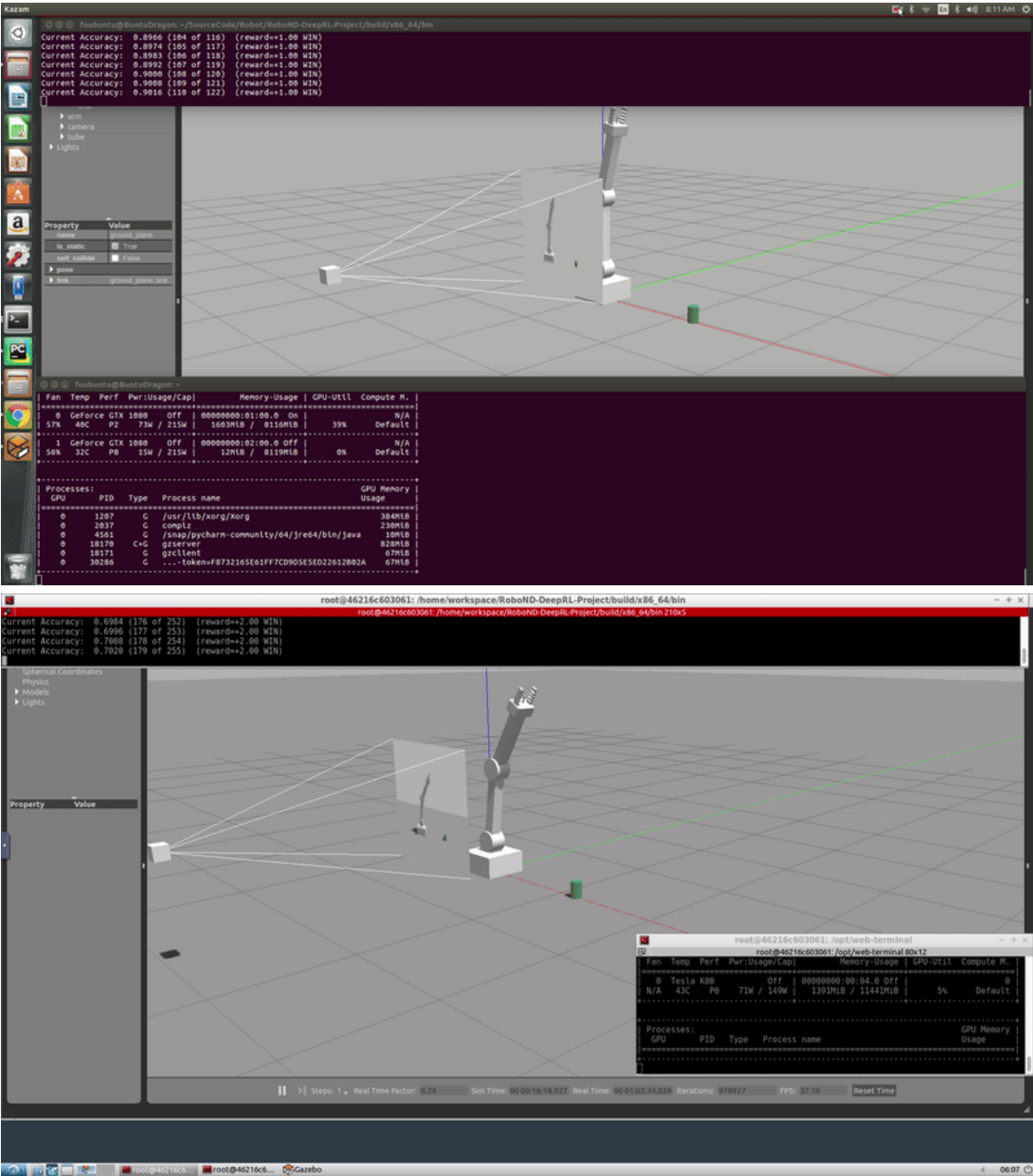


Deep RL Arm Manipulation



This project is based on the Nvidia open source project "jetson-reinforcement" developed by [Dustin Franklin](#). The goal of the project is to create a DQN agent and define reward functions to teach a robotic arm to carry out two primary objectives:

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

Building from Source (Nvidia Jetson TX2)

Run the following commands from terminal to build the project from source:

