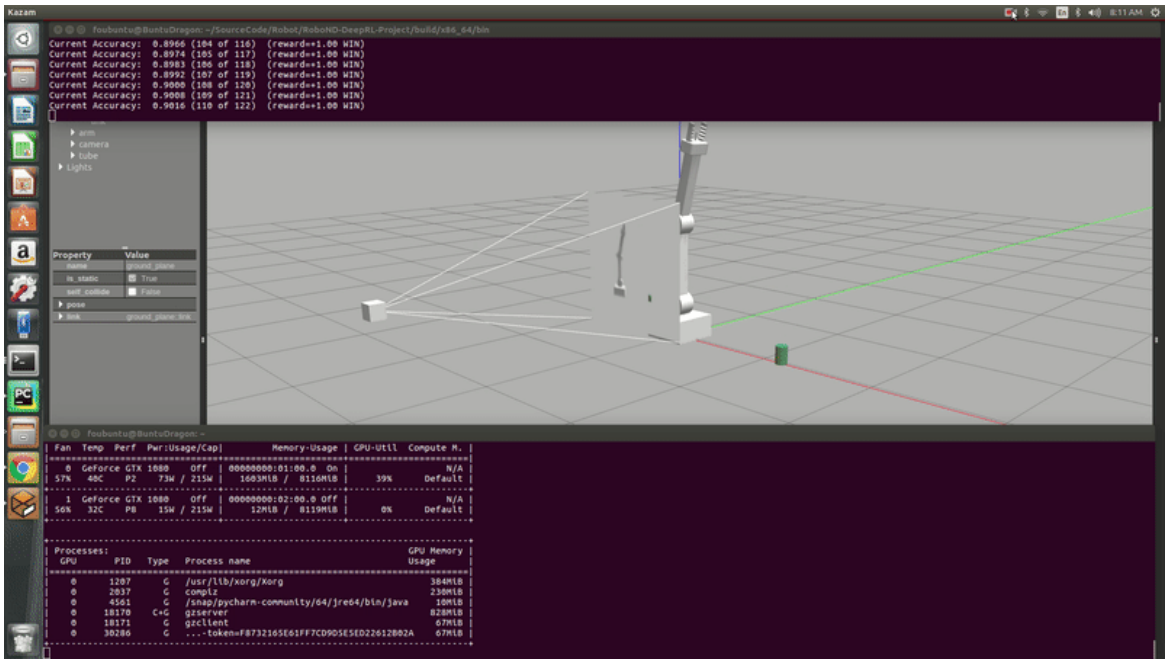
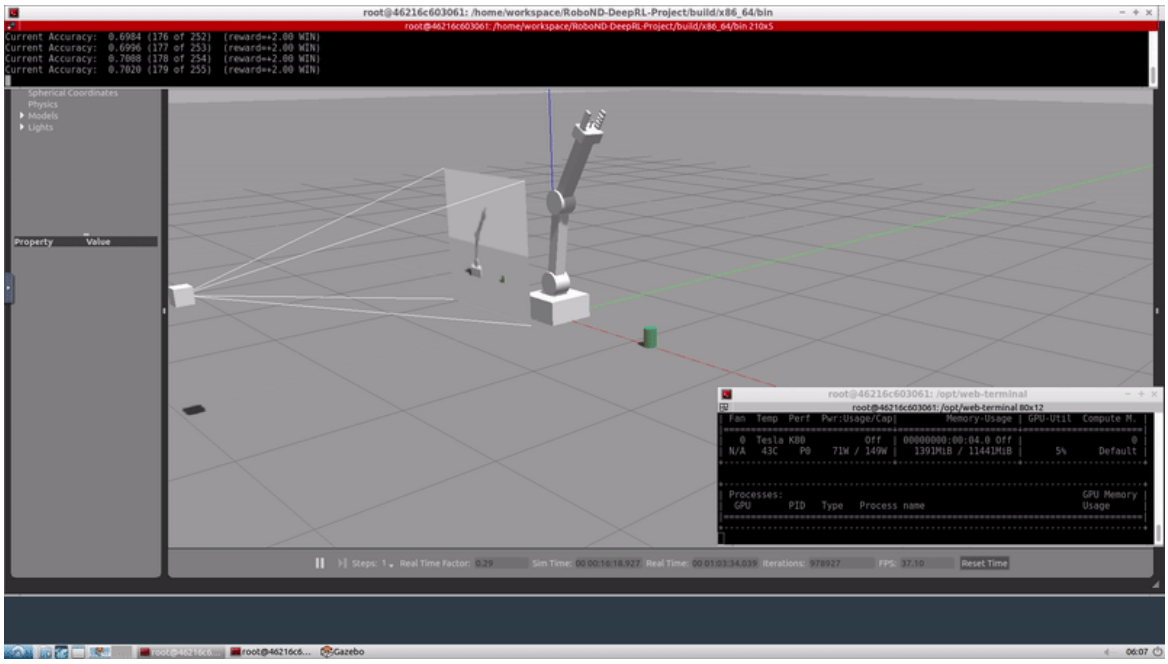
README.md

