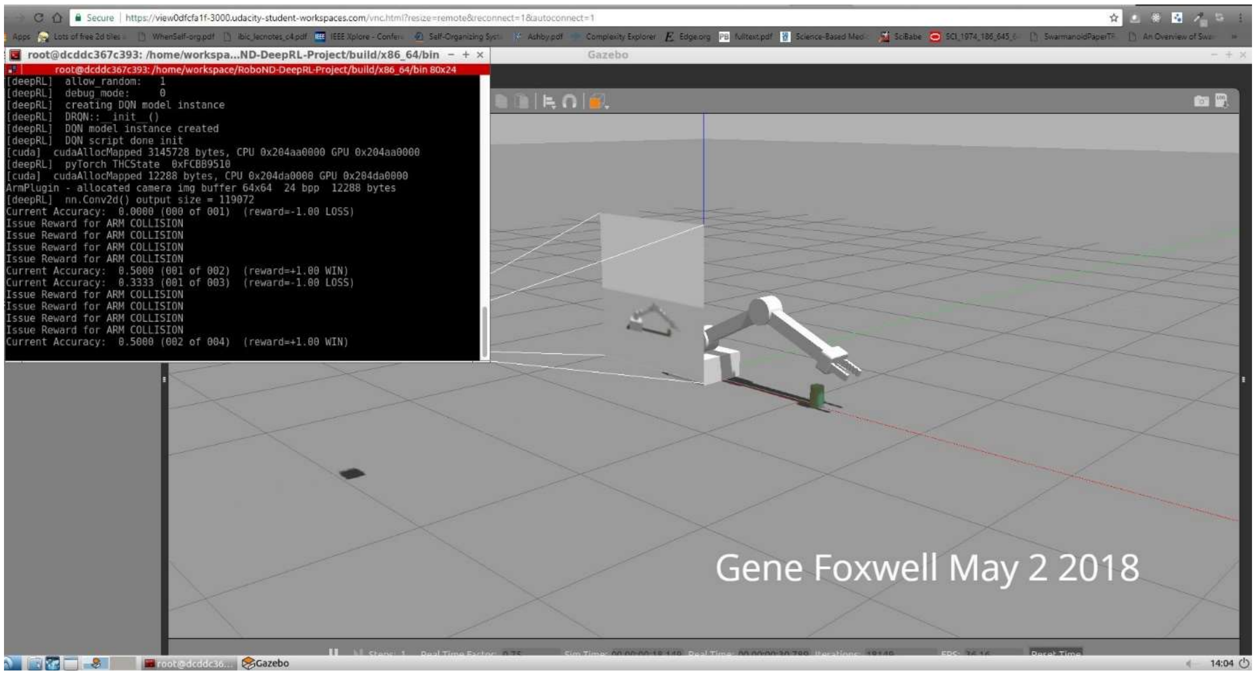
# Deep RL Manipulator

*Gene Foxwell; Udacity Robotics Software Engineer Project*

## Abstract

This project trains a simulated three degree of freedom manipulator to pick up a cylindrical object. We used Deep RL in conjunction with gazebo to train the robotic arm using a simulated camera for inputs. After training the arm we were able to get the arm to contact the cylindrical object in at least 90% of cases and to position the gripper at the object in 80% of test cases.



```
[deepRL]  allow_random:   1
[deepRL]  debug_mode:     0
[deepRL]  creating DQN model instance
[deepRL]  DRQN::__init__()
[deepRL]  DQN model instance created
[deepRL]  DQN script done init
[cuda]  cudaAllocMapped 3145728 bytes, CPU 0x204aa0000 GPU 0x204aa0000
[deepRL]  pyTorch THCState  0xFCBB9510
[cuda]  cudaAllocMapped 12288 bytes, CPU 0x204da0000 GPU 0x204da0000
ArmPlugin - allocated camera img buffer 64x64  24 bpp  12288 bytes
[deepRL]  nn.Conv2d() output size = 119072
Current Accuracy:  0.0000 (000 of 001)  (reward=-1.00 LOSS)
Issue Reward for ARM COLLISION
Issue Reward for ARM COLLISION
Issue Reward for ARM COLLISION
Issue Reward for ARM COLLISION
Current Accuracy:  0.5000 (001 of 002)  (reward=+1.00 WIN)
Current Accuracy:  0.3333 (001 of 003)  (reward=-1.00 LOSS)
Issue Reward for ARM COLLISION
Issue Reward for ARM COLLISION
Issue Reward for ARM COLLISION
Issue Reward for ARM COLLISION
Current Accuracy:  0.5000 (002 of 004)  (reward=+1.00 WIN)
```

