ROS Bridge Suite and Unity Editor: Creating a Virtual Environment for Reinforcement Learning with Husky and Turtlebot3 Waffle Pie URDF Models

Progress report
for
Anubhav Fellowship

Submitted to
IIT Mandi iHub and HCI Foundation

Duration
Dec 2022-Feb 2023

Submitted by
Student Name: Ravi H M
Course: BTech, Year: 2nd and Branch name: Data science and engineering
College Name: IIT Mandi

Mentor by
Faculty/Mentor Name: Dr. Varun Dutt
Designation: Mr. Shashank Kapoor

Certificate of Completion

This is to certify that I Ravi H M has successfully completed a 3-month (Dec 2022 - Feb 2023) internship under the supervision of Dr. Varun Dutt in the field of Applied cognitive algorithm to robots to perform search and retrieve task.

During the Anubhav fellowship, Ravi H M has demonstrated excellent skills and knowledge in the aforementioned areas. He has worked on various projects related to Applied cognitive algorithm to robots to perform search and retrieve task, showcasing his ability to code efficiently and effectively. He has also shown remarkable dedication and commitment to the tasks assigned to him.

Signed: Recommended via Email

Faculty/Mentor Name: Dr. Varun Dutt
Designation: Mr. Shashank Kapoor

Acknowledgement

It has been great honour and privilege to undergo training at IIT Mandi iHub and HCI Foundation.

I am highly indebted to Dr. Varun Dutt for their guidance and constant supervision as well as for providing necessary information regarding the project and also for their support in completing the project. His constant guidance and willingness to share his vast knowledge made me understand this project and its manifestations in great depths ad helped me to complete the assigned tasks on time.

I would like to express my gratitude towards Mr. Shashank for his kind cooperation and encouragement which helped me in completion of this project.

-Ravi H M

Abstract/Summary

The project aims to create a virtual environment using ROSbridge and Unity Editor for training and testing reinforcement learning algorithms for training mobile robots. Specifically, the virtual environment will use Husky and Turtlebot3 Waffle Pie URDF models to create a simulated environment for training and testing.

The project will involve several steps, including:

1. Integrating ROSBridge Suite and Unity Editor: This involves setting up ROSBridge Suite to communicate with Unity Editor, and importing the Husky and Turtlebot3 Waffle Pie URDF models into the Unity Editor.
2. Creating a virtual environment: This involves designing and developing a virtual environment in Unity Editor that mimics the real-world environment in which the robot will operate.
3. Developing reinforcement learning algorithms: This involves developing and testing reinforcement learning algorithms that can be used to train the Husky and Turtlebot3 Waffle Pie robots in the virtual environment. These algorithms will enable the robots to learn how to navigate the environment, avoid obstacles, and complete other tasks.
4. Evaluating performance: Once the reinforcement learning algorithms have been developed, the performance of the robots will be evaluated in the virtual environment. The performance metrics will include factors such as the time taken to complete tasks, the number of obstacles encountered, and the accuracy of navigation.
5. Validating the virtual environment: Finally, the virtual environment will be validated by comparing the performance of the robots in the virtual environment with their performance in the real world. This will help to ensure that the virtual environment is an accurate representation of the real world, and that the trained robots are able to transfer their learning to the real world.

Overall, the project aims to create a flexible and scalable virtual environment for training and testing reinforcement learning algorithms for mobile robots, using ROSBridge Suite and Unity Editor. The project will contribute to the development of more reliable and effective mobile robots, by allowing researchers and developers to test and refine their algorithms in a safe and controlled virtual environment, before deploying them in the real world.

Table of Contents

Introduction

Unity .NET 4.x framework

Unity .NET 4.x framework is built on top of the .NET Framework and provides a managed environment for C# code to execute. This means that when you write C# code in Unity, it is compiled to Intermediate Language (IL) and executed by the .NET runtime.

C# is the primary language used in Unity for scripting gameplay mechanics, user interfaces, and other features. Unity's implementation of C# is based on the Microsoft .NET Framework, which means that C# code in Unity is written using the same syntax and language features as standard .NET applications.

ROS Bridge

ROSbridge is a websocket server that provides a way to interact with a ROS (Robot Operating System) system using a variety of programming languages, including JavaScript, Python, and C++. The server is built using the roslibpy library, which is written in Python and runs on a ROS system.