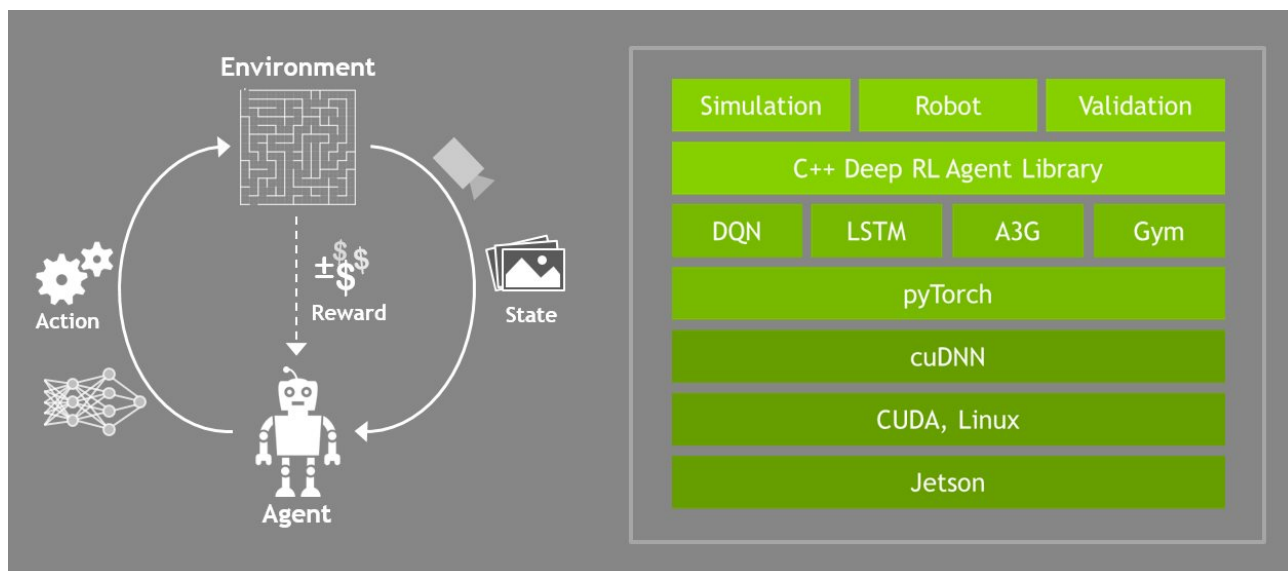
```
$ sudo apt-get install cmake
$ git clone https://github.com/fouliex/DeepRLArmManipulation.git
$ cd DeepRLArmManipulation
$ git submodule update --init
$ mkdir build
$ cd build
$ cmake ../
$ make
```

During the `cmake` step, Torch will be installed so it can take awhile. It will download packages and ask you for your `sudo` password during the install.

C++ API

To successfully leverage deep learning technology in robots, one need to move to a library format that integrate with robotics and simulators. In addition, robots require real-time responses to changes in their environments so computation performance matters. Therefore the API provides an interface to the Python code written with Pytorch, but the wrappers use Python's low level C to pass memory objects between the user's application and Torch without extra copies. By using a compiled language(C/C++) instead of an interpreted one, performance is improved and speeded up even more when GPU acceleration is leveraged.

API stack for Deep Reinforcement Learning



API Stack for Deep RL is from Nvidia [repo](#)

Arm Plugin

The Gazebo Arm Plugin

The robotic arm model found in the Gazebo world calls upon a gazebo plugin called `ArmPlugin`. This plugin is responsible for creating the Deep Q-Network(DQN) agent and training it to learn to touch the prop.

The gazebo plugin shared `libgazeboArmPlugin.so` a object file that attached to the robot model in the Gazebo world. That object file is responsible for integrating the simulation environment with the Reinforcement Learning(RL) agent. The plugin is defined in the `ArmPlugin.cpp` file located in the [gazebo](#) folder.

The Arm Plugin Base functions

The Arm Plugin Source Code

The `ArmPlugin.cpp` take advantage of the **C++ API**. This plugin creates specific functions for the class `ArmPlugin` defined in `ArmPlugin.h`.

ArmPlugin::Load()

This function is responsible for creating and initializing nodes that subscribe to two specific topics, one for the camera and one for the contact sensor for the object.