*Figure 1 Gazebo Simulation for Deep RL*

## Reward Function and Hyperparameters

We trained a Udacity provided DQN Agent to solve two related tasks using Deep Reinforcement Learning. The agent controlled a simulated robot by updating the various joint velocities of the arm – an image of this robot in action can be seen in Figure 1. Two actions where associated with each joint, one action to increase the joints angular velocity, one action to decrease it. Input for the agent was provided by a simulated camera in the gazebo world situated to capture both the robot arm and the goal cylinder.

Our tasks where to train the agent to touch the cylinder with any part of the arm at an accuracy of 90% after at least 100 iterations. Our second task was an extension of the first, where instead of just allowing any part of the arm to touch the cylinder we are restricting the agent to touching the cylinder with the gripper only. Our target was to obtain an accuracy of 80% at this task after at least 100 iterations.

For each task a different set of reward functions and hyperparameters was used. These parameters are summarized in the subsections below.

## Task 1 – Get any part of the arm to touch the cylinder

### Reward Function

For the task of getting any part of the arm to touch the cylinder, we designed the reward function in several parts.

1. **Number of Episodes Exceeded:** If the arm has not completed its episode after the maximum number of iterations, we are providing a penalty of -200. This large penalty was chosen to discourage the algorithm from getting stuck in solutions that appear to be exploiting the interim rewards. (i.e. the algorithm should never get enough reward from the interim rewards that letting its episodes run over appears to be a valid solution).
2. **Gripper Contacts Ground:** We provide a small negative reward for the gripper touching the ground. This reward is proportional to the distance between the point that the gripper touches the ground and the position of the cylinder. This was chosen to encourage the algorithm to explore positions closer and closer to the cylinder. Touching the ground with the gripper also terminates the episode. The equation used here is: **-1.0 + exp(-distGoal)**, where distGoal is the distance from the gripper to the goal cylinder.
3. **Interim Reward:** We provide a small positive reward at each iteration based on the how far the gripper is from the goal point. We increase this reward exponentially as the gripper gets closer to the goal. The intuition behind this was that we want to encourage the algorithm to continue to move towards the goal point by providing a clear set of increasing goal values between any gripper position and the goal. On the flip side, we don't really want to reward the algorithm for moving too far away from the goal, so a constant was used to set to offset the positive reward. The equation used here was: **exp(-distGoal) - 0.5f**.
4. **Arm Touches Cylinder**: A high reward of 110 was given any time a part of the arm encountered the goal. This reward was set high to encourage the arm to choose to touch the goal rather than try to maximize interim rewards.

### Hyperparameters

Input width and height were reduced from their default values of 512 down to 256 for this experiment. It was found that when left at the default values the CUDA system would run out of memory at about 25 – 30 episodes, which was not nearly enough training time to complete the experiment.

To this end, REPLAY_MEMORY, BATCH_SIZE, and LTSM_SIZE where also reduced, to ensure that we did not run into memory restrictions during the training process. Experiments where performed with the LTSM turned off, but these did not appear to get the same quality of results as simply leaving it on.

| Parameter | Description | Value |
|---|---|---|
| **INPUT_WIDTH** | The width of the image we are using to train the DQN | 256 |
| **INPUT_HEIGHT** | The height of the image we are using to train the DQN | 256 |
| **OPTIMIZER** | The optimization function used when training the DQN. | RMSprop |
| **LEARNING_RATE** | Learning rate for training the DQN | 0.01 |
| **REPLAY_MEMORY** | The number of previous events to store in the Experience Replay buffer. | 2000 |
| **BATCH_SIZE** | The number of training examples to sample from the Replay Buffer per training iteration. | 4 |
| **USE_LTSM** | Whether or not to use LTSM (Long Term Short Term Memory Nodes) in the DQN. | True |
| **LTSM_SIZE** | Number of LTSM nodes to use in the DQN. | 8 |
| **DISTANCE_ALPHA** | An auxiliary constant used to calculate the running average of the arms distance from the cylinder. | 0.2 |

## TASK 2 – GRIPPER ONLY CONTACT WITH THE CYLINDER

This task proved to be significantly more challenging than the previous one – which prompted many alterations to the original solution used for Task 1.