ROSbridge provides a number of services and topics that allow clients to interact with the ROS system, such as subscribing to topics, publishing to topics, and calling ROS services. Clients can connect to the ROSbridge server using a WebSocket connection, and can use JSON messages to send and receive data to and from the server.

When a client connects to the ROSbridge server, it sends a JSON message containing information about the client and the types of messages it can handle. The server uses this information to create a WebSocket connection with the client, and to subscribe to any topics or services that the client is interested in.

Once the connection is established, the client can send messages to the server to interact with the ROS system. For example, a client could send a message to subscribe to a topic, and then receive messages from the server whenever a new message is published to that topic. Similarly, a client could send a message to call a ROS service, and receive a response from the server containing the result of the service call.

ROS Sharp (ROS#)

ROS# is an open-source library that provides C# developers with a way to work with ROS (Robot Operating System) in their projects. It allows developers to communicate with ROS nodes and exchange data with them.

ROS# provides a C# implementation of the ROS client library, which allows C# code to communicate with ROS nodes using the same ROS communication protocol that is used by the ROS system. This implementation is based on the .NET Standard 2.0, so it can be used in various .NET environments, such as .NET Core, .NET 4.x , Xamarin, Unity, and more.

The ROS# library is structured into several components, each providing specific functionalities:

1. RosBridgeClient: This component provides a C# implementation of the rosbridge protocol, allowing the communication between ROS and non-ROS systems through websockets.
2. RosSharp: This component contains the core library functionalities, such as the implementation of ROS messages, nodes, publishers, subscribers, services, and actions.
3. RosBridgeClientTest: This component provides a set of tests to ensure that the rosbridge protocol implementation is working properly.
4. RosBridgeClientUnity: This component provides a set of Unity-specific classes and methods to work with ROS# in Unity.

Methodology and Outcome

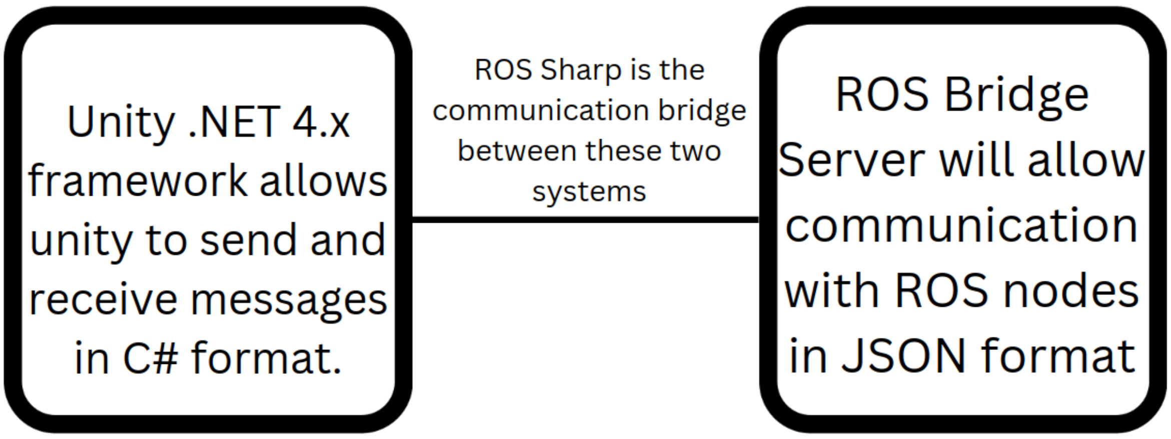

Unity .NET 4.x framework allows unity to send and receive messages in C# format.
ROS Sharp is the communication bridge between these two systems
ROS Bridge Server will allow communication with ROS nodes in JSON format

Fig 1: Connection between ROS bridge and Unity

This Project was Achieved in 6 tasks:

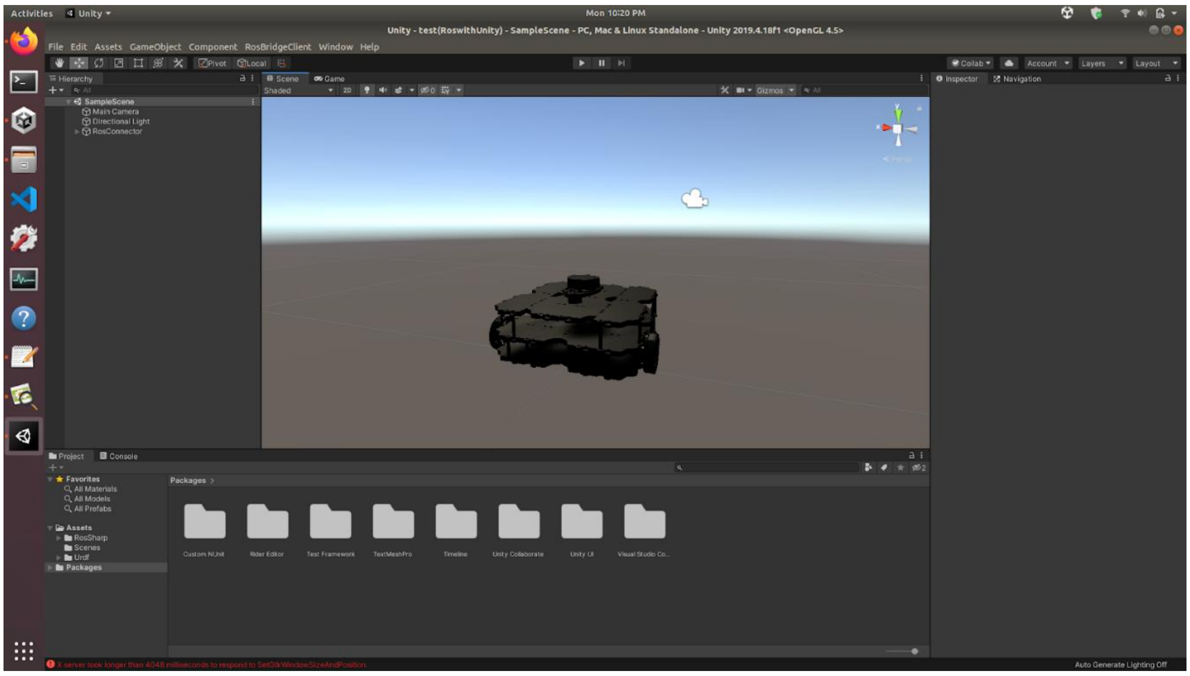
Task 1: Installation and learning.

Gaining the necessary skills and knowledge to work with Ubuntu Terminal, ROS Melodic, and Unity Editor, which are the key tools required for the project. By choosing Unity Editor, which provides a more immersive and realistic environment than some other software options, such as Gazebo, we have tried to build a high-quality virtual environment. Additionally, selecting version 2019.4.18f (LTS) for compatibility with ROS Sharp packages will ensure that the project runs smoothly and is able to take full advantage of the features offered by both Unity and ROS.

Task 2: Importing Turtlebot3 Waffle Pie URDF model

By using the Turtlebot3 GitHub repository URDF packages, importing the Turtlebot3 Waffle Pie URDF model into Unity Editor was successful. Which allows us to work with a realistic representation of the Turtlebot3 Waffle Pie robot, which will be an important component of the virtual environment used for reinforcement learning.

The communication between the URDF model and ROS Master is typically established through the use of the ROS communication protocol, which allows nodes to exchange messages and services. The ROS Bridge suite package provides a way to bridge the communication between ROS and non-ROS systems, such as Unity, by translating ROS messages to a format that can be understood by the non-ROS system (ROS messages to JSON).



X server took longer than 4048 milliseconds to respond to SetGtkWindowSizeAndPosition

Fig 2: Imported turtlebot3 waffle pie model

Task 3: Communication with imported URDF model

Establish communication between the imported Turtlebot3 Waffle Pie URDF model and ROS Master using the ROS Bridge suite package over WebSocket protocol. Additionally, we have enabled teleoperation of the Turtlebot3 Waffle Pie model using the ‘cmd_vel’ topic.

Task 4: Importing Husky URDF model

By using the Husky GitHub repository URDF packages, importing the Husky URDF model into the Unity Editor environment was successful.

