Experiment Design and Implementation for Physical Human-Robot Interaction Tasks

Xiangyu Xie

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Mechanical Engineering

University of Washington

2020

Committee:

Joshua R. Smith, Chair

Sawyer B. Fuller

Xu Chen

Program Authorized to Offer Degree:
Mechanical Engineering

University of Washington

Abstract

Experiment Design and Implementation for Physical Human-Robot Interaction Tasks

Xiangyu Xie

Chair of the Supervisory Committee:
Joshua R. Smith

Human-robot interaction has been an active research field for several years with the focus on understanding, designing, and evaluating robotic systems that are useful in helping people to perform certain task. Competitive interaction is one of these tasks that is studied less but has great potentials in certain scenarios like entertainment, physical therapy and athletic training areas. The main challenge in this field is to design algorithms that can allow robots to play against human.

Recent research in multi-agent reinforcement learning have achieved satisfactory results in generating complex and dexterous robotic behaviors even under non-stationary environment. This progress opens up the research opportunity on applying multi-agent reinforcement learning in human-robot interaction tasks. To facilitate current and future research in such direction, this thesis aims to design and implement a robotic system to evaluate the viability of various multi-agent reinforcement learning algorithms under certain human-robot interaction scenarios.

The robotic system consists of the following subsystems: 1. A real-time perception system that captures the dynamical state of certain human body frames(hands, feet, torso, and etc.). 2. Computational server to deploy deep learning models for decision making purpose. 3. A robot with high degree-of-freedom arms and a low latency communication method to connect all subsystems to ensure robot's real-time reactions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my committee members, Professor Joshua Smith, Professor Sawyer Fuller and Professor Xu Chen for taking time to serve on the committee.

I would like to thank a PhD student, Boling Yang, for his suggestions and help in this project. It is a great pleasure to work with you and learn to maintain a broader view of researches before I am moving to the next stage in my academic career.

I would also like to thank all the friends I have met during the two-year studies at University of Washington. Your accompanies and support are crucial to me through the hard time.

At last I want to thank my family for believing in me and supporting me both financially and emotionally.

Chapter 1

INTRODUCTION

1.1 Motivation

Robots are artificial agents that are able to perceive the environment the make actions based on the observation without human interference. There are many types of robots nowadays, classified mainly by their different functionalities and the field they are used in. Industrial robots can substitute human workers to perform repetitive tasks in factories; Drones and self-driving cars can locate themselves in the environment, plan a route and move accordingly; Medical robots can help doctors in delicate surgeries. With the advancement in the field of robotics research, robots are becoming more and more intelligent and can be expected to become an essential part in our daily life soon.

Among all the active robotics research field, human-robot interaction (HRI) has become a hot topic in recent years. As robots now have the ability to perform complex tasks like collaborating with or compete against humans, the interactions between humans and robots have to be thoroughly studied. It also opens opportunities in many other applications. Rehabilitation robots and training robots are examples of assisting people to achieve certain goals by interactions.

Reinforcement learning (RL), on the other hand, has recently achieved remarkable results in creating artificial intelligence to solve complex problems that are intractable for other learning methods. The RL agent can beat the best human player in board games and learn to play complicated real-time strategy video games. Naturally, it has also been used in teaching robots to learn a strategy through objectives on themselves. The progress in this area has attracted us to apply RL in human-robot interaction tasks.

In order to test the feasibility and performance of RL algorithms on HRI tasks, a platform

needs to be created for simulation and testing. While actual robots are required to interact with human players, tasks setup in the real world can be severely limited and inefficient. The desired platform should be able to support different kinds of interaction tasks while allowing communications between robots and humans. Virtual reality (VR) can fulfill the requirements by creating an immersive environment where the task can be carried out. The accurate and real-time tracking system can enable robots to perceive the states of human.

1.2 Related Work

Human-robot interaction is a multidisciplinary field with contributions from robotics, artificial intelligence, social sciences and etc. In [5] the authors thoroughly discussed the key aspects in HRI. They explained why HRI has emerged as new field rather than a collection of studies in other field. The applications and the challenges that HRI is currently facing were also presented.

Most HRI research focused on robots assisting or collaborating with humans. [4] presented research on socially assistive robots. Specific task called handover was used as to study robots' performance under collaborations with human [11]. In [2], the authors introduced the project using PR2 to assist people with disabilities. Platforms for specific purpose have been designed to further examine HRI relationships like in [3] the authors mainly focused on robots learning how to detect opportunities to assist humans. The experience of interacting with a robot is very different compared to people's interaction experience with other technologies and artifacts. [13] studied the difference between a robotic assistant and a human assistant in manufacturing industries and [14] showed that social and emotional components had to be taken into consideration for evaluating HRI.

Applying learning methods for robots in HRI is also an active research field. In [12] the authors used machine learning to allow robots to learn objects in a new environment. Human demonstration or guidance are proven to be important in early exploration stage to make the learning process more efficient and reinforcement learning can be helpful in teaching robots from human demonstration [1].

In order to evaluate the performance of HRI work, standards needs to be followed. [10] presented common metrics in HRI from several aspect: perception, manipulation, communication.

1.3 Problem Description

The goal of this thesis is to design a system that can serve as the platform for testing HRI tasks with RL algorithms. In order to make this system versatile, we decide to use VR to create an virtual task environment while synchronizing between the real and the virtual world to create realistic experience with physical contacts available. The structure of this system consists of three parts:

- A robot with flexible arm and quick response ability to interact with human in real time.
- A perception system that enables communication between robots and humans; Both humans and robots should be able to perceive each other's states in real time.
- A server that can deploy RL models to receive information from the robot and control it accordingly.

1.4 Thesis Outline

There are total of 5 chapters in this thesis. Chapter 1 introduces the research topic and also provides some related background knowledge. It explains the reason to design a platform for HRI tasks and choices of VR and RL. Chapter 2 describes the specific VR game that we create and former research of VR in HRI and RL field. In Chapter 3, all the component in the system are shown, together with the techniques used for communication and calibration. Chapter 4 tested some RL algorithms in the presented task with results and discussions. Chapter 5 summarize the whole research and points what can be done in the future.

Chapter 2

VIRTUAL REALITY

Virtual reality (VR) is a computer generated environment that can be similar to or completely different from the real world. It can visually separates users from the physical world and allows them to interact in an immersive environment. The three most important characteristics of VR are: immersion, presence and interaction. While both immersion and presence describe the perception of being physically present in a non-physical world, the former refers to the objective descriptions of aspects of the system such as the eld of view (FOV), stereoscopy, display resolution and input devices, and the latter refers to the subjective sensation of the virtual experience. Interaction, on the other hand, can be more straightforwardly defined as the communication between the human agency and the virtual environment.

From a broader scope of view, displaying a 3D virtual world on a regular graphics monitor can be regarded as the most primitive VR system. Mouse are usually used to navigate through this kind of virtual environment and stereo glass or stereo projection can be used to turn the environment stereo. Most first-person video gaming falls into this category. However, this type of VR system is characterized as *Non-Immersive* as it does not provide users with the ability to know what is happening around them. Most VR system referred today are *Fully-Immersive*, which provides 360 full view of the virtual environment. An earlier type of such system is called the Cave Automatic Virtual Environment (CAVE) that uses three to six projectors in a room-sized cube to create immersive experience for users. The high cost and poor portability, however, makes it impossible for commercial use. Head-mounted displays (HMD) gained more popularity since 2010 and has now became the dominant commercially available VR device. It is typically a display device that worn on

the head of the user like a helmet, with a small high resolution OLED or LCD monitor in front of each eye. Separate images are presented for each eye to create depth perceptions for rendering a 3D virtual world, while inertial measurement units (IMU) provide the positional and rotational head tracking to enable the user to look around, thus give the fully-immersive experience.

Although most consumer VR devices nowadays are designed for entertainment purposes, like video gaming and film watching, the rapid development of all kinds of technologies in the VR field has stimulated numerous application areas. For example, VR has been used in 3D art and design area for artists to easily create and present their work, VR simulation can achieve educational purpose by simulating virtual experiments, virtual tourism uses VR to bring realistic experience to the user at any places, VR can also be used for medical purpose to treat anxiety disorders or help rehabilitation processes. In many engineering fields, VR can be used to simulate real workspace environment without concerning about safety issues while minimizing the cost.

Besides the typical type of VR technology, many extensions have been developed recently, together making a broader scope of VR field. Augmented reality (AR) blends the real and virtual environment to present computer-generated objects on the real world, and Mixed reality (MR) combines VR and AR to make real-virtual interaction possible.

In conclusion, VR, as a relative new technology with about half century history, has created lots of new possibilities to different applications, and is continuing to gain more and more importance in both everyday life and research areas.

2.1 VR in HRI and RL

Virtual reality is gaining popularity in the field of Human-robot interaction for the abilities to create immersive environment, allow interaction and provide embodiment. Compared to other simulation techniques, VR is superior in creating realistic experience. For example, [7] showed that VR can increase performance on collaborative tasks compared to 2D simulations. In [9], the authors designed a VR training system to enable real-time cooperation between

industrial robotic manipulators and humans on simple manufacturing tasks.

In the field of reinforcement learning, VR can also be used to collect user data for imitation learning or behavior studying [15]. The tracking system in VR can be further exploited in teleportation tasks.

2.2 VR Fencing Game

In order to fulfill the whole structure of the VR system design, a particular VR game is presented for demonstration purpose. Note that the system is designed as a common platform for human-robot competitive tasks, rather than confined to this game.

The VR game we present is called fencing, which consists of two agents, one attacker and one defender. One of the agent is played by human and the other is played by robot. Two agents are standing on the ground facing each other, with a ball in the middle. Each agent has a cylinder-shape bat in one hand, and only the arm with bat are allowed to move through the game. The goal for the attacker is to hit the ball with his/its bat and the defender tries to protect the ball from being hit by blocking the attacker's bat with its own bat.

This game is designed as a continuous game, which means there is no time or steps limit. During every initialization of the game, the starting position for each agent will be changed slightly based on the equilibrium position, in order to give some randomness to each game.