### Reward Function

As before, the reward function for this task can be broken down into several parts:

1. **Gripper Contacts Cylinder:** Here we are providing a high reward (REWARD_WIN = 400) plus the number of iterations remaining in the current episode. This "bonus" reward is given to encourage the robot to find efficient solutions to the problem at hand.
2. **Non-Gripper Arm Contact with Cylinder:** Here the reward was a composite of the REWARD_LOSS (-2.0) and the average velocity towards the goal. (**REWARD_LOSS + avgGoalDelta**) The thinking behind this was that while we absolutely want the arm to learn not to touch the cylinder with the non-gripper section, some movements should be preferred over others. Specifically, we want to reward attempts that were moving towards to goal at a reasonable speed over those where the arm simply flattens itself out and "coasts" down on the goal.

3. **Number of Episodes Exceeded:** Unlike before, we are not using a significantly higher number here, simply **2*REWARD_LOSS**. The difference between this reward and other areas where a REWARD_LOSS is issued is intended to signify that simply waiting around and doing nothing is worse than at least trying at gripping the cylinder.
4. **Gripper Contacts the Ground:** The reward function for this scenario was designed to lead a trail of "breadcrumbs" to the cylinder, much like in the previous task. Here however, we used negative rewards of the shape REWARD_LOSS – REWARD_LOSS * exp(-distGoal). As before, the intuition is that lower negative rewards closer to the cylinder would help guide the gripper in the correct direction.
5. **Interim Reward:** A completely different reward function from before was used here. For the interim reward in this case we used the Udacity suggested average velocity to the goal function: **(avgGoalDelta * DISTANCE_ALPHA) + (delta * (1 - DISTANCE_ALPHA)).** DISTANCE_ALPHA in this case was set to: 0.2f. This was used under the assumption that it would give the robot a bit more incentive to keep moving the arm forward at a reasonable velocity. While the previous tasks interim reward only rewarded reducing the distance to the goal, regardless of the arms velocity.

## Hyperparameters

Some important changes were made to the hyperparameters for this task. First, the input size was reduced significantly. This was done to mitigate some out of memory errors that would occur during training if the system took too many episodes to converge. Secondly the batch size and LTSM sizes where increased as experimentation showed this to be helpful. Finally, the EPS_END constant was tuned for this task, this was done to avoid instabilities that would occasionally show up during the training process which would cause the system to seemingly "unlearn" its behavior for short periods of time.