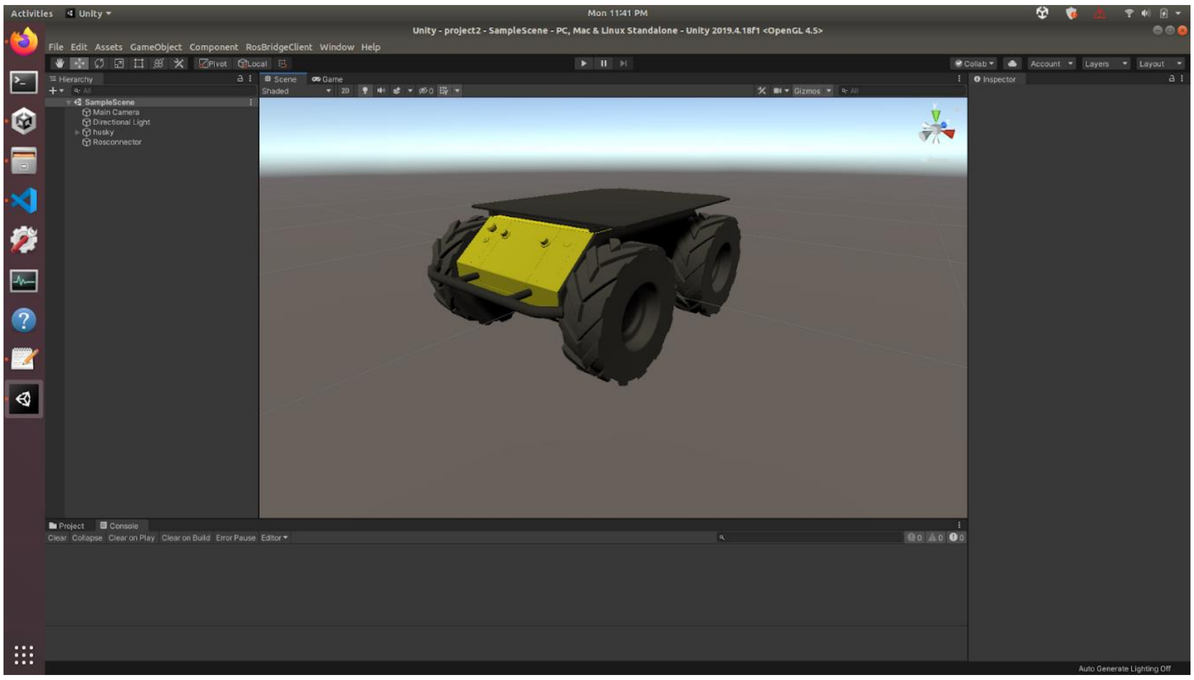

Activities
Unity
Mon 11:41 PM
Unity - project2 - SampleScene - PC, Mac & Linux Standalone - Unity 2019.4.18f1 <OpenGL 4.5>
File Edit Assets GameObject Component RosBridgeClient Window Help
Hierarchy
SampleScene
Main Camera
Directional Light
husky
Rosconnector
Scene
Game
Inspector
Project
Console
Clear Collapse Clear on Play Clear on Build Error Pause Editor
Auto Generate Lighting Off

Fig 3: Imported Husky URDF

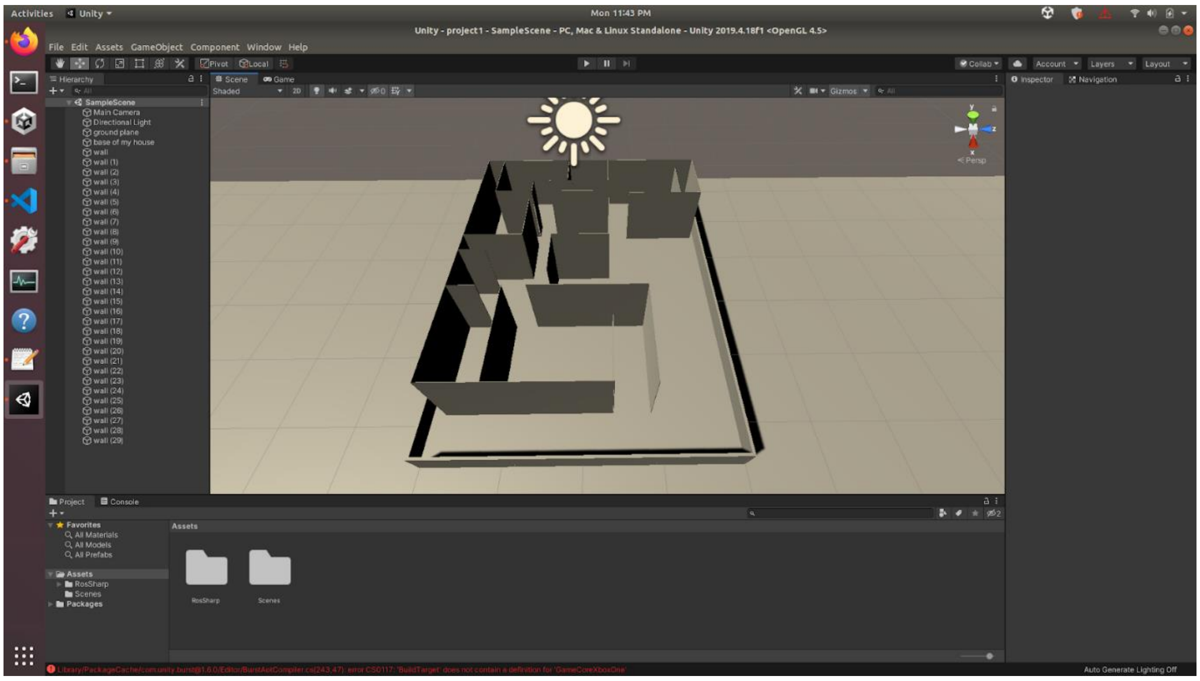
Task 5: Setting up ml-agents environment