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:

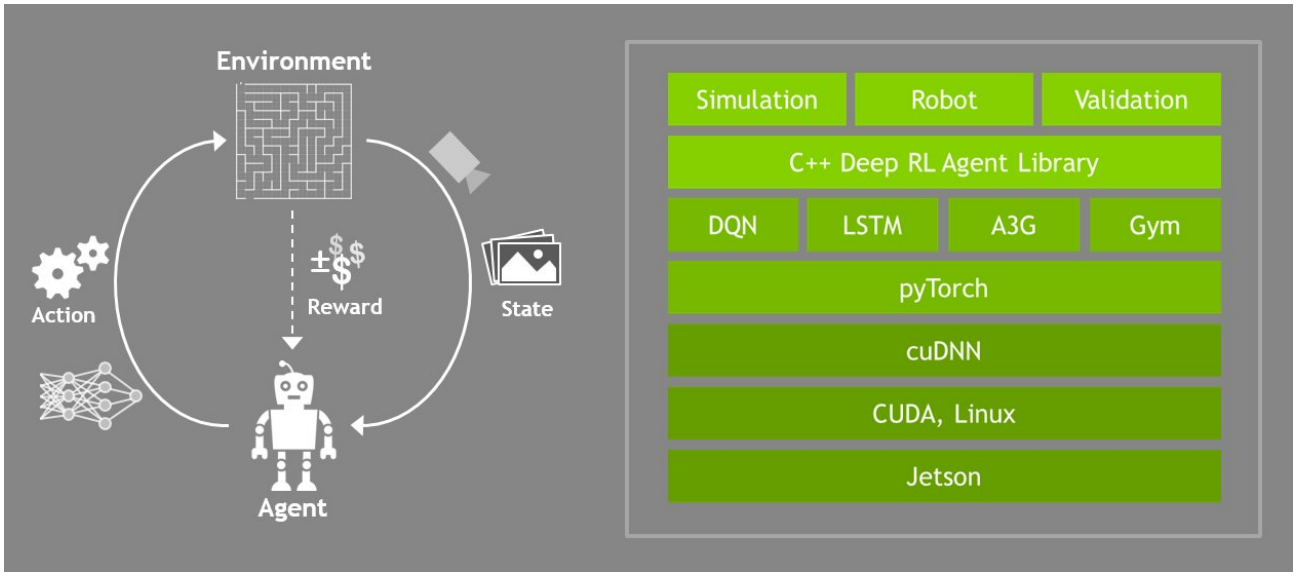
```
$ sudo apt-get install cmake
$ git clone https://github.com/fouliex/DeepRLArmManipulation.git
$ cd DeepRLArmManipulation
$ git submodule update --init
$ mkdir build
$ cd build
$ cmake ../
$ make
```

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 robotivs 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 complied 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 Deel 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 int 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 calllback 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(`REWORD_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