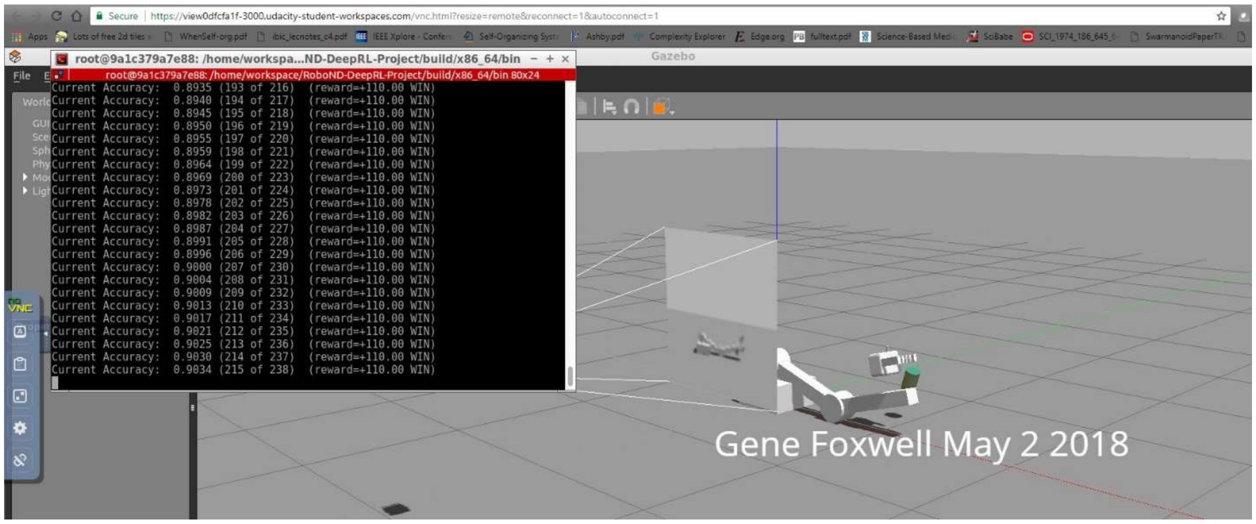
| Parameter | Description | Value |
|---|---|---|
| **INPUT_WIDTH** | The width of the image we are using to train the DQN. | 64 |
| **INPUT_HEIGHT** | The height of the image we are using to train the DQN. | 64 |
| **OPTIMIZER** | The optimization function used when training the DQN. | RMSprop |
| **LEARNING_RATE** | Learning rate for training the DQN | 0.01 |
| **REPLAY_MEMORY** | The number of previous events to store in the Experience Replay buffer. | 10000 |
| **BATCH_SIZE** | The number of training examples to sample from the | 128 |

| | | |
|---|---|---|
| | Replay Buffer per training iteration. | |
| **USE_LTSM** | Whether or not to use LTSM (Long Term Short Term Memory Nodes) in the DQN. | True |
| **LTSM_SIZE** | Number of LTSM nodes to use in the DQN. | 16 |
| **DISTANCE_ALPHA** | An auxiliary constant used to calculate the running average of the arms distance from the cylinder. | 0.2 |
| **EPS_END** | The lowest allowed value for the probability of "exploration". As this system seemed particularly unstable once trained, this value was reduced from its default of 0.05 to 0.01 | 0.01 |