Figure 2.1 shows the fencing game in the simulated environment. Here we use two robots for simplification. The robot model in this environment is the PR2 robot, and is exactly the robot we choose to compete with humans in the real world. More details about this robot will be discussed in the System Design chapter.

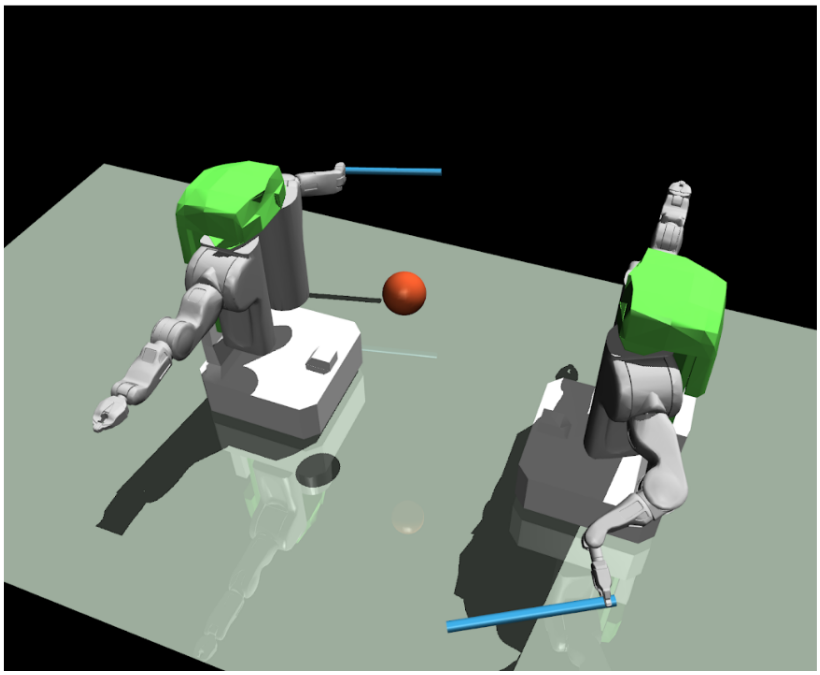

Figure 2.1: Fencing game example

Chapter 3

SYSTEM DESIGN

The whole structure of our system can be illustrated by Figure 3.1. There are five main components in our system: Robot Operating System (ROS), Unity game engine, real PR2 robot, VR device and a Reinforcement learning model. ROS acts as the core in our system to directly control the robot and allows communications between other parts. The PR2 robot and VR device are the hardware in our system, the former serves as an agent to play against a human player in the fencing game, and the latter is the input device in the VR world. Since our game can be played either in real or virtual world, the VR model of PR2 robot is required as well as the real robot. VR device are essential to play the VR game, but in real game it can also be used as a tracking device for human movement. Unity game engine is chosen to implement the VR environment because it perfectly supports various VR devices and provides abundant interface. Many successful VR games were developed in Unity and can be run on different platforms with different VR equipment. The last part of the system is the RL model, which is the key algorithm to control the PR2 robot during the game. More details about RL model will be discussed in the next Chapter.

This chapter will explain each component of the system in detail from four aspects: hardware, software, communication and calibration.

3.1 Hardware

In the hardware part, we will introduce the robot and the VR device we use for our task.

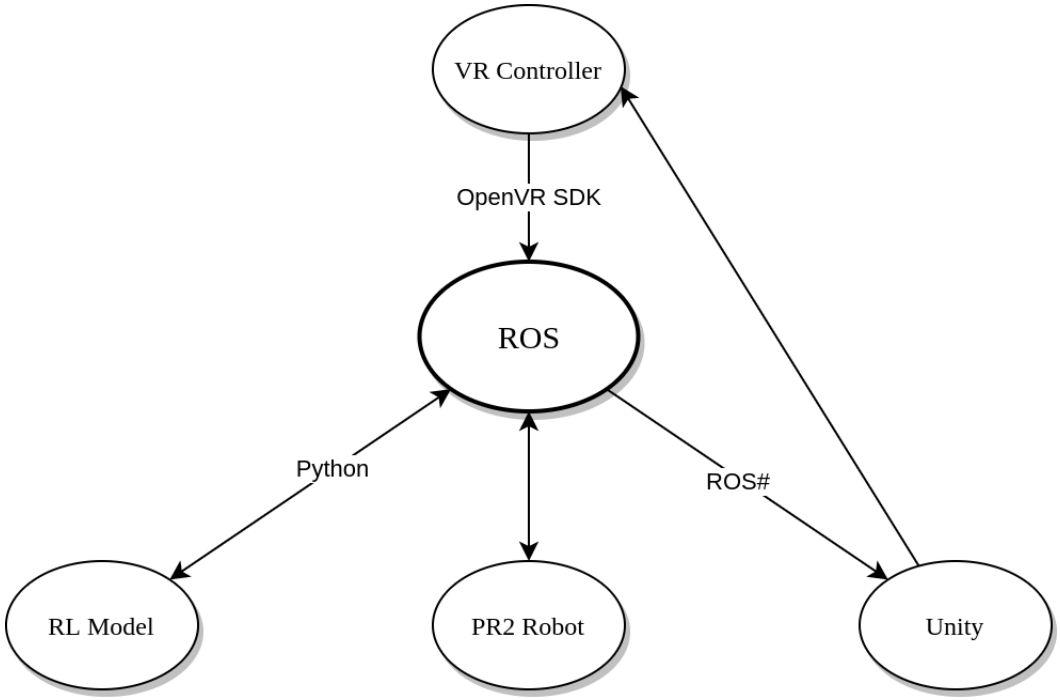

Figure 3.1: System structure

3.1.1 PR2 robot

PR2 stands for Personal Robot 2, a robot that has approximately the same size of a person and similar human body structures. It consists of a mobile base station with four steered and driven casters, a torso linked to the base that can move up and down, a head with stereo cameras, two arms and two grippers. Computers and battery system are included in the base, making PR2 an independent robot system. The torso can move up and down relative to the base and the head is linked to the torso which can be rotated and tilted up and down just like the human head. Left and right arms are attached to the torso with symmetry and each arm contains seven joints to allow dexterous movement. The gripper consists of a central palm linked to the wrist with only one actuated degree of freedom. Compared to human fingers, it can only do simple tasks like picking up objects.

Unlike industrial robots that are typically designed for manufacturing tasks, which re-

quires large carrying capacity, the PR2 robot is created as a robotics research and development platform. For safety concerns, the arms are designed to be back-drivable so when a PR2's arm encounter an object, the motor will drive the arm to a stop. Also motors on PR2 are relatively small. One reason is because the payload it requires is small, only 1.8 kilograms for each arm; the other is that the PR2 has a complicated counterbalance spring mechanisms on each arm. This passive equipment compensates for the effect of mass of the arm in different configurations and allows the arm to "float" without extra force.

There are mainly three reasons that we use the PR2 robot for our task:

- The RR2 robot has basically the same size and body structure as humans.
- Its arm has 7 joints, each with 1 degree of freedom, which allows flexible arm movements.
- It is designed as a hardware and software platform for robotics research, so it comes with comprehensive support and interfaces that can be easily integrated with other softwares.

Specific to our fencing game, the PR2 robot is not required to move its base station relative to the ground or its torso up and down. The PR2 will be fixed in a calibrated position before the game starts and only one of its arm is allowed to move during the game. Basically we are treating PR2 as an arm robot in our task environment and only the specs of arms need to be studied. Figure 3.2 shows the real PR2 robot and its 7 joints in one arm. Each joint has one DOF and can be controlled separately. From the name of the joints we can tell that its arm has similar structures as humans with shoulder, elbow, forearm and wrist. The way that each joint can move is also straightforward to see like lift, roll and flex, combined to allow flexible movements of the PR2's arm.

For each joint of the arm, there are some specification to its position, velocity and torque limits. These limits are essential to our system as 1. We need to represent the exact specs in

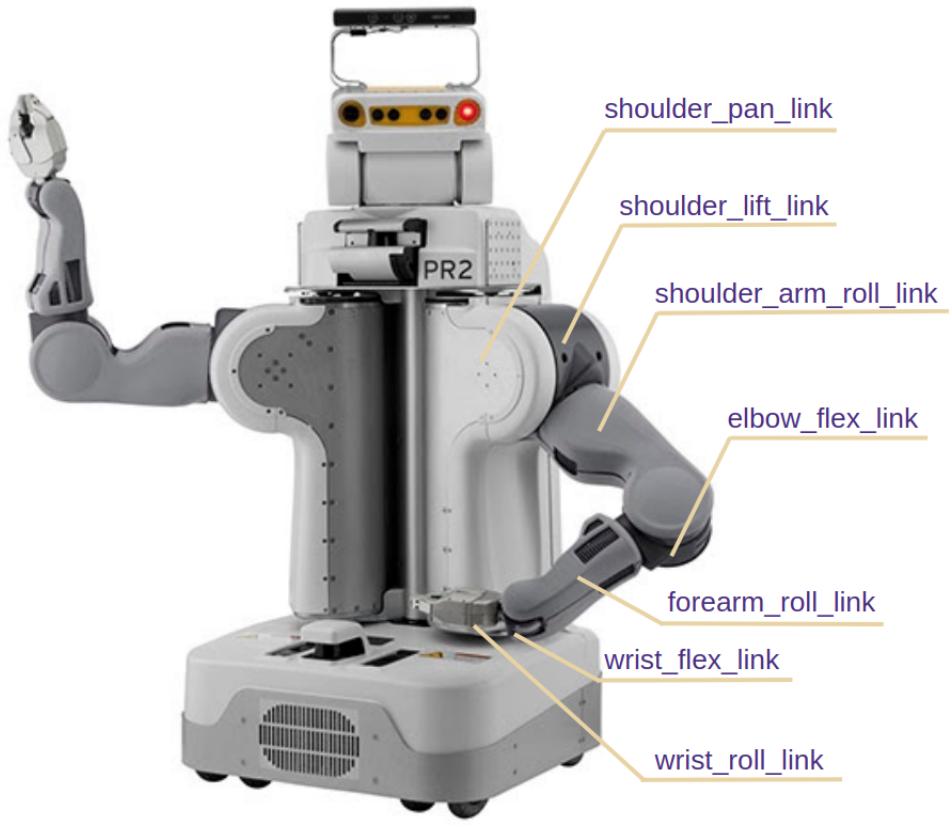

Figure 3.2: PR2 robot with arm joints

reinforcement learning environment to learn a strategy for the game; 2. For feasibility and safety concerns in real environment. Table 3.1 lists corresponding joint limits for reference.