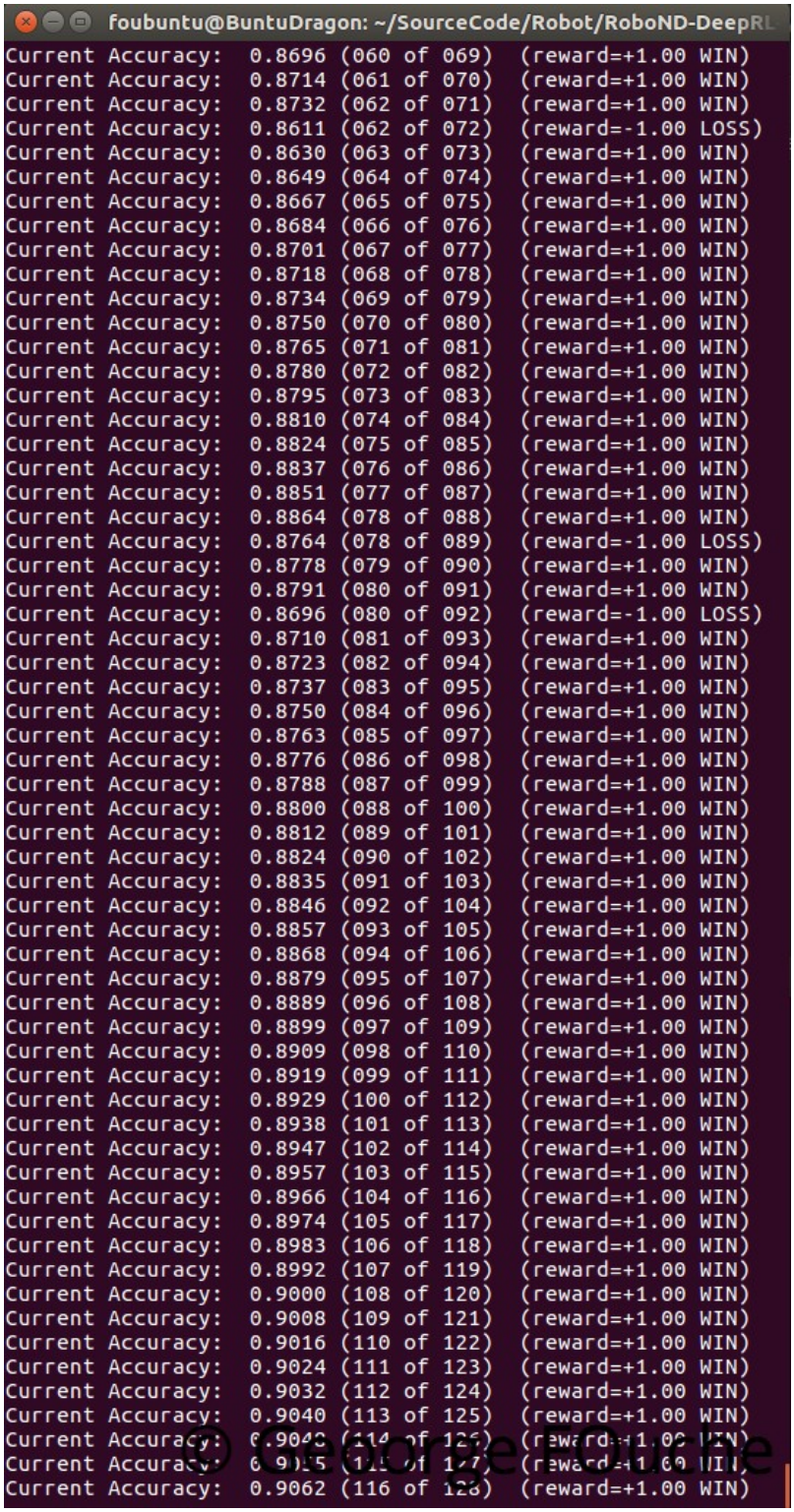
Both objectives is achieve in the physical environment(Ubuntu 16.04, Objective 1) and the virtual environment(Udaticity VM, objective 2). As long as the robot reaches the goal on the first 50 tries the faster the training goes.

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.

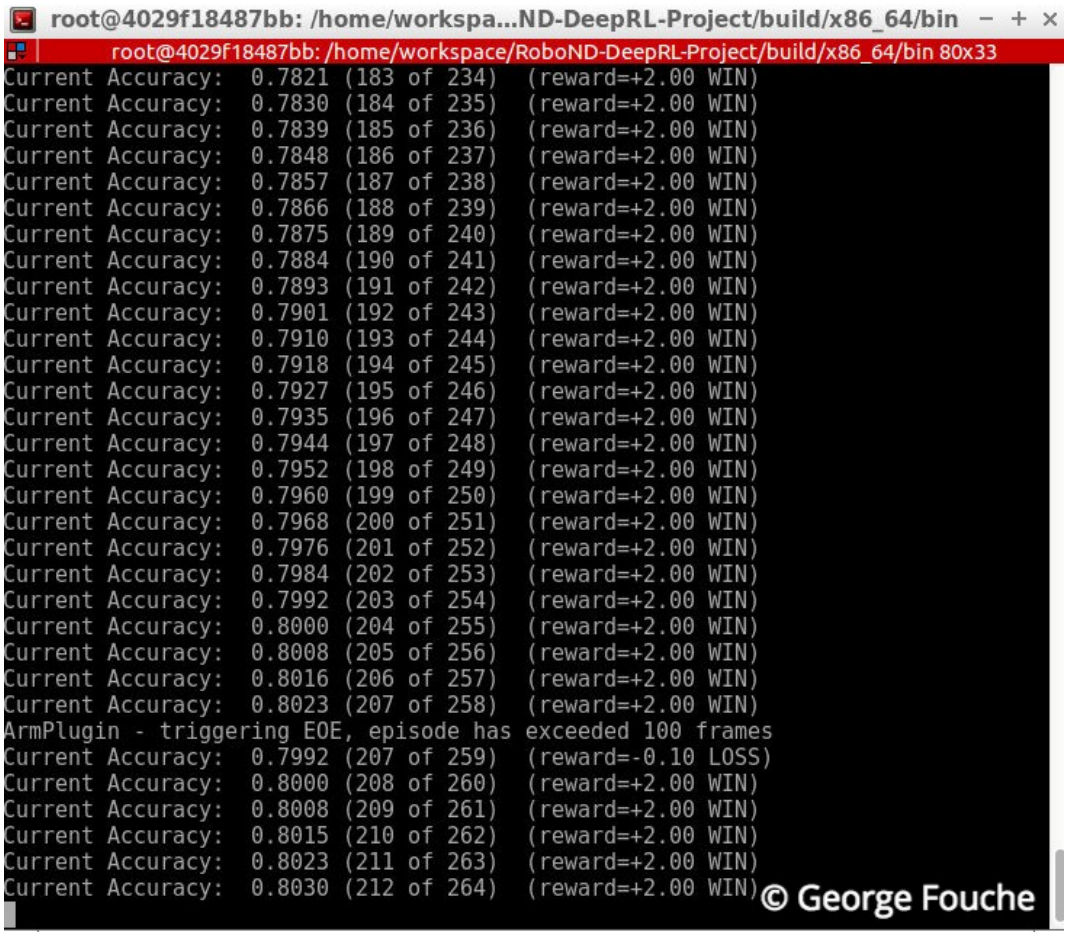
Physical environment(Ubuntu 16.04) screenshot