## RESULTS

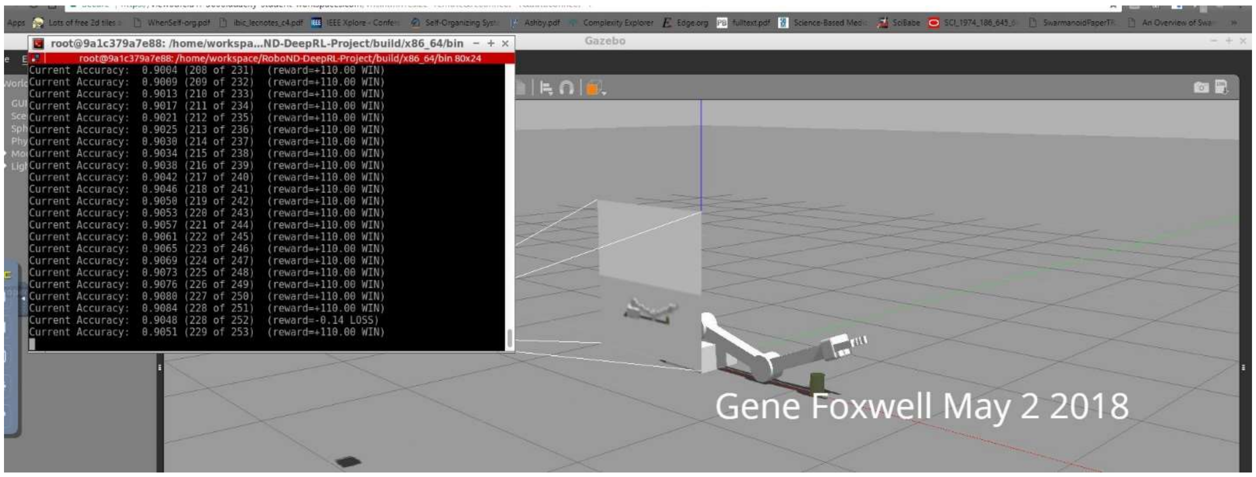
### TASK 1

In this task the robot arm learned to touch the cylinder with arm by slightly bending at the middle joint and then allowing the rest of the arm to simply "flop" on top of the cylinder. This behavior generally resulted in a win, but also tended to send the cylinder spinning about the simulation until everything reset.  Examples of this behavior can be seen in Figure 2 and Figure 3.  This behavior was achieved with 90% accuracy after 207 training episodes.   (Results can also be seen in Figures 1 and 2).



```
root@9a1c379a7e88: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@9a1c379a7e88: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.8935 (193 of 216)  (reward=+110.00 WIN)
Current Accuracy:  0.8940 (194 of 217)  (reward=+110.00 WIN)
Current Accuracy:  0.8945 (195 of 218)  (reward=+110.00 WIN)
Current Accuracy:  0.8950 (196 of 219)  (reward=+110.00 WIN)
Current Accuracy:  0.8955 (197 of 220)  (reward=+110.00 WIN)
Current Accuracy:  0.8959 (198 of 221)  (reward=+110.00 WIN)
Current Accuracy:  0.8964 (199 of 222)  (reward=+110.00 WIN)
Current Accuracy:  0.8969 (200 of 223)  (reward=+110.00 WIN)
Current Accuracy:  0.8973 (201 of 224)  (reward=+110.00 WIN)
Current Accuracy:  0.8978 (202 of 225)  (reward=+110.00 WIN)
Current Accuracy:  0.8982 (203 of 226)  (reward=+110.00 WIN)
Current Accuracy:  0.8987 (204 of 227)  (reward=+110.00 WIN)
Current Accuracy:  0.8991 (205 of 228)  (reward=+110.00 WIN)
Current Accuracy:  0.8996 (206 of 229)  (reward=+110.00 WIN)
Current Accuracy:  0.9000 (207 of 230)  (reward=+110.00 WIN)
Current Accuracy:  0.9004 (208 of 231)  (reward=+110.00 WIN)
Current Accuracy:  0.9009 (209 of 232)  (reward=+110.00 WIN)
Current Accuracy:  0.9013 (210 of 233)  (reward=+110.00 WIN)
Current Accuracy:  0.9017 (211 of 234)  (reward=+110.00 WIN)
Current Accuracy:  0.9021 (212 of 235)  (reward=+110.00 WIN)
Current Accuracy:  0.9025 (213 of 236)  (reward=+110.00 WIN)
Current Accuracy:  0.9030 (214 of 237)  (reward=+110.00 WIN)
Current Accuracy:  0.9034 (215 of 238)  (reward=+110.00 WIN)
```