3.1.2 HTC Vive Pro

The HTC Vive is a high end virtual reality headset with room scale tracking ability. It was developed by a mobile company HTC and Valve, creator of the largest PC gaming platform Steam. It was first released in 2016 with the rising of VR games. In 2018, HTC upgraded the model to be known as HTC Vive Pro. Figure 3.3 shows the basic components of HTC Vive Pro.

	Velocity (rad/s or m/s)	**Torque (Nm or N)**	**Limit (degree)**
shoulder_pan_joint	2.10	30	[-40, 130]
shoulder_lift_joint	2.10	30	[-30, 80]
upper_arm_roll_joint	3.27	30	[-44, 224]
elbow_flex_joint	3.30	30	[0, 133]
forearm_roll_joint	3.60	30	\
wrist_flex_joint	3.10	10	[0, 130]
wrist_roll_joint	3.60	10	\

Table 3.1: PR2's arm joint limits

The HTC Vive Pro comprises of a headset, two controllers and two base stations. With the help of two AMOLED screens, the headset supports high-resolution displays of 1440 × 1600 for each eye with super-rich colors and contrast. The refresh rate is 90 Hz with a field of view (FOV) of about 110 degrees. Attachable headphones and a noise-canceling microphone are included in the headset to provide full gaming experience. The controller is the main device for interaction and allows users to move freely in the VR environment. Besides the position and orientation of the controller will be tracked and shown in the simulated world, it also provide multiple types of physical interactions like the trackpad, trigger and button. Two base stations, also known as lighthouses, are the tracking device that emits pulsed infrared red (IR) lasers to track both the headset and the controller. Table 3.2 gives the detail specification of the HTC Vive Pro Headset and Table 3.3 gives the detail of the Controller and the Tracked area requirements.

While one base station is able to track both the headset and the controller, two base stations can avoid obstruction issues and information is synchronized through multiple devices to determine the user's position in real time. The tracked area that it requires can be made as small as standing or seated space or large as 5m × 5m. With Steam VR Tracking 2.0

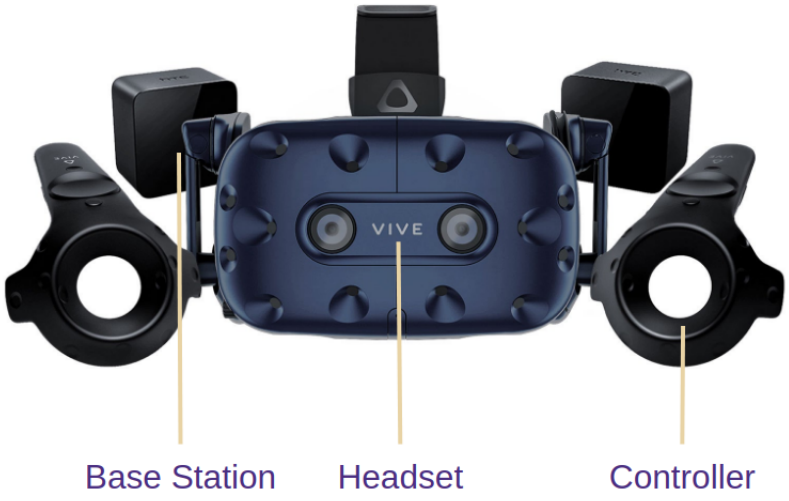

Figure 3.3: HTC Vive Pro components

support, it can be extended to four base stations with support up to 10m × 10m. Figure 3.4 shows the typical room setup and how the synchronization works.

Besides the basic components of the HTC Vive Pro Full Kit, Vive also provides lots of accessories that can greatly extend its functionalities. While the initial controllers are wireless, the headset has to be connected through a HDMI cable to ensure the large data transferring from the computer to the headset screen. Vive provides wireless adapters that can make the headset wireless too, which benefits games with more convenience. As mentioned above, the base station can be extended to four with large room space support. For our Human-Robot interaction research, this is an important feature as more games can be designed and tested without space constraints. Moreover, there are different types of controllers, focusing on certain sports games, like a racket with tracker on it. The stand alone tracker is also available and can be used even to track users' body movements.

Note that HTC Vive Pro supports SteamVR platform that can be run both on Windows and Linux. This is another reason to choose it for our task as ROS, a software to control the robot, is also fully supported on Linux.

Screen	Dual AMOLED 3.5” diagonal
Resolution	1440 × 1600 pixels per eye
Refresh rate	90 Hz
Field of view	110 degrees
Audio	Hi-Res certificate headset Hi-Res certificate headphones (removable) High impedance headphones support
Input	Integrated microphones
Connections	Bluetooth, USB-C port for peripherals
Sensors	SteamVR Tracking, G-sensor, gyroscope, proximity, Eye Comfort Setting (IPD)
Ergonomics	Eye relief with lens distance adjustment Adjustable Eye Comfort Setting (IPD) Adjustable headphones Adjustable headstrap

Table 3.2: Specification of HTC Vive Pro Headset

3.2 Software

The software part will talk about the softwares we use to control the robots and design the VR environment.

3.2.1 ROS

The Robot Operating System (ROS) is not an operating system as its name suggests but rather a framework for writing robot software. It is a large, open-source project with collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

For robotics research, designing a general-purpose robot software is hard as there are too

Sensors	SteamVR Tracking 2.0
Input	Multifunction trackpad Grip buttons dual-stage trigger System button Menu button
Use per charge	Approx. 6 hours
Connections	Micro-USB charging port
Tracked Area Requirements (Standing/Seated)	No min. space requirements
Tracked Area Requirements (Room-scale)	Up to 10m x 10m using four SteamVR Base Station 2.0. The included two base stations support up to 5m x 5m.

Table 3.3: Specification of HTC Vive Pro Controller and Tracked Area Requirements

many tasks and environment specific problems. ROS provides services similar to an operating system such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. With its core software, users can create their own packages on top of that to solve certain robotics problems, which forms a greater software ecosystem. ROS was developed for multiple robots from the start and has become the most popular robotics platform in research area.

Since ROS depends on large collections of open-source software dependencies, Unix-like systems like the Ubuntu Linux is fully supported while other systems like Windows and macOS are still listed as experimental. For this reason, we choose the Ubuntu 16.04 LTS in our research.

ROS operates in a graph architecture where processes (nodes) are distributed to receive, post and multiplex sensor data, control, state, planning, actuator, and other messages. The processed can be grouped into packages and can be easily implemented in modern program-

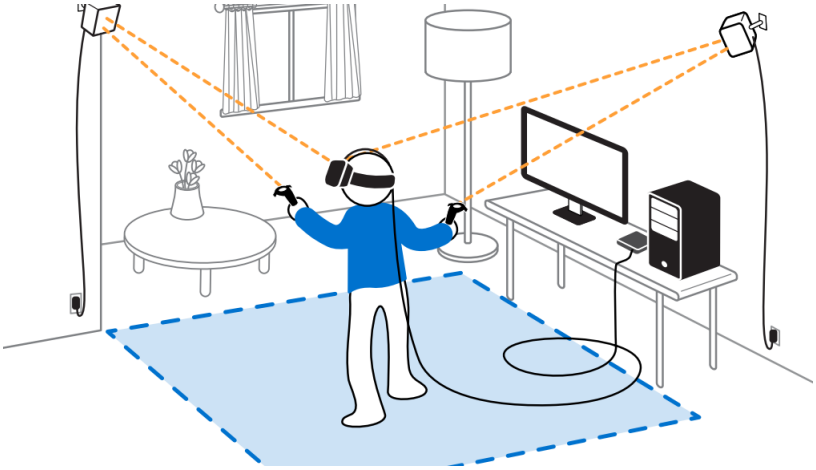

Figure 3.4: HTC Vive Pro Room Setup

ming languages like C++ and Python. ROS can be used to simulate the robotics environment or directly control the real robot with low latency. Specifically for our research, ROS serves as the core to communicate with other hardware and software, like controlling the motors in the real PR2 robot, tracking the position of VR controllers and passing the information to robots, synchronizing the movement between the real PR2 and the model in the VR environment.

The PR2 software system is written entirely in ROS, so that all PR2 capabilities are available via ROS interfaces. For example, states and actuators information can be easily accessed with ROS; ROS can control the PR2 to do teleportation, navigation and motion planning; Sensors like the camera can be read and visualized through ROS. For visualization purpose, a ROS package called rviz can be used to provide detail information about the robot running in real time. Figure 3.5 displays the interface of rviz visualizing the PR2 robot. Models of the PR2 robot was imported in rviz and the position and orientation of each joints were determined by relative transformations maintained by another ROS package called tf.

Besides the ability to directly interact with the real PR2 robot, ROS also provides a simulation package for PR2. However, the simulation in our environment is done inside

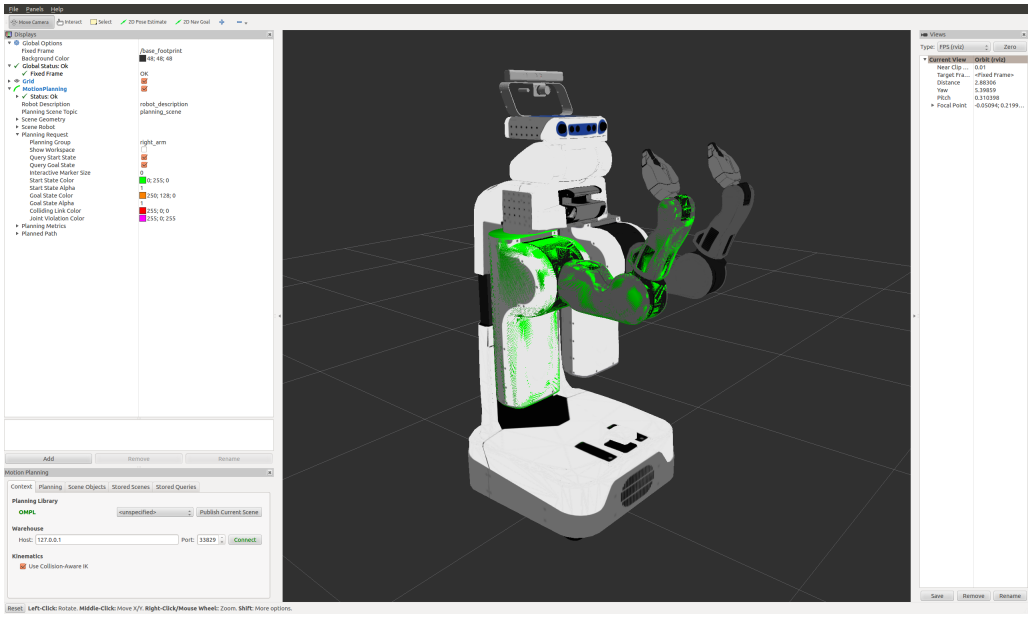

Figure 3.5: rviz interface

Unity so that this package is not required.