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



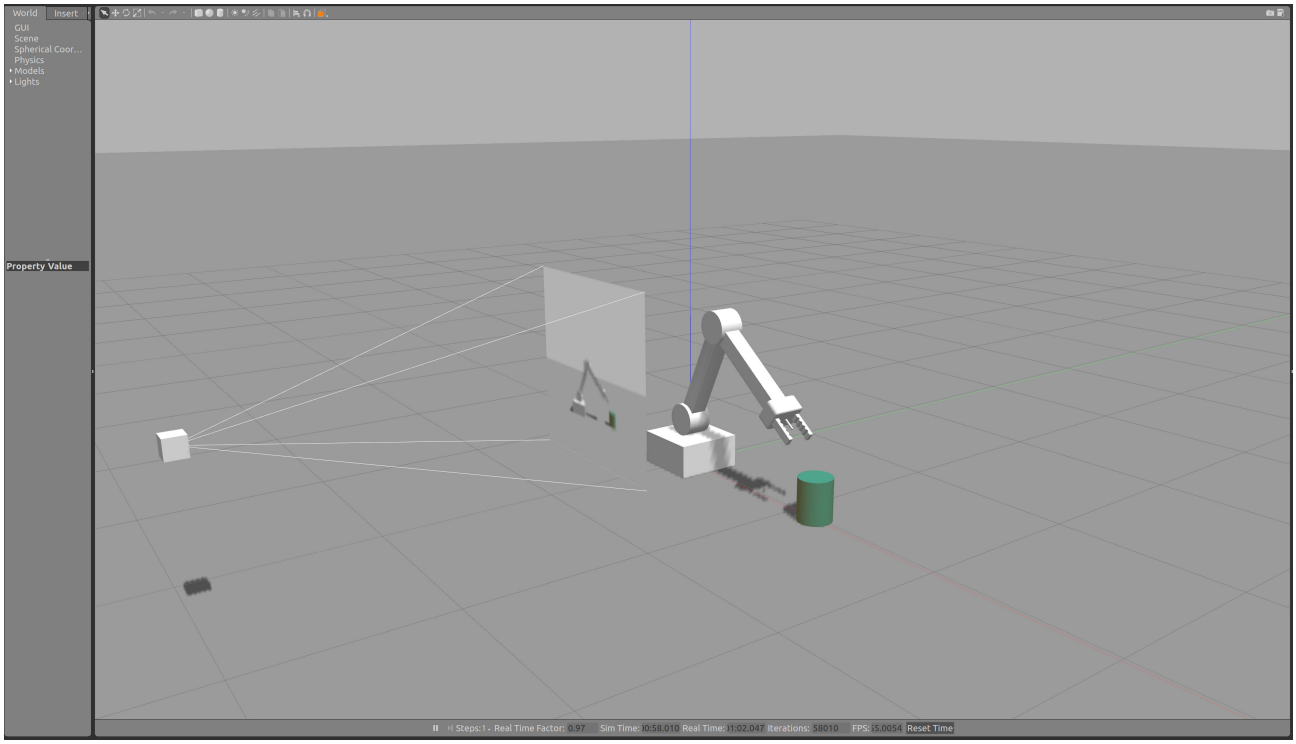
Future Work

- 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`.