*Figure 2 Arm after touching cylinder*

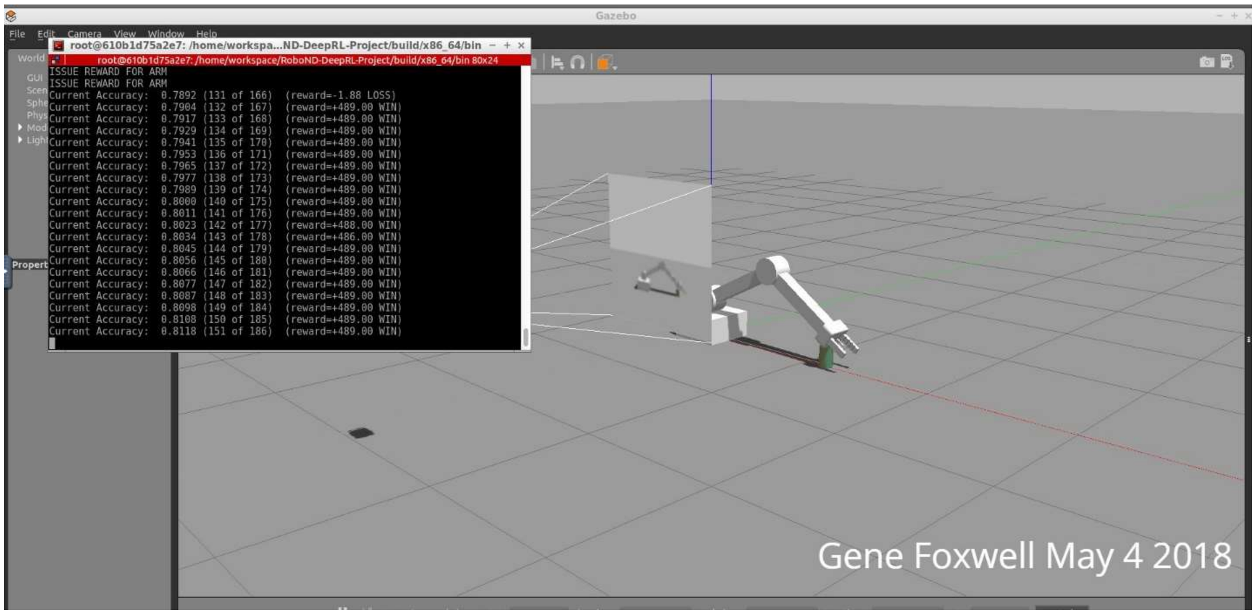


```
root@9a1c379a7e88: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@9a1c379a7e88: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Current Accuracy:  0.9004 (208 of 231)  (reward=+110.00 WIN)
Current Accuracy:  0.9009 (209 of 232)  (reward=+110.00 WIN)
Current Accuracy:  0.9013 (210 of 233)  (reward=+110.00 WIN)
Current Accuracy:  0.9017 (211 of 234)  (reward=+110.00 WIN)
Current Accuracy:  0.9021 (212 of 235)  (reward=+110.00 WIN)
Current Accuracy:  0.9025 (213 of 236)  (reward=+110.00 WIN)
Current Accuracy:  0.9030 (214 of 237)  (reward=+110.00 WIN)
Current Accuracy:  0.9034 (215 of 238)  (reward=+110.00 WIN)
Current Accuracy:  0.9038 (216 of 239)  (reward=+110.00 WIN)
Current Accuracy:  0.9042 (217 of 240)  (reward=+110.00 WIN)
Current Accuracy:  0.9046 (218 of 241)  (reward=+110.00 WIN)
Current Accuracy:  0.9050 (219 of 242)  (reward=+110.00 WIN)
Current Accuracy:  0.9053 (220 of 243)  (reward=+110.00 WIN)
Current Accuracy:  0.9057 (221 of 244)  (reward=+110.00 WIN)
Current Accuracy:  0.9061 (222 of 245)  (reward=+110.00 WIN)
Current Accuracy:  0.9065 (223 of 246)  (reward=+110.00 WIN)
Current Accuracy:  0.9069 (224 of 247)  (reward=+110.00 WIN)
Current Accuracy:  0.9073 (225 of 248)  (reward=+110.00 WIN)
Current Accuracy:  0.9076 (226 of 249)  (reward=+110.00 WIN)
Current Accuracy:  0.9080 (227 of 250)  (reward=+110.00 WIN)
Current Accuracy:  0.9084 (228 of 251)  (reward=+110.00 WIN)
Current Accuracy:  0.9048 (228 of 252)  (reward=-0.14 LOSS)
Current Accuracy:  0.9051 (229 of 253)  (reward=+110.00 WIN)
```

*Figure 3 Arm just about to touch cylinder*

## TASK 2

The robot converged to a straightforward solution to the problem.  It would bend its arm inwards, then arc down towards the cylinder until the gripper came into contact and the reward was received.  Having learned this solution, the arm was able to achieve an accuracy of 81% after 186 episodes.  Training was discontinued after this point to save on GPU cycles.  A screenshot of the results for this task can be seen in Figure 4.