3.2.2 Unity

Unity is the world's most popular game engine with cross-platform support. It was created to design 2D and 3D games with physics engines, and has supported VR, AR games as well as simulations so far. The engine has also attracted artists and engineers to use it in film, architecture and construction areas. To develop Unity games, its own Unity editor has to be used with support on multiple platforms including Windows, macOS and Linux. The editor provides user-friendly drag and drop functionality, as well as scripting API in C#.

Figure 3.6 shows the basic interface of the Unity editor. In the middle part there are two scenes separated vertically; On the left is the scene view where we can visually navigate and edit the scene; On the right is the game view to display the scene through our created scene cameras. The editor can also run the game that we created for rendering and testing purposes. On the left is the hierarchy window that shows every objects we have created in

the scene and their relationships, e.g. one object is attached to another one. The inspector window shown on the right allows us to examine and edit all the properties of an selected object; We can also add/delete properties here using its provided interface or write our own scripts. On the bottom is the project window that displays the library assets that are available to use, like materials, objects, scripts, etc. Unity has its own asset store where we can import the official or user-created assets.

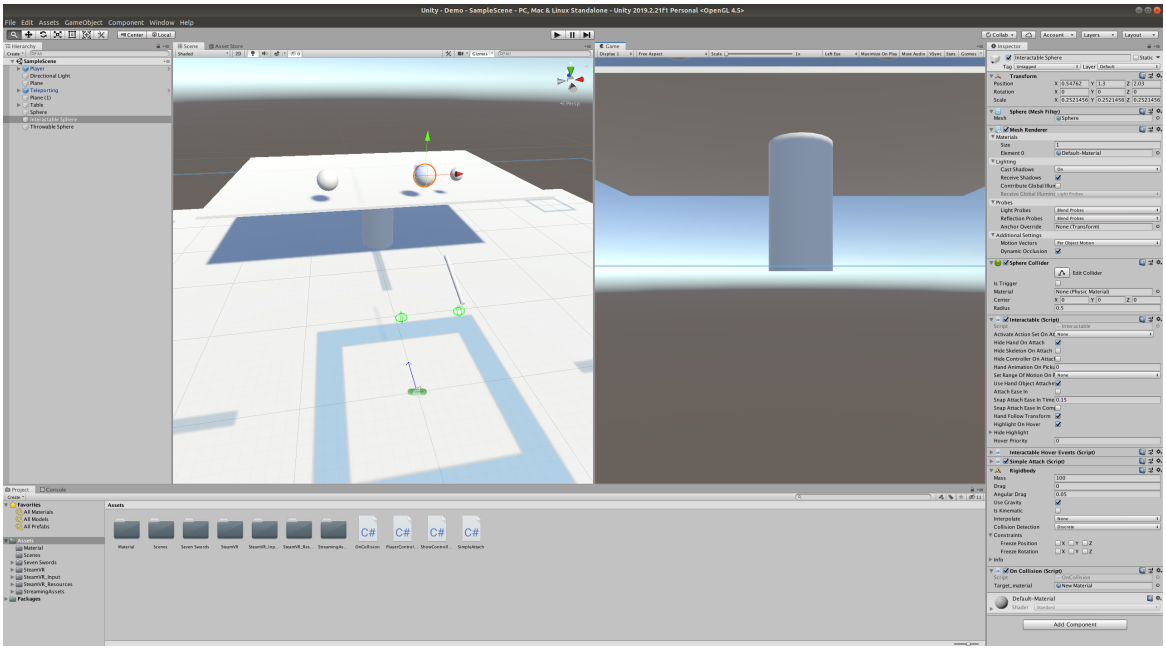

Figure 3.6: Unity editor interface

One of the most important reasons that we choose Unity as the simulated environment platform is its support for VR game design. It dominates the VR business by numerous tools built for VR creators and compatibility with VR devices. Unity fully supports SteamVR as Valve published the official Unity plugin to provide API to connect to and interact with VR devices. In short, HTC Vive Pro can be connected to Unity using SteamVR to develop and test VR games on Linux.

3.3 Communication

With the hardware and software decided, the next thing is how to communicate between different parts to build a complete system. The goals of our system are:

- The robot should be able to access the user's bat position and orientation.
- The user can compete with the robot in an virtual environment.

The first requirement can be solved by tracking the VR controller and passing the information to ROS; the other needs synchronization of the real PR2 robot in the simulated world designed by Unity.

3.3.1 VR and ROS

To communicate between ROS and the VR device, we need to first get the information of the VR device and then publish it to ROS. OpenVR SDK is created by Valve to provide greater integration to the SteamVR platform and its tools for the HTC Vive headset. A ROS package can be built on top of this to access the position and orientation published using this SDK. It is written in C++ that is also supported by ROS. An open source ROS package called vive_ros is used in our project. The processes to integrate HTC Vive Pro and ROS are:

1. Download and build Valve's OpenVR SDK.
2. Install Steam and SteamVR on the computer.
3. Connect HTC Vive Pro to the computer and configure the display.
4. Run SteamVR's server by using the vive_ros package.
5. Launch the node provided by vive_ros to publish transformation in ROS.

Figure 3.7 shows the result of accessing VR device information in ROS. Four transformations are visualized in rviz with different names. The two lighthouses represent the VR base station, which is fixed in certain location to track other devices. The hmd means the VR headset and the one left is the controller. Since the only information we require is the controller's position as well as orientation, simple transformation information will suffice. Note that here only the relative translation and rotation of the headset and controller to base station is accurate, the world coordinates and orientation should be specified manually to allow base station visualized in a correct way.

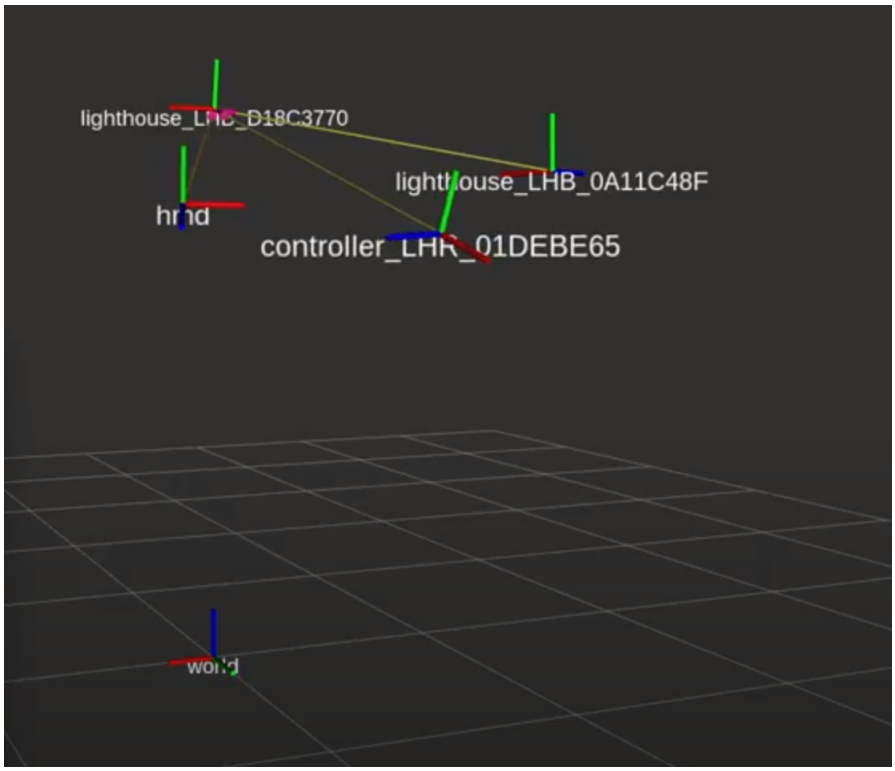

Figure 3.7: Demonstration of connecting HTC Vive Pro with ROS

3.3.2 ROS and Unity

After creating the VR environment using Unity, users can enjoy the immersive experience with the HTC Vive Pro Headset and interaction with the controller. However, to fulfill the requirements of human-robot interaction task, a robot, specifically the PR2 robot has to be modeled inside the Unity. Moreover, the movement of PR2 inside the Unity game has to be exactly the same as the real PR2 robot under control. ROS# is used in our research to connect ROS with Unity.

ROS# is a set of open source software libraries and tools in C# for communicating with ROS from .NET applications, in particular Unity. Basically it is a .NET solution for RosBridgeClient, URDF and MessageGeneration. Rosbridge is a ROS package to provide interface for non-ROS programs, and RosBridgeClient is a similar package to allow data transferred between ROS and Unity. URDF stands for Unified Robot Description Format, which is an XML format for representing a robot model that is fully supported in ROS. The model of robots in ROS simulation is represented by URDF and can be parsed and visualized easily, while Unity does not officially support this format. The URDF solution in ROS# can transfer a URDF from ROS to Unity with all structures maintained. Finally, the Message-Generation allows Unity to generate ROS messages, services and actions, making simulation synchronization between ROS and Unity possible. Figure 3.8 illustrates the general schema of ROS# controlling and visualizing the simulation in Unity side with actual steps happen in ROS side.