ArmPlugin::onCameraMsg()

This function is the callback function for the camera subscriber. It takes the message from the camera topic, extracts the image and saves it. This is then passed to the DQN.

ArmPlugin::onCollisionMsg()

This function is the callback function for the object's contact sensor. It is used to test whether the contact sensor, called `my_contact`, defined for the object in Gazebo world, observes a collision with another element/model or not.

ArmPlugin::createAgent()

This function serves to create and initialize the agent. Various parameters that are passed to the `Create()` function for the agent are defined at the top of the file such as:

```
#define INPUT_WIDTH    512
#define INPUT_HEIGHT   512
#define OPTIMIZER      "None"
#define LEARNING_RATE  0.0f
#define REPLAY_MEMORY  10000
#define BATCH_SIZE     8
#define USE_LSTM       false
#define LSTM_SIZE      32
```

ArmPlugin::updateAgent()

This function receives the action value from the DQN and decides to take that action. For every frame that the camera receives, the agent needs to take an appropriate action. Because the DQN agent is discrete, the network selects one output for every frame. This output which is the action value can then be mapped to a specific action thus controlling the arm joints.

There are two possible ways to control the arm joints:

- Velocity Control
- Position Control

For both of these types of control, one can increase or decrease either the joint velocity or the joint position, by a small delta value.

ArmPlugin::OnUpdate()

This function is utilized to issue rewards and train the DQN. It is called upon at every simulation iteration and can be used to update the robot joints, and issue both rewards based on the desired goal:

- End of Episode(EOE)
- Interim

At EOE, various parameters for the API and the plugin are reset, and the current accuracy of the agent performing the appropriate task is displayed on the terminal.

Arm Plugin Rewards

The reward information are defined in the `ArmPlugin::OnUpdate()` function. The arms joints are updated using position control and for each joint there are two action which is to increase or decrease the joint position.

Objective 1

Any part of the robot arm touch the object of Interest logic

- A reward(`REWARD_WIN * 10`) is giving If any part of the arm touch the object.
- A reward(`REWARD_WIN`) is giving if a positive weighted average is derived.
- A penalty(`REWARD_LOSS * 10`) is giving if any part of the robot touch the ground and the episode end.
- A penalty(`REWARD_LOST * distance to goal`) is provided if a negative weighted average is derived.
- Any collision ends the episode.

Objective 2

The gripper base of the robot arm touch the object of Interest logic

Reward Win and Reward loss

- A reward(`REWARD_WIN * 20`) is giving if the gripper base of the robot touch the object.
- A penalty(`REWARD_LOSS * 5`) is giving if any part of the robot touch the object.
- A penalty(`REWARD_LOSS * 10`) is the robot touch the ground
- A penalty(`REWARD_LOST`) is added for no movement if the absolute average goal is less than 0.001 for the gripper base.
- Any collision ends the episode.

Hyperparameters

Objective 1

Objective 1 was perform on a physical machine(Ubuntu 16.04) with a GTX1080 GPU. Below are the Hyperparameters:

- `INPUT_WIDTH` and `INPUT_HEIGHT` are set to 64.
 - Image dimensions are set to the same size as the input.
 - Training is perform on a GTX1080 therefore there's not need to restrict memory usage.
- `OPTIMIZER` is set to Adam. It performs better then RMSProp in this project.
- `LEARNING_RATE` is set to 0.1 for this objective.
- `REPLAY_MEMORY` is set to 10000 for this objective.
- `BATCH_SIZE` is set to 512.
 - 512 is use because there's enough memory on the GTX1080.
- `USE_LSTM` is set to true.
- `LSTM_SIZE` is set to 256.

Objective 2

Objective 2 was perform on a Udacity virtual machine with a Tesla k80 GPU.

- `INPUT_WIDTH` and `INPUT_HEIGHT` are set to 64.
- `OPTIMIZER` is set to Adam. it performs better then RMSProp.
- `LEARNING_RATE` is set to 0.01 for the second objective.
- `REPLAY_MEMORY` is set to 20000 for the second objective.
 - Due to the smaller surface area a higher replay memory is use to allow more discrete learning.
- `BATCH_SIZE` is set to 512.
 - 512 is use because there's enough memory on the Tesla k80.
- `USE_LSTM` is set to true.

- `LSTM_SIZE` is set to 256.