Gazebo
File Edit Camera View Window Help
World
GUI
Scen
Sphe
Phys
Mod
Light
Propert
root@610b1d75a2e7: /home/workspa...ND-DeepRL-Project/build/x86_64/bin
root@610b1d75a2e7: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x24
Gene Foxwell May 4 2018

```
ISSUE REWARD FOR ARM
ISSUE REWARD FOR ARM
Current Accuracy:  0.7892 (131 of 166)  (reward=-1.88 LOSS)
Current Accuracy:  0.7904 (132 of 167)  (reward=+489.00 WIN)
Current Accuracy:  0.7917 (133 of 168)  (reward=+489.00 WIN)
Current Accuracy:  0.7929 (134 of 169)  (reward=+489.00 WIN)
Current Accuracy:  0.7941 (135 of 170)  (reward=+489.00 WIN)
Current Accuracy:  0.7953 (136 of 171)  (reward=+489.00 WIN)
Current Accuracy:  0.7965 (137 of 172)  (reward=+489.00 WIN)
Current Accuracy:  0.7977 (138 of 173)  (reward=+489.00 WIN)
Current Accuracy:  0.7989 (139 of 174)  (reward=+489.00 WIN)
Current Accuracy:  0.8000 (140 of 175)  (reward=+489.00 WIN)
Current Accuracy:  0.8011 (141 of 176)  (reward=+489.00 WIN)
Current Accuracy:  0.8023 (142 of 177)  (reward=+488.00 WIN)
Current Accuracy:  0.8034 (143 of 178)  (reward=+486.00 WIN)
Current Accuracy:  0.8045 (144 of 179)  (reward=+489.00 WIN)
Current Accuracy:  0.8056 (145 of 180)  (reward=+489.00 WIN)
Current Accuracy:  0.8066 (146 of 181)  (reward=+489.00 WIN)
Current Accuracy:  0.8077 (147 of 182)  (reward=+489.00 WIN)
Current Accuracy:  0.8087 (148 of 183)  (reward=+489.00 WIN)
Current Accuracy:  0.8098 (149 of 184)  (reward=+489.00 WIN)
Current Accuracy:  0.8108 (150 of 185)  (reward=+489.00 WIN)
Current Accuracy:  0.8118 (151 of 186)  (reward=+489.00 WIN)
```

*Figure 4 Gripper only Result*

### Discussion

While the arm was able to meet the set requirements in both cases, the solutions discovered are not without their shortcomings:

- Both tasks were prone to knocking the cylinder over, this would not be a desirable property in a real-world robot (imagine the cylinder were made of glass!). This is likely an artifact of the object being evaluated – since we do not consider contact safety when evaluating the agent's performance, the agent was free to ignore it.
- In the first Task, the agents preferred solution involved resting its elbow on the ground. Depending on the environment, this may not be a desirable behavior – for example if the ground is extremely hot or composed of a material that is hazardous to the robot this could easily result in damage to the arm.
- The policy learned by the agent for Task 2 appears to be very sensitive to any randomness exhibited by the robot. This manifested in the agent seemingly "forgetting" the optimal policy during training attempts which would result in a series of consecutive failed "attempts" before it was able to regain itself again. This indicates that this agent may have trouble controlling a robot that experiences high uncertainty in its various servo controllers.

## Future Work

There are a few ways that the results might be improved on.

1. Now that we know the agent can be trained to align the gripper with the cylinder, we could extend to train the agent to approach the cylinder at a reasonable speed.  To do this we could track the grippers velocity and only provide a high reward if the gripper contacts the cylinder with a gentle motion.
2. A similar approach to the one suggested above could be used to keep the arm from smashing down on the cylinder in the first task.
3. The agent's representation of the robot as a set of buttons is not completely intuitive.  It maybe worthwhile refactoring the agent to represent the joint velocities in a continuous manner, controlling the robot with several "dials" instead of buttons.  Each dial would control the acceleration of a robot joint.  This would remove the need to learn that button 1 and button 2 have opposite effects on joint 1 as the idea would be encoded in the representation.
4. As mentioned previously, the agent we trained for the second task felt a little unstable.  We might be able to mitigate this by building a certain amount of uncertainty into the agent's actions.  In the current representation, if the agent chooses action 1, action 1 is preformed with probability one unless another random action is chosen (which is done with decreasing probability epsilon as the number of episodes increase) and then moves the joint by a set amount.  We might be able to train the agent to anticipate uncertainty by adding it into the actions. So, action 1 moves the joint by the same amount as before, but with some added gaussian noise to simulate uncertainty.  This could make the resulting solution more robust to random actions as they would get learned by the model over time.
5. The first task wasn't always taking the most efficient route to the solution, we could improve this by using the same approach used for task 2, by adding a bonus to the reward whenever the agent solves the problem quickly.