In our task, ROS is responsible to control the real PR2 robot; Messages of the states of PR2 will be published through the rosbridge websocket to Unity. As a result, synchronization of movements between real PR2 and Unity PR2 model can be achieved. Users can see PR2 moving in the virtual environment while expecting the real PR2 robot shares the same states. Sine the data that we transferred is only the position of the joints rather than large image files, the synchronization happens in real time with no perceivable delay. Figure 3.9 demonstrates the PR2 visualized in Unity but controlled by ROS. We can see the robot

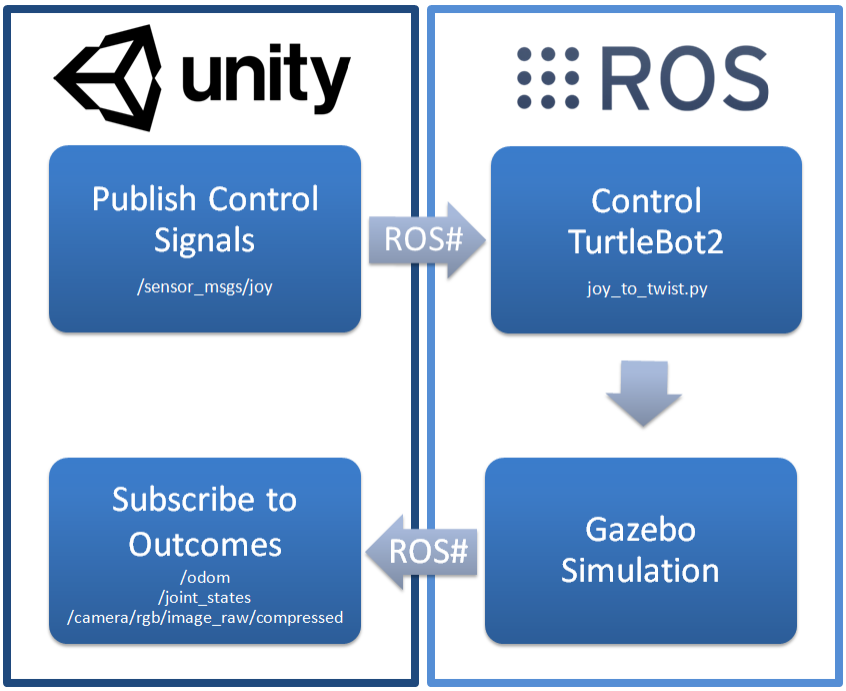

Figure 3.8: Simulation synchronization between Unity and ROS

model is correctly imported and moves correspondingly. Here for simplicity reasons, we did not actually control the PR2 in real time and synchronize in Unity, but played a recorded bag to publish the states of PR2. This does not hurt the result as in real experiments the same message types will be automatically published by ROS when controlling the PR2 robot, which is the only information we need.

3.4 Calibration

The calibration is essential in our system as the human-robot interaction task requires virtual environment with physical contacts. Although the communication strategies described above can sync between the real and the virtual world, the relative position of the user and the PR2 robot has to be determined to allow successful interactions. The VR device itself has to be calibrated also since it has more than one component. Overall, the calibration that

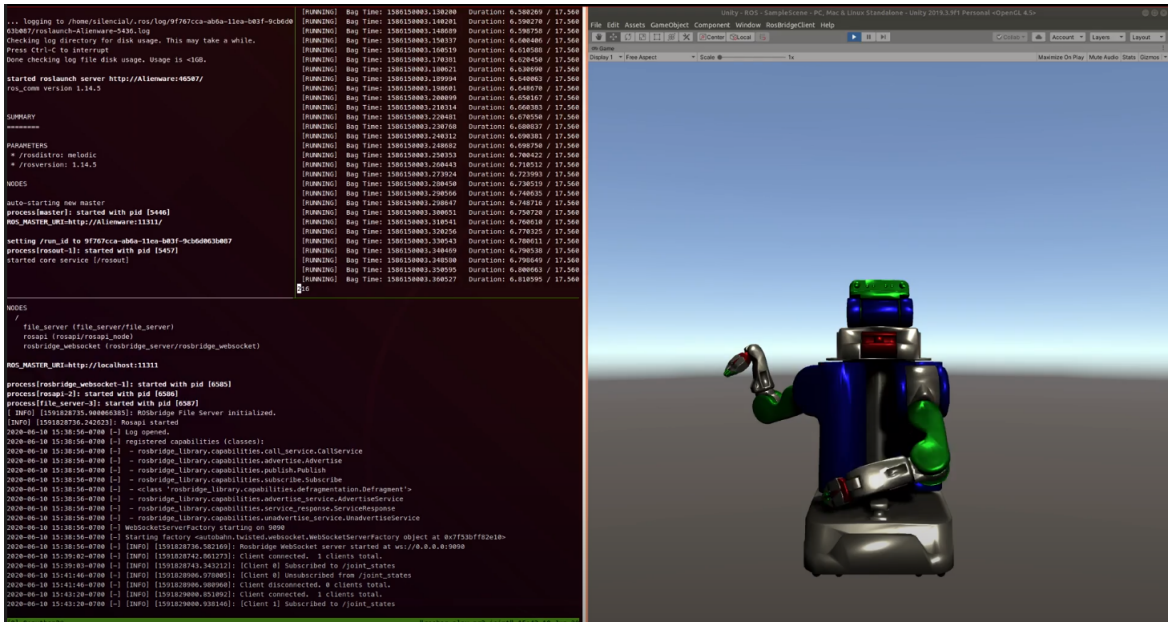

Figure 3.9: PR2 model and simulation visualized in Unity controlled by ROS

our system requires can be divided into two parts:

- Calibration of the VR device, e.g. base stations, headset and controllers.
- Calibration of the PR2 robot with respect to users position.

For the first part, SteamVR provides its own solution to calibrate on the first initialization. Figure 3.10 displays the interface when setting VR on the computer for the first time. Before running the first-time setup, headset needs to be connected to the computer; base stations have to be placed and fixed in appropriate positions to see both the headset and the controller. After SteamVR recognize all parts of the device, the setup can begin with first specifying the space requirements. There are two options: standing only and room-scale. While standing only does not have minimum space requirements and can be easily set up, the room-scale setup takes more time as a square space has to be allocated by following the instructions. The room-scale option can provide a more immersive experience for users as they can move

freely without worrying about safety issues. Visual information will be displayed in the virtual world as users are close to the boundaries of the space they specified. After the space setup, the software will ask users to calibrate the center of the space and the ground by manually input the height of the headset. The position and orientation of the headset and the controller will be accurately tracked by the base station without any operations.

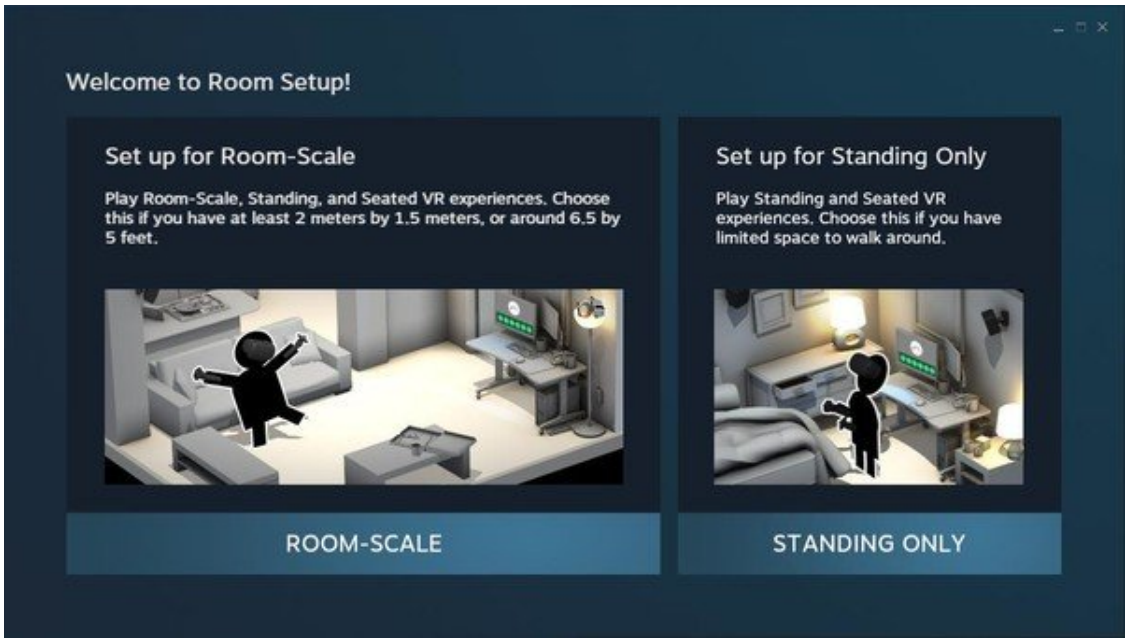

Figure 3.10: SteamVR calibration interface

The next step is to calibrate between the user and the real PR2 robot to be displayed correctly inside Unity VR environment. Since both the VR base station and the PR2 are fixed in certain places, this problem can be simplified to determine the translation and rotation relationship between these two objects.

Although there are numerous ways to achieve this purpose, we want to automated this process with existing hardware in our system. The PR2 robot has an Xbox 360 camera on its head, which is an RGBD camera as shown in Figure 3.11. The depth information captured by this camera can be used to calculate the distance.

ARTag is a fiducial marker system supported by augmented reality (AR). They can be

Figure 3.11: Xbox 360 camera

used for object detection and tracking that calculate a camera's position and orientation relative to physical markers. With the help of stereo cameras, the tracking can be achieve with high accuracy. Figure 3.12 shows some examples of the AR tag. ROS has its own package for AR tag tracking called ar_track_alvar and the tracking process are as follows:

1. Generate AR tags by the ar_track_alvar package, given size and ID.

2. Turn on the camera and publish the information to ROS.

3. Place the AR tag where the cameras can see and launch the provided tracking nodes.

4. Position and orientation of AR tags relative to the camera will be published if succeed, and can be visualize in rviz.