ML-Agents is an open-source Unity plugin that allows researchers and developers to train and test AI agents in a variety of environments. It provides an interface for Unity to communicate with popular deep learning frameworks such as TensorFlow and PyTorch.

To set up the ML-Agents environment, I downloaded and installed the package from the Unity Asset Store and configured the project settings to use it. I then added the necessary components to the Unity Editor environment and installed the ml-agents package in Ubuntu Desktop.

The ML-Agents environment will be used to train the AI agents to navigate the virtual environment and perform tasks using reinforcement learning techniques.

Task 6: Building a realistic environment in the Unity Editor for training and testing the AI agent.



```
Library/PackageCache/com.unity.burst@1.6.0/Editor/BurstAotCompiler.cs(243,47): error CS0117: 'BuildTarget' does not contain a definition for 'GameCoreXboxOne'
```

Fig 4: Building built

Coding and Implementation

Installation Steps

Make a partition in Windows to install Ubuntu 18.04 (Bionic).
Reference link1

Open ubuntu.

Install ros1 melodic.
Reference link2

Download and install unity hub
Reference link3

Download and install Unity Version 2019.4.18f (LTS)*.
Reference link4
(You can also do it directly from Unity Hub)

ENVIRONMENT SETUP for Turtlebot3 Waffle Pie BOT

Reference link5

Follow the steps in ubuntu terminal.

```
$ mkdir -p ~/unity_ws/src
$ cd unity_ws
$ catkin_make
$ source devel/setup.bash

$ cd ~/unity_ws/src
$ git clone -b melodic-devel https://github.com/ROBOTIS-GIT/turtlebot3
$ git clone -b melodic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs
$ git clone -b melodic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations
$ git clone -b ros1 https://github.com/RobotWebTools/rosbridge_suite

$ catkin_make
$ source devel/setup.bash
```

clone the ros# repository to any of the directories (not in catkin workspaces):

```
$ git clone https://github.com/siemens/ros-sharp.git
```

Copy the folder named ROS to the src folder of your catkin workspace (here it's unity_ws).

```
$ cd ~/unity_ws
$ catkin_make
```

Now open publish_description_turtlebot2.launch file (we need to edit it)
Path: unity_ws/src/ROS/file_server/launch/publish_description_turtlebot2.launch

after opening the .launch file
delete that text and copy paste this:

```
<launch>

   <include file="$(find file_server)/launch/ros_sharp_communication.launch">
            <arg name="port" value="9090" />
   </include>

   <!--<arg name="base" default="$(env TURTLEBOT_BASE)" />
   <arg name="stacks" default="$(env TURTLEBOT_STACKS)" />-->
   <arg name="MODEL" default="waffle_pi" />
   <arg name="urdf_file" default="$(find xacro)/xacro.py '$(find
turtlebot3_description)/urdf/turtlebot3_$(arg MODEL).urdf.xacro'" />

   <param name="robot/name" value="turtlebot3_waffle_pi" />
   <param name="robot_description" command="$(arg urdf_file)" />

</launch>
```

Save and Close.

```
$ cd ~/unity_ws
$ source devel/setup.bash

$ roslaunch file_server publish_description_turtlebot2.launch
```

Now open Unity Hub, then Unity Editor 19.4.18f (LTS)

Follow these steps in Unity Editor

Create a New Project

Window > Asset Store search for Ros# and download Ros#

Edit > Project Settings > Player > Other Settings > Configuration > API compatibility Level
set it to .Net 4.x Equivalent

RosBridgeClient > import URDF from Ros...