Result

Both objectives is achieve in the physical environment(Ubuntu 16.04, Objective 1) and the virtual environment(Udaticity VM, objective 2). As long as the robot reaches the goal on the first 50 tries the faster the training goes.

Below are screenshots and videos that the goal were achieved with a minimum of 100 runs:

Objective 1

Any part of the robot arm touch the object of interest with at least a 90% accuracy.

Physical environment(Ubuntu 16.04) screenshot


```

foubuntu@BuntuDragon: ~/SourceCode/Robot/RoboND-DeepRL
Current Accuracy: 0.8696 (060 of 069) (reward==+1.00 WIN)
Current Accuracy: 0.8714 (061 of 070) (reward==+1.00 WIN)
Current Accuracy: 0.8732 (062 of 071) (reward==+1.00 WIN)
Current Accuracy: 0.8611 (062 of 072) (reward=-1.00 LOSS)
Current Accuracy: 0.8630 (063 of 073) (reward==+1.00 WIN)
Current Accuracy: 0.8649 (064 of 074) (reward==+1.00 WIN)
Current Accuracy: 0.8667 (065 of 075) (reward==+1.00 WIN)
Current Accuracy: 0.8684 (066 of 076) (reward==+1.00 WIN)
Current Accuracy: 0.8701 (067 of 077) (reward==+1.00 WIN)
Current Accuracy: 0.8718 (068 of 078) (reward==+1.00 WIN)
Current Accuracy: 0.8734 (069 of 079) (reward==+1.00 WIN)
Current Accuracy: 0.8750 (070 of 080) (reward==+1.00 WIN)
Current Accuracy: 0.8765 (071 of 081) (reward==+1.00 WIN)
Current Accuracy: 0.8780 (072 of 082) (reward==+1.00 WIN)
Current Accuracy: 0.8795 (073 of 083) (reward==+1.00 WIN)
Current Accuracy: 0.8810 (074 of 084) (reward==+1.00 WIN)
Current Accuracy: 0.8824 (075 of 085) (reward==+1.00 WIN)
Current Accuracy: 0.8837 (076 of 086) (reward==+1.00 WIN)
Current Accuracy: 0.8851 (077 of 087) (reward==+1.00 WIN)
Current Accuracy: 0.8864 (078 of 088) (reward==+1.00 WIN)
Current Accuracy: 0.8764 (078 of 089) (reward=-1.00 LOSS)
Current Accuracy: 0.8778 (079 of 090) (reward==+1.00 WIN)
Current Accuracy: 0.8791 (080 of 091) (reward==+1.00 WIN)
Current Accuracy: 0.8696 (080 of 092) (reward=-1.00 LOSS)
Current Accuracy: 0.8710 (081 of 093) (reward==+1.00 WIN)
Current Accuracy: 0.8723 (082 of 094) (reward==+1.00 WIN)
Current Accuracy: 0.8737 (083 of 095) (reward==+1.00 WIN)
Current Accuracy: 0.8750 (084 of 096) (reward==+1.00 WIN)
Current Accuracy: 0.8763 (085 of 097) (reward==+1.00 WIN)
Current Accuracy: 0.8776 (086 of 098) (reward==+1.00 WIN)
Current Accuracy: 0.8788 (087 of 099) (reward==+1.00 WIN)
Current Accuracy: 0.8800 (088 of 100) (reward==+1.00 WIN)
Current Accuracy: 0.8812 (089 of 101) (reward==+1.00 WIN)
Current Accuracy: 0.8824 (090 of 102) (reward==+1.00 WIN)
Current Accuracy: 0.8835 (091 of 103) (reward==+1.00 WIN)
Current Accuracy: 0.8846 (092 of 104) (reward==+1.00 WIN)
Current Accuracy: 0.8857 (093 of 105) (reward==+1.00 WIN)
Current Accuracy: 0.8868 (094 of 106) (reward==+1.00 WIN)
Current Accuracy: 0.8879 (095 of 107) (reward==+1.00 WIN)
Current Accuracy: 0.8889 (096 of 108) (reward==+1.00 WIN)
Current Accuracy: 0.8899 (097 of 109) (reward==+1.00 WIN)
Current Accuracy: 0.8909 (098 of 110) (reward==+1.00 WIN)
Current Accuracy: 0.8919 (099 of 111) (reward==+1.00 WIN)
Current Accuracy: 0.8929 (100 of 112) (reward==+1.00 WIN)
Current Accuracy: 0.8938 (101 of 113) (reward==+1.00 WIN)
Current Accuracy: 0.8947 (102 of 114) (reward==+1.00 WIN)
Current Accuracy: 0.8957 (103 of 115) (reward==+1.00 WIN)
Current Accuracy: 0.8966 (104 of 116) (reward==+1.00 WIN)
Current Accuracy: 0.8974 (105 of 117) (reward==+1.00 WIN)
Current Accuracy: 0.8983 (106 of 118) (reward==+1.00 WIN)
Current Accuracy: 0.8992 (107 of 119) (reward==+1.00 WIN)
Current Accuracy: 0.9000 (108 of 120) (reward==+1.00 WIN)
Current Accuracy: 0.9008 (109 of 121) (reward==+1.00 WIN)
Current Accuracy: 0.9016 (110 of 122) (reward==+1.00 WIN)
Current Accuracy: 0.9024 (111 of 123) (reward==+1.00 WIN)
Current Accuracy: 0.9032 (112 of 124) (reward==+1.00 WIN)
Current Accuracy: 0.9040 (113 of 125) (reward==+1.00 WIN)
Current Accuracy: 0.9048 (114 of 126) (reward==+1.00 WIN)
Current Accuracy: 0.9055 (115 of 127) (reward==+1.00 WIN)
Current Accuracy: 0.9062 (116 of 128) (reward==+1.00 WIN)

