the maximum accuracy that can be achieved and how many iterations (learning time) it will take to reach it.

DQN can be also used by itself to fine tune some of the parameters of another DQN where reward can be issued to the first DQN based on obtained accuracy from the second DQN.

if there is a good computing power and memory to use; it is possible to use a higher input number for example 512x512 and train the DQN to play a more complicated game such as classical DOOM. Rewards will be based on the game score and actions will include shooter movement and shooting.