> Settings

Change the Address to the IP address of the machine running ROS (If you are using WSL on windows you can use localhost instead). Make sure to change the protocol from Web Socket NET to Web Socket Sharp in the settings, as shown in the image.

> Read Robot Description

import turtlebot3_wiffle_pi window > Yes

Interrupt and Close all Terminals

Connection between ROS Node and Unity (.NET)

Reference link6

Create a new empty game object (RosConnector) in Unity New Project (that was already created earlier).

In the Inspector Window, click "Add Component".

Search for "Ros connector (script)" and ADD it.

Make sure the protocol is "Web Socket Sharp".

Change RosBridge Server Url to “localhost”.

In the Inspector Window, click "Add Component".

Search and ADD "TwistSubscriber (Script)".

In the Topic, type "/cmd_vel"

Drag "turtlebot3_wiffle_pi" from Hierarchy Window and drop in Subscribed Transform.

Click Play Button (placed middle of the screen)

Open 3 Terminals

Terminal 1: $ roslaunch $(rospack find rosbridge_server)/launch/rosbridge_websocket.launch

Terminal 2: $ rosrun rqt_graph rqt_graph

Terminal 3: $ export TURTLEBOT3_MODEL=waffle_pi
$ roslaunch $(rospack find turtlebot3_teleop)/launch/turtlebot3_teleop_key.launch

Check for correctness of rqt_graph.

ENVIRONMENT SETUP for Husky bot

Reference link7, link8 and link9

In Terminals

```
$ sudo apt-get install ros-melodic-husky-simulator
$ cd ~/unity_ws/src
$ git clone -b melodic-devel https://github.com/husky/husky.git
```

Copy the file description.launch (file path: unity_ws/src/husky/husky_description/launch) and paste it into path: unity_ws/src/ROS/file_server/launch.

Rename the file to husky_description_unity.launch

Now open husky_description_unity.launch file and paste this:

```
<?xml version="1.0"?>

<launch>

    <include file="$(find file_server)/launch/ros_sharp_communication.launch">
            <arg name="port" value="9090" />
    </include>

  <arg name="robot_namespace" default="/"/>

  <param name="robot_description" command="$(find xacro)/xacro '$(find
husky_description)/urdf/husky.urdf.xacro'
            robot_namespace:=$(arg robot_namespace)" />

</launch>
```

Save and Close.

```
$ cd ~/unity_ws
$ source devel/setup.bash

$ roslaunch file_server husky_description_unity.launch
```

Now open Unity Hub, then Unity Editor 19.4.18f (LTS).

Follow these steps in Unity Editor

Create a New Project

Window > Asset Store, search for Ros# and download Ros#

Edit > Project Settings > Player > Other Settings > Configuration > API compatibility Level
set it to .Net 4.x Equivalent

RosBridgeClient > import URDF from Ros...

> Settings

Change the Address to the IP address of the machine running ROS (If you are using WSL on windows you can use localhost instead). Make sure to change the protocol from Web Socket NET to Web Socket Sharp in the settings, as shown in the image.

> Read Robot Description

import Husky window > Yes

Kill and Close all Terminals

ML-Agents environment Setup

Reference link10 and link11

In Terminal

```
$ python3 -v
```

```
$ sudo apt install python3.7-venv
```

Open a new Terminal in the created Unity Project folder

```
$ python3.7 -m venv project-env
```

```
$ source project-env/bin/activate
```

```
$ pip install requests
```

Next Step, follow this link12 (Install the com.unity.ml-agents Unity package only).

```
$ python -m pip install mlagents==0.26.0
```

Other references:

- APT Proxy Setup Link13.
- PIP Proxy Setup Link14.

Future scope

Once we have the environment set up, we can start working on the reinforcement learning algorithms to train our bot.

There are many reinforcement learning algorithms to choose from, such as Q-learning, SARSA, and deep reinforcement learning algorithms like DQN and A3C. we'll need to choose an algorithm that's appropriate for our future problem and train our bot using simulation data.

Once our bot is trained, we can test it in the real world by deploying it to a physical robot and observing its behavior.