```

Objective 2

The gripper base of the robot arm touch the object of Interest with at least a 80% accuracy.

Virtual environment(Udacity VM) screenshot

```

root@4029f18487bb: /home/workspa...ND-DeepRL-Project/build/x86_64/bin - + x
root@4029f18487bb: /home/workspace/RoboND-DeepRL-Project/build/x86_64/bin 80x33
Current Accuracy: 0.7821 (183 of 234) (reward==+2.00 WIN)
Current Accuracy: 0.7830 (184 of 235) (reward==+2.00 WIN)
Current Accuracy: 0.7839 (185 of 236) (reward==+2.00 WIN)
Current Accuracy: 0.7848 (186 of 237) (reward==+2.00 WIN)
Current Accuracy: 0.7857 (187 of 238) (reward==+2.00 WIN)
Current Accuracy: 0.7866 (188 of 239) (reward==+2.00 WIN)
Current Accuracy: 0.7875 (189 of 240) (reward==+2.00 WIN)
Current Accuracy: 0.7884 (190 of 241) (reward==+2.00 WIN)
Current Accuracy: 0.7893 (191 of 242) (reward==+2.00 WIN)
Current Accuracy: 0.7901 (192 of 243) (reward==+2.00 WIN)
Current Accuracy: 0.7910 (193 of 244) (reward==+2.00 WIN)
Current Accuracy: 0.7918 (194 of 245) (reward==+2.00 WIN)
Current Accuracy: 0.7927 (195 of 246) (reward==+2.00 WIN)
Current Accuracy: 0.7935 (196 of 247) (reward==+2.00 WIN)
Current Accuracy: 0.7944 (197 of 248) (reward==+2.00 WIN)
Current Accuracy: 0.7952 (198 of 249) (reward==+2.00 WIN)
Current Accuracy: 0.7960 (199 of 250) (reward==+2.00 WIN)
Current Accuracy: 0.7968 (200 of 251) (reward==+2.00 WIN)
Current Accuracy: 0.7976 (201 of 252) (reward==+2.00 WIN)
Current Accuracy: 0.7984 (202 of 253) (reward==+2.00 WIN)
Current Accuracy: 0.7992 (203 of 254) (reward==+2.00 WIN)
Current Accuracy: 0.8000 (204 of 255) (reward==+2.00 WIN)
Current Accuracy: 0.8008 (205 of 256) (reward==+2.00 WIN)
Current Accuracy: 0.8016 (206 of 257) (reward==+2.00 WIN)
Current Accuracy: 0.8023 (207 of 258) (reward==+2.00 WIN)
ArmPlugin - triggering EOE, episode has exceeded 100 frames
Current Accuracy: 0.7992 (207 of 259) (reward=-0.10 LOSS)
Current Accuracy: 0.8000 (208 of 260) (reward==+2.00 WIN)
Current Accuracy: 0.8008 (209 of 261) (reward==+2.00 WIN)
Current Accuracy: 0.8015 (210 of 262) (reward==+2.00 WIN)
Current Accuracy: 0.8023 (211 of 263) (reward==+2.00 WIN)
Current Accuracy: 0.8030 (212 of 264) (reward==+2.00 WIN) © George Fouché