For our specific system, we can generate AR tags and attached it to the VR base station. Next we can control the PR2's camera to identify the position and orientation of the tags. Considering the transformation of the PR2 robot is known and fixed, we can easily calculate the relative transformation between the base station and the PR2 robot, thus calibrate the whole system.

Figure 3.12: AR tags examples

Chapter 4

REINFORCEMENT LEARNING

Reinforcement learning (RL) is a type of machine learning that aims at learning by interacting with an environment. It considers an agent in a game-like situation where at each time step, the agent takes an action, and receive observation and reward. The goal of RL algorithms is to maximize the reward by taking optimized action based on the observation information through a trial-and-error process. RL, among supervised learning and unsupervised learning, are three basic machine learning paradigms.

The idea of RL is not new and can date back to the early days of cybernetics. It is believed to be more natural and similar to how animals learn than other learning methods. However, the performance of RL at that time was severely limited by weak computer infrastructure. It was not until recent years did RL make a huge breakthrough by combining neural networks with RL to solve tasks that were nearly impossible in the past. This is called Deep Reinforcement Learning (DRL). Artificial intelligence designed using this method has achieved stunning results. For example, the famous AlphaGo is the first computer program to defeat a world champion in Go games, Atari57 beats humans benchmark at 57 classic Atari games with one single model, RL can even be used to train computers to play real-time strategy games like StarCraft and multi agents to compete in a hide-and-seek game in complex environment. These achievements, along with many other showcases, have proven that RL, especially DRL, is strong enough to solve complicated game-like tasks and suitable for our human-robot competitive game environment.

The most important feature of RL compared to other learning methods is that it can interact with the environment. Although the reward is given, there are no hints on how to solve the problem. The agent must take random strategies at first and update itself with the

information from the environment. Figure 4.1 illustrates the core idea of RL algorithms.

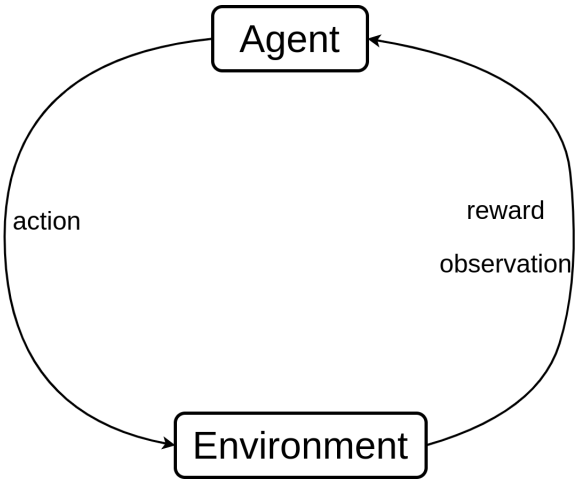

Figure 4.1: Reinforcement learning algorithm

Before diving into details of specific RL algorithms for our task, we will given mathematical representation of RL using a Markov decision process (MDP) model. A MDP describes the stochastic environment that the agent is interacting with. It comprises of 4 components:

- $\mathcal{S}$ is the state space;
- $\mathcal{A}$ is the action space;
- r is the reward function;
- $P(r, s'|s, a)$ is the transition probability distribution, which specifies the probability of the next state s' and reward r of the environment, given current state s and agent's action a

In $P(r, s'|s, a)$ we actually assume that the environment has Markov property, which means that the state dynamics only depends on the current observations. Any previous states, actions and rewards is not required to predict the next state and reward.

A policy or strategy π for an MDP is a mapping $\pi : \mathcal{S} \to \mathcal{A}$, where $\pi(s)$ is the action that the agent takes in states s. The goal is maximize the total reward received:

$$\mathbb{E}\left[\sum_{t=1}^{N} r(s_t, \pi(s_t))\right] \tag{4.1}$$

or if the game has infinite horizon, we are maximizing the average reward:

$$\lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} r(s_t, \pi(s_t))\right] \tag{4.2}$$

While in DRL, the policy is represented by a neural network with parameter θ, the goal is to find an optimal policy network that satisfies:

$$\theta^\star = \arg\max_{\theta} \mathbb{E}\left[\sum_{t} r(s_t, \pi(s_t))\right] \tag{4.3}$$

In this chapter, we will discuss the difference between reinforcement learning and supervised learning and the reason why we choose the former learning strategy. Then specific RL algorithms that we chose for our task will be presented starting from single agent to multi agents. Results will be given to see the feasibility and efficiency of the algorithms.

4.1 Supervised Learning

Supervised learning is one the most popular machine learning models. The goal of supervised learning can be simplifies as given input data X and output label Y, find a mapping $f : X \to Y$. The function approximator is usually represented by a neural network. A supervised learning algorithm requires labeled data, where each example is a pair consisting of an input object and an output value. Figure 4.2 shows an example of supervised learning in image classification tasks. The input here are images and the output are corresponding labels. A supervised learning algorithm to solve this task will first randomly initialize its neural network parameters and update using gradient descent when receiving the data. Since the data contains explicit output, the algorithm will know whether it is right or wrong and thus enhancing or correcting its knowledge about the input-output relationship.

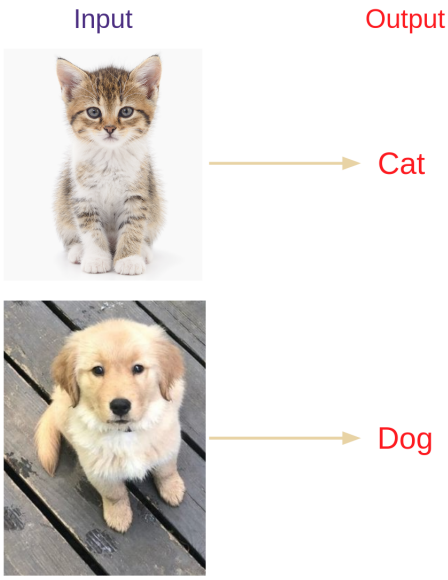

Figure 4.2: Image classification example

Usually the data will be split into training data and test data. A supervised learning algorithm can learn using the training data but the performance needs to be evaluated using the test data. This requires the algorithm has the ability to generalize the relationship presented in the training set and apply to unseen data. Although supervised learning has achieved satisfying results in many applications, there are certain task that cannot be solved by this technique. When the data does not consist of explicit input and output, like in most game-like environments, supervised learning can fail, or at least, far less efficient than other learning methods.

Take our fencing game as an example where the agent tries to hit the ball with its bat. The final goal is to come with a policy that output the action given current state to maximize the reward function. At first glance, this could be solved by taking states and actions as output. However, since the goal is to maximize the reward, we have to choose the action corresponding to the maximal reward, which requires large amount of data as the action space of this environment is huge. Much worse, higher current reward does not mean

higher total reward, which again increased the difficulty of collecting valid data, making it impossible to be solved by supervised learning.

Reinforcement learning, on the other hand, solves this issue by keeping a balance between exploration and exploitation. At first, the agent will interact with the environment to explore the reward structure; After experience is enough to reveal the correlations, the agent can further exploit the idea and test the performance. Since the agent can update itself after certain time, it can avoid selecting poor actions and only focus on the action space that is promising.

4.2 RL for Single Agent

In this section we will begin applying specific RL algorithms on our task. Understanding the single-agent RL algorithm is crucial before moving to multi agents as the environment and algorithm are similar.

4.2.1 Task Environment

First we will simplified our human-robot competitive fencing game to a single-agent version. Figure 4.3 illustrates the environment where a PR2 robot and a ball are presented. The goal of this game is to control the PR2 robot to hit the ball with its ball holding in one of its hand. Both the robot and the ball are fixed in certain positions without any changes during the game. The only thing we can control is the PR2's arm with 7 joints. Each joint has its own motor and can be controlled separately, however the position and the orientation of the bat are dependant of all these 7 joints. There is no way to explicitly know the information of the bat, which means the robot has to learn the kinematics of its joints to change the states of the bat.

Based on the rules given above, the four components specifically for our task are listed below:

- $\mathcal{S}$: states consist of the positions and velocities for 7 joints, $s \in \mathbb{R}^1 4$;

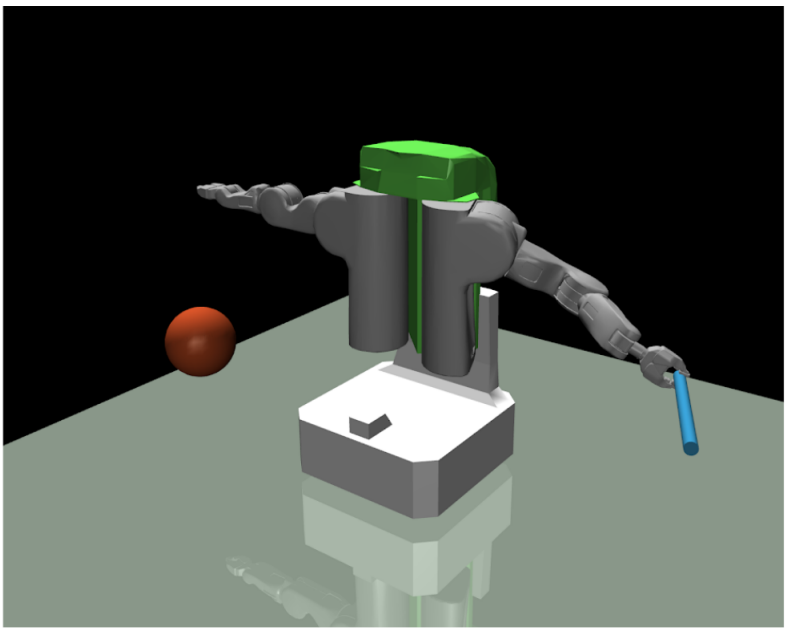

Figure 4.3: Single-agent environment in MuJoCo

- $\mathcal{A}$: actions consist of the torques we apply in 7 joints, $a \in \mathbb{R}^7$;
- r: the goal is to hit the ball with the bat;
- $P(r, s'|s, a)$: the dynamics of the system is specified inside the simulation environment.

