Master Thesis

EFFICIENT REINFORCEMENT LEARNING IN ROBOTICS MANIPULATION CONTROL BY MIMICKING REAL WORLD ENVIRONMENT WITH SIMULATOR

Huong Van Do

HCI - Robotics

UNIVERSITY OF SCIENCE AND TECHNOLOGY

June 2019

EFFICIENT REINFORCEMENT LEARNING IN ROBOTICS MANIPULATION CONTROL BY MIMICKING REAL WORLD ENVIRONMENT WITH SIMULATOR

Huong Van Do

A Dissertation Submitted in Requirements
For the Degree of Master in major of HCI - Robotics

June 2019

UNIVERSITY OF SCIENCE AND TECHNOLOGY
Supervisor Dr. Lee Woosub

Declaration of Student

I, HUONG VAN DO, declare that the title of thesis, 'EFFICIENT REINFORCEMENT LEARNING IN ROBOTICS MANIPULATION CONTROL BY MIMICKING REAL WORLD ENVIRONMENT WITH SIMULATOR' with the tasks presented in it are my own. I confirm that:

- This work was done mainly while in candidate for a master degree at this University.
- When I consulted the published work of others always clearly attributed.
- When I quoted from work of others, the source was given. This thesis entirely is my own works.
- I have made clear what was done by myself and what was done by others.

Signed: ______________________________

Date: ______________________________

We hereby approve the M.S.
Thesis of "Huong Van Do"

June 2019

Chairman of Thesis Committee

Signature

Thesis Committee Member

Signature

Thesis Committee Member

Signature

UNIVERSITY OF SCIENCE AND TECHNOLOGY

“From too much study, and from extreme passion, cometh madness.”

— Isaac Newton —

UNIVERSITY OF SCIENCE AND TECHNOLOGY

Abstract

Major: *HCI - Robotics* in Department of *Media and Robotics Institute*
Korea Institute of Science and Technology

Master of Engineering

by Huong Van Do

Efficiently Reinforcement Learning in Robotics Manipulation Control by mimic real world environment with simulator. In this thesis will build and develop the software and hardware for implementation Reinforcement Learning in Manipulator control with using ROS programming. Recent achievement in Deep Neural Network allows to apply machine learning algorithms to various fields including vision, recognition and robotics[6]. Adapting deep neural network to train the policy of Reinforcement Learning successfully solved manipulator or navigation tasks. However, the training process requires numerous experiments to repeat the task until finding satisfying policy. To avoid this expensive experimental training, we implemented a hardware and software test-bed to share the control and training process. Some test-beds were taken to demonstrate the consistence between simulation and real world. The result of performance was tested under the Monte-Carlo algorithm. The software platform was built in ROS Programming (Robot Operating System) for implementation learning algorithms. . . .

Keywords: *Reinforcement Learning; Markov Decision Processes; Monte-Carlo; Deep Deterministic Policy Gradient; Hindsight Experience Replay; ROS; Gazebo; OpenAI; Q-Learning; UR5.*

Acknowledgements

I owe my gratitude to all the people who have made this thesis possible and who have made my graduate experience one that I will cherish forever.

First and foremost, I would like to thank my advisors, Professor Dr.Lee Woosub and Dr.Kim Soonkyum, without you this thesis would not have been possible. I would lilke to thank you for giving me all the freedom in the research. I would also like to thank my professor for his patience in listening to my proposed ideas and my problems and then supporting me in all my adventures during the two years of my graduate studies at KIST. Your advice, as well as your kindness, definitely had a huge impact on this thesis. To be honest, for me, Dr.Kim and Dr.Lee are more than an academic advisor. The way you work and stories you shared inspire the students have more motivate to become a professor in the future.

I would like to thank Dr.An Byungchul for his insightful knowledge in some research problems we discussed. It helps me a lot when I struggle to evaluate the effective of some ideas. In addition, I would like to thank other colleagues in Center for Medical Robotics for their helps and it help me have an unforgettable time in the lab and in KIST.

I acknowledge Korea Institute of Science and Technology for the financial support during my studies.

Last but not least, I wish to thank my parents and my girlfriend. Their unconditional supports absolutely bolster my motivation. By dint of complete this master thesis will be a great meaningful gift of me would like to spend to my whole family and my girlfriend . . .

— Huong Van Do, Seoul 30th May, 2019 —–

Contents

List of Figures

Abbreviations

MDP	**M**arkov **D**ecision **P**rocesses
DRL	**D**eep **R**einforcement **L**earning
RL	**R**einforcement **L**earning
HSV	**H**ue **S**aturation **V**alue
PCL	**P**oint **C**loud **L**ibrary
RGB	**R**ed **G**reen **B**lue
ROS	**R**obot **O**perating **S**ystem
TCP	**T**ransmission **C**ontrol **P**rotocol
XML	E**X**tensible **M**arkup **L**anguage
DP	**D**ynamic **P**rogramming
URDF	**U**niversal **R**obotic **D**escription **F**ormat
PID	**P**roportional **I**ntegral **D**erivative
GPI	**G**eneralized **P**olicy **I**teration
ES	**E**xploring **S**tarts

Chapter 1

Introduction

1.1 Motivation.

Nowadays, with the development of technology and the significant achievement in reinforcement learning researching, especially in the high-dimensional problems such as Atari games games [6] and Go [7]. Regarding to the robotics field was saw a huge benefit from this technique to get a better result on robotics tasks. In addition, the transfer learning control policies from simulation to reality still in challenge due to the uncertainties of dynamic and environment[10] and real observation[11].
Reinforcement learning(RL) has been achieved outstanding results in virtual environments such as GO, Atari games, etc. Although many kinds of research applying RL to real hardware systems are conducted based on these results not to come up to those achieved in simulation environments. Since the simulation environments cannot model the real-world perfectly, it is natural that real-world experiments do not reach those results. To overcome the limitation of simulation, there are many attempts to train a policy using real hardware.
With the development of technology and software aid in design and programming nowadays, there are simulator software for robotics have been developed in the main focus on complexity, accuracy, and flexibility in manipulation task. By dint of using Robot Operating System (ROS) and Gazebo[1] can support for simulate robotics structures.However, some simulators have limited by two-dimensional or the interact between robot and the environment just in approximately dynamics.ROS contained libraries and tools for creating the application of robot. Playing a part of the Player Project[5], Gazebo provides 3D simulation tools. Thanks to a well-designed in simulator for making rapidly test algorithms, robot designs or perform regression testing even in training Artificial Intelligent (AI) by using realistic scenarios. Moreover, Gazebo provides the

ability in efficiently simulate and accurate of robots in both of indoor and outdoor environments.
In this thesis is aimed to apply a framework to share the control and training process for Robotics system. In order to do that, we mimicked the real-world physics in simulation environment to avoid take experiment with real robot. By dint of comparing the result of training a simple ball putting problem both simulation and in the real world, we can see how consistence between simulation to the real. In detail, we completed to build a virtual environment for UR5 (Universal Robot) with golf putting task. This is a task of pushing an object on the table to a determined target from a randomly selected start position is trained using the Monte-Carlo algorithm[32]. The trained policies are testing on simulation.

1.2 Contributions

The thesis is divided into two distinguish parts, but their common background mainly relate to reinforcement learning theory and control. In the first part consisting of chapter 3 is mainly related with how to build virtual environment and implement the reinforcement learning algorithms of robot and the environment. The following parts belongs to how transfer the result from simulation into the real hardware, the result will be shown in the chapter 5.

1.2.1 Test-bed for Reinforcement Learning algorithms

In chapter 2, we rely on how to transfer the result of training from software into the real environment. By dint of policy output from the virtual environment we can execute that directly in manipulator and check the output accordingly.

1.2.2 Building virtual environment for Reinforcement Learning algorithms

In chapter 3, we rely on how to build the virtual environment with the aid of software and mimic the real environment by the 3D model.
In addition, we implemented the reinforcement learning algorithms - the Monte-Carlo algorithm - in virtual environment to get optimal policy and control.

1.3 Reinforcement Learning

1.3.1 Fundamental

The interaction between the agent with the environment were described by the figure 1.1. In basically, the reinforcement learning system contains four main elements: *a policy, a value function, a reward signal and the model of the environment.*

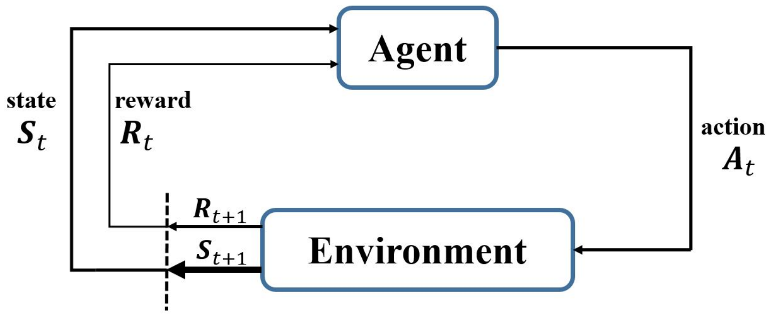


FIGURE 1.1: The graph shows the reinforcement learning system.

A policy describe the behavior of agent with the environment at a time. Beyond the given states of the environment, a policy will take a map from set of states to the set of actions accordingly. The policy is the core of the agent to determine behavior with the environment. It is can be deterministic or stochastic of policy in reinforcement learning.
A reward signal reflect how good or bad of agent while interact with the environment. It is a goal for the agent to maximize the value of reward signal, it is a scalar data.
The *value function* describes how status in each state of the agent for the long run. In each state will have their own value function which mean the value of the expected reward value will be received from these states until the terminate state.
model of the environment is the last element in the reinforcement learning problem which allows how the environment will behave with the agent. Nowadays, main methods for solving reinforcement learning rely on use models or planning.

1.3.2 Markov decision processes definition

In this section will introduce the most essentials