```

Future Work

- Optimize the reward functions because after 50 runs if the gripper base doesn't touch the object of interest, there's very low chance for it to recuperate and learn how to touch the object.
- Play around by increasing the hyperparameters and see how the robot performs.
- Try making the robot to achieve the goals with the RMSProp optimizer instead of the Adam.

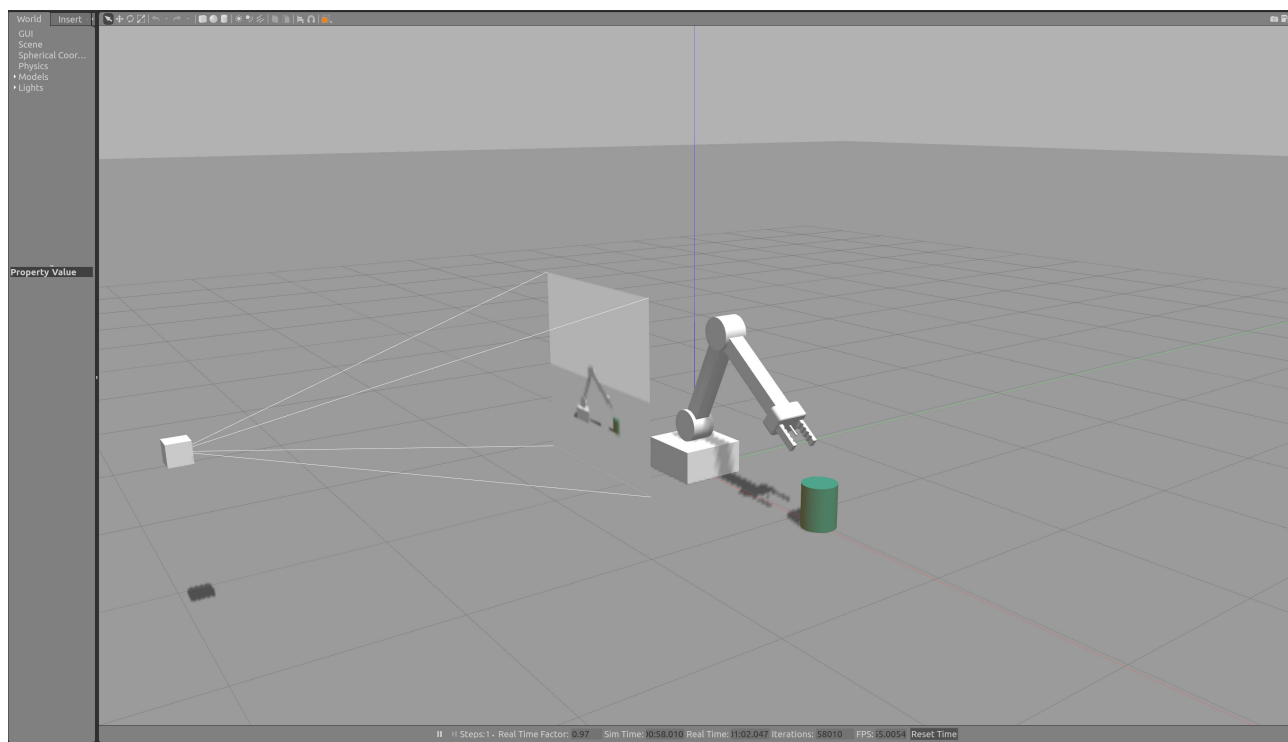
Project Environment

To get started with the project environment, run the following:

```

$ cd RoboND-DeepRL-Project/build/aarch64/bin
$ chmod u+x gazebo-arm.sh
$ ./gazebo-arm.sh

```



The plugins which hook the learning into the simulation are located in the `gazebo/` directory of the repo. The RL agent and the reward functions are to be defined in [ArmPlugin.cpp](#).