Note that the reward function here is discrete since only when the bat hit the ball will the agent receive reward. Discrete reward is bad for RL algorithms for it gives no information at a large action space and the robot cannot learn anything from the beginning. To solve this problem, we redefine the reward function to become continuous:

$$r = w_1 I(contact) - w_2 dist(bat, ball) \tag{4.4}$$

where $I(contact)$ represents whether the ball has contact with the bat. It is 1 if there is contact and 0 otherwise. $dist(bat, ball)$ is the distance between the bat and the ball in the

Euclidean space. w_1 and w_2 are weights to distribute different importance to each term and both are positive values. We use $w_1 \gg w_2$.

Now the reward function is continuous, the robot will try to decrease the distance between the bat and the ball at the beginning to increase the reward. When the bat hit the ball, the robot will get a large reward to further update its policy.

4.2.2 Algorithm

In RL field, there are many algorithms available, each has its own advantages and limitations. Here our goal is not to compare the performance of different RL algorithms in our task, but rather test the feasibility of using RL algorithm to solve human-robot competitive task. Thus we choose a popular model called Deep Deterministic Policy Gradient (DDPG) [6] to solve this continous control task, then a multi-agent version of DDPG is applied in the next section.

Before discussing the details of DDPG algorithm, we have to first define some concepts for better understanding. First we define the Q-value function that evaluate how good a state-action pair is:

$$Q(s_t, a_t) = \mathbb{E} \sum_{i=t} r(s_i, a_i) \tag{4.5}$$

For the DDPG algorithm, we have two networks, the policy network to approximate the policy function π and value network to approximate the Q-function. Besides these two networks, we also have two lagged version target networks. For every update, the Q-function is updated by minimizing the mean square error with respect to the target value and the policy is updated by taking gradient of Q with respect the policy parameters θ. The detail of DDPG algorithm is given below:

Algorithm 1 Deep Deterministic Policy Gradient

Initialize policy π parameter θ and Q-function parameter ϕ
Set target network $\theta' \leftarrow \theta$, $\phi' \leftarrow \phi$
for episode = 1 to M **do**
    Observe state s and select action a according the the policy function
    Execute action and observe new states s' and reward r
    Store (s, a, r, s') in replay buffer $\mathcal{D}$
    $s \leftarrow s'$
    if time to update **then**
        Sample a random batch transitions $\{(s, a, r, s')\}$ from $\mathcal{D}$
        Compute target values

$$y' = r + \gamma Q_{\phi'}(s', \pi_{\theta'}(s'))$$

        Update Q-function by gradient descent using

$$\phi \leftarrow \phi - \alpha \sum_j \frac{\mathrm{d}Q_\phi}{\mathrm{d}\phi}(s_j, a_j) \left(Q_\phi(s_j, a_j) - y_j\right)$$

        Update policy network by gradient ascent using

$$\theta \leftarrow \theta + \beta \sum_j \frac{\mathrm{d}\pi}{\mathrm{d}\theta}(s_j) \frac{\mathrm{d}Q}{\mathrm{d}a}(s_j, a)$$

        Update target networks with Polyak averaging

$$\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$$
$$\theta' \leftarrow \theta + (1 - \tau)\theta'$$

    end if
end for

4.2.3 Results

In this environment, we choose 1000 timesteps for an episode and each timestep is 0.01 seconds. The graph of the training process is represented in Figure 4.4. As we can see, the robot starts with low reward but update itself fast to get higher and higher reward. There is a collapse around 100 episodes where the robot try to explore the action space but fail, then it recovers quickly to reach the highest reward and converge. The result shows that the agent can solve this task, which is in correspondence with the video when testing the trained

model. The robot will swing its arm quickly at first to minimize the distance between the bat and the ball, but stop immediately after it successfully hit the ball with the bat. Since the initialized positions of each game has small variations, the agent can take different strategies based on the joint states at first.

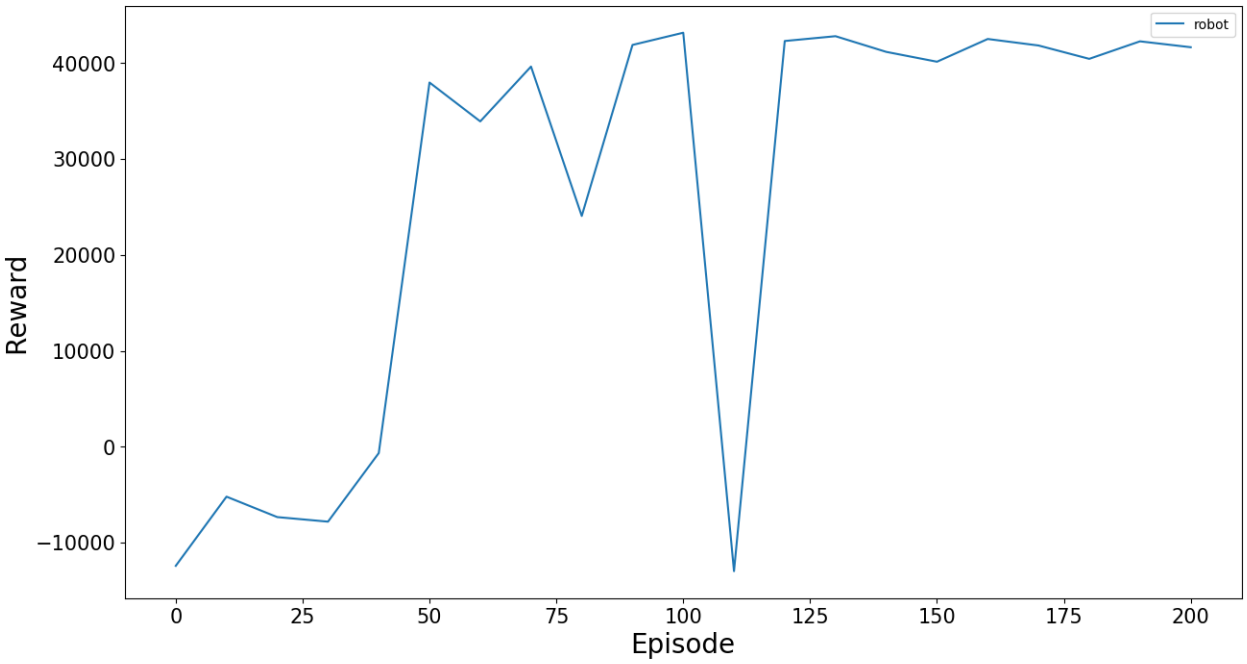

Figure 4.4: DDPG algorithm result

4.3 RL for Multi Agents

4.3.1 Task Environment

The task environment for multi agents is similar to single-agent version as shown in Figure 4.5. The difference here is that another agent with the same structure is presented in the environment. While one of the agent has the same goal in the single-agent environment to hit the ball with the bat, the other agent will try to defend the ball from being hitted.

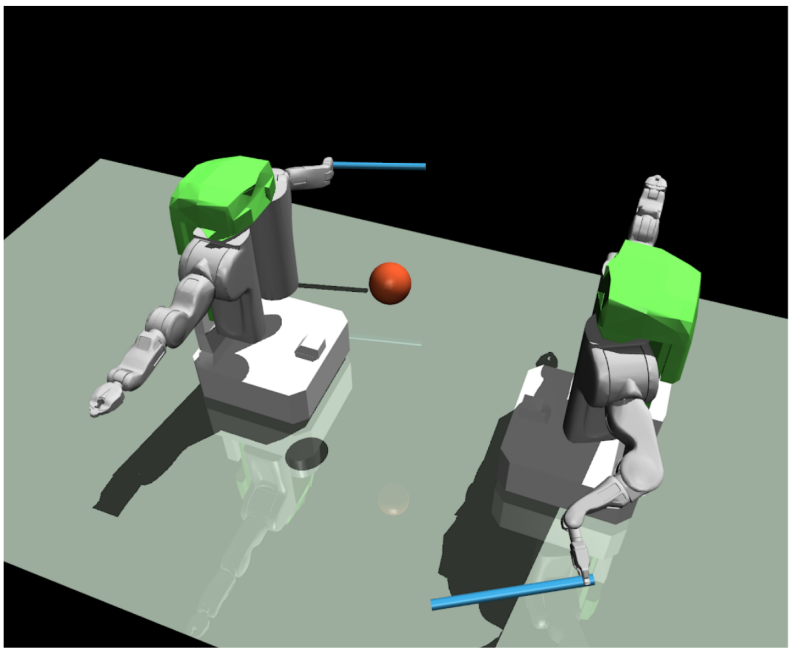

Figure 4.5: Multi-agent environment in MuJoCo

Similarly, we give the four components specifically for this multi-agent task:

- $\mathcal{S}$: states consist of the positions and velocities for 14 joints, 7 for each agent, $s \in \mathbb{R}^2 8$;
- $\mathcal{A}$: actions consist of the torques we apply in 14 joints, 7 for each agent $a \in \mathbb{R}^1 4$;
- r: the attacker's reward function stays the same, denoted by r_1, the defender's reward function is defined as $r_2 = -w_3 dist(bat, bat) - w_4 r_1$;
- $P(r, s'|s, a)$: the dynamics for each robot stays the same, with interactions dynamics added to the environment.

The main difference here is the definition of the defender's reward. Unlike the attacker's reward can be represented explicitly as minimizing the distance between the bat and the ball, the defender's reward is hard to defined. Here we represent the reward function to be

correlated with its competitor's reward: the defender will try to minimize the reward of its opponent. We also add an extra term as the distance between two bats as the only way to protect the ball from being hit is to block the attacker's bat with the defender's bat.

4.3.2 Algorithm

The algorithm we use here is the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [8] and is based on DDPG. Besides that we have two separate policy network and Q-value network for two agents, the main difference here is we change the Q-function to incorporate all the information to tell how good the state-action pair is. In another words, the Q-function considers both its own and the opponent's states and actions, and make decision upon that. The detail of MADDPG algorithm is given below:

Algorithm 2 Deep Deterministic Policy Gradient

Initialize policy π parameter θ_i and Q-function parameter ϕ_i for both agents
Set target network $\theta_i' \leftarrow \theta_i$, $\phi_i' \leftarrow \phi_i$ for $i = 1, 2$
for episode = 1 to M **do**
    Observe state s and select action a according the the policy function
    Execute action and observe new states s' and reward r
    Store (s, a, r, s') in replay buffer $\mathcal{D}$
    $s \leftarrow s'$
    if time to update **then**
        for every agent i **do**
            Sample a random batch transitions $\{(s, a, r, s')\}$ from $\mathcal{D}$
            Compute target values

$$y_i' = r_i + \gamma Q_{\phi_i'}(s', a_1', a_2')$$

            Update Q-function by gradient descent using

$$\phi_i \leftarrow \phi_i - \alpha \sum_j \frac{\mathrm{d}Q_{\phi_i}}{\mathrm{d}\phi_i}(s^j, a_1^j, a_2^j) \left(Q_{\phi_i}(s^j, a_1^j, a_2^j) - y_i^j\right)$$

            Update policy network by gradient ascent using

$$\theta_i \leftarrow \theta_i + \beta \sum_j \frac{\mathrm{d}\pi_i}{\mathrm{d}\theta}(s^j) \frac{\mathrm{d}Q_i}{\mathrm{d}a_i}(s^j, a_1, a_2)$$

        end for
        Update target networks for each agent i with Polyak averaging

$$\phi_i' \leftarrow \tau\phi_i + (1 - \tau)\phi_i'$$
$$\theta_i' \leftarrow \theta_i + (1 - \tau)\theta_i'$$

    end if
end for

4.3.3 Results

The graph of the training process is represented in Figure 4.6. The blue curve is the attacker's reward through episodes while the red one represent the defender's reward. We can see that the result corresponds to the reward function we defined. The increase of one agent's reward will inevitably decrease the other's. Since the reward function for the attacker is much simpler than for the defender, the attacker can still achieve a satisfying result. Compared

to the result of single-agent task, the attacker's reward is less stable, which means that the defender can sometimes succeed in blocking the attacker's bat. Occasionally the defender can prevent the opponent from hitting the ball as can be seen from the graph when the attacker has reward close to zero.

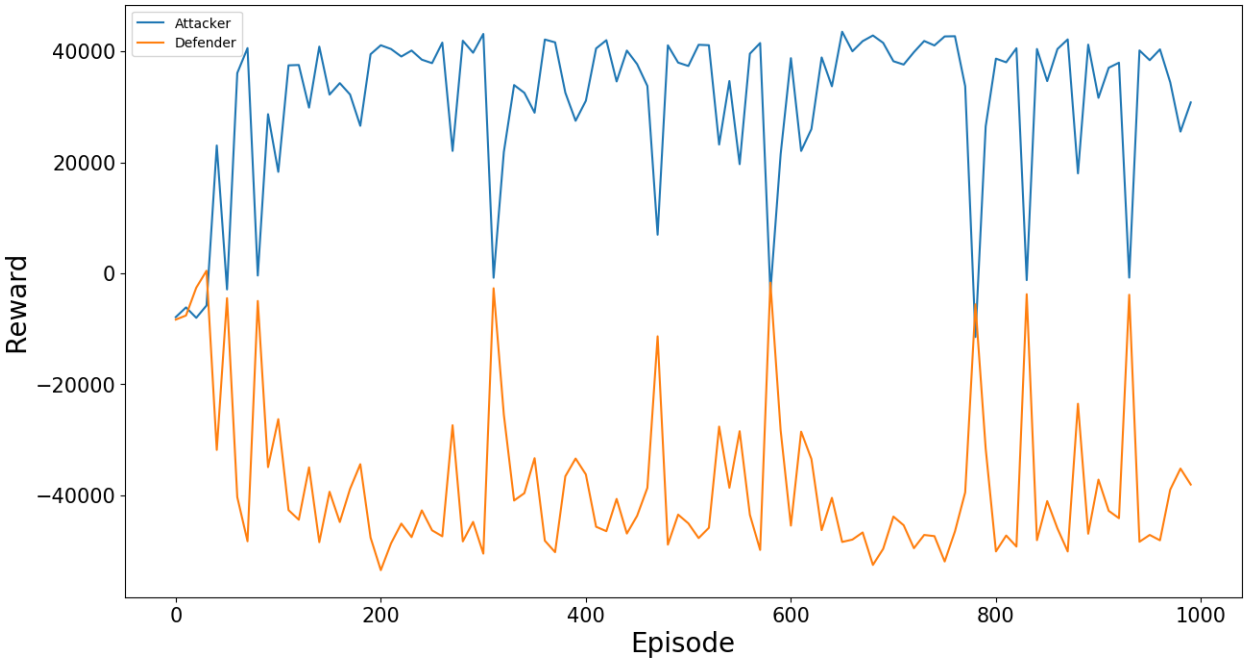

Figure 4.6: MADDPG algorithm result

The result is also in accordance with the expected robots behavior in real. It is much more difficult for the defender to succeed than the attacker. The defender requires much more complex strategies to beat its opponent while the attacker's goal is simple and straightforward. The correlations in the reward function make the simulation much trickier than in the single-agent task.

Chapter 5

CONCLUSIONS & FUTURE WORK

5.1 Conclusions

This thesis presents a platform design that can be used to study human-robot interactions and test the feasibility of reinforcement learning algorithm in multi-agent competitive game-like environment. In order to create a versatile framework for competitive games to be presented, we use virtual reality to create an immersive environment and synchronize the real robot's states to allow the game being played both in real and virtual world. This benefits us to create scenes in the software without concerning about the limitation in the real world. During the game, users can still have physical contact with real robots to ensure the same experience when competing with robots in the real environment, which cannot be achieved solely with VR.

A VR game is first created as a concrete example to support the system design and for the RL algorithm to apply on. The game is designed as competitive with two agents facing each other and a ball in the middle. Their roles in the game are represented as Attacker and Defender with different objectives. The game is compatible with the device and the robot including in the system and also straightforward enough to be learned by RL algorithms.

For the system part, various hardware and software are used to achieve different functionalities. PR2 robot is used as the opponent because of its human-like shape and body structures, together with its comprehensive support with ROS. Unity is responsible for creating the VR environment and interface with VR headset and controllers. HTC Vive PRO is chosen as the VR device in the framework for it has great extensibility and complete SDK for development purpose. ROS plays the essential part in this system as the interface to all the software and hardware for synchronization. PR2's model and states can be synced to the VR

environment through ROS; VR controllers positions and orientations can be accessed using ROS; RL models can control the robot by connecting to ROS. At last, calibration solution is proposed using AR tags and a RGBD camera mounted on PR2's head.

For the algorithm part, this work first shows that reinforcement learning can be powerful in solving complex continuous task with an example of single-agent version fencing game. Then for multi-agent competitive task, an updated version of RL model is used to test the feasibility. Although multi-agent reinforcement learning is tricky and still remains as an open problem, the result suggest that reasonable strategies can be learned with this model.

5.2 Future Work

The work presented in this thesis has its limitations. For example, the calibration of the system is not fully automated and should be done every time a new experiment starts. This can be cumbersome and time-consuming and should be further improved. In the RL section we mainly focus on the feasibility of using the general model in the task while there are many other specific algorithms other than the one we choose. The performance of different RL models on our task should be evaluated and compared.

As this work belongs to a larger project, some future progress in this work includes:

- Test the platform with real robots competing with users.
- Created a new or improved existing RL algorithms to better suite our task.
- Conduct user studies to compare virtual experience with real.
- Design more games to increase the robustness of the system.

BIBLIOGRAPHY

[1] Cynthia Breazeal and Andrea L Thomaz. Learning from human teachers with socially guided exploration. In *2008 IEEE International Conference on Robotics and Automation*, pages 3539–3544. IEEE, 2008.

[2] Tiffany L Chen, Matei Ciocarlie, Steve Cousins, Phillip M Grice, Kelsey Hawkins, Kaijen Hsiao, Charles C Kemp, Chih-Hung King, Daniel A Lazewatsky, Adam E Leeper, et al. Robots for humanity: using assistive robotics to empower people with disabilities. *IEEE Robotics & Automation Magazine*, 20(1):30–39, 2013.

[3] Aaron St Clair and Maja J Matarić. Task coordination and assistive opportunity detection via social interaction in collaborative human-robot tasks. In *2011 International Conference on Collaboration Technologies and Systems (CTS)*, pages 168–172. IEEE, 2011.

[4] David Feil-Seifer and Maja J Matarić. Socially assistive robotics. *IEEE Robotics & Automation Magazine*, 18(1):24–31, 2011.

[5] Michael A Goodrich and Alan C Schultz. *Human-robot interaction: a survey*. Now Publishers Inc, 2008.

[6] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[7] Oliver Liu, Daniel Rakita, Bilge Mutlu, and Michael Gleicher. Understanding human-robot interaction in virtual reality. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 751–757. IEEE, 2017.

[8] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390, 2017.

[9] Elias Matsas and George-Christopher Vosniakos. Design of a virtual reality training system for human–robot collaboration in manufacturing tasks. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 11(2):139–153, 2017.

[10] Aaron Steinfeld, Terrence Fong, David Kaber, Michael Lewis, Jean Scholtz, Alan Schultz, and Michael Goodrich. Common metrics for human-robot interaction. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 33–40, 2006.

[11] Kyle Strabala, Min Kyung Lee, Anca Dragan, Jodi Forlizzi, Siddhartha S Srinivasa, Maya Cakmak, and Vincenzo Micelli. Toward seamless human-robot handovers. *Journal of Human-Robot Interaction*, 2(1):112–132, 2013.

[12] Andrea L Thomaz and Maya Cakmak. Learning about objects with human teachers. In *2009 4th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 15–22. IEEE, 2009.

[13] Vaibhav V Unhelkar, Ho Chit Siu, and Julie A Shah. Comparative performance of human and mobile robotic assistants in collaborative fetch-and-deliver tasks. In *2014 9th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 82–89. IEEE, 2014.

[14] James E Young, JaYoung Sung, Amy Voida, Ehud Sharlin, Takeo Igarashi, Henrik I Christensen, and Rebecca E Grinter. Evaluating human-robot interaction. *International Journal of Social Robotics*, 3(1):53–67, 2011.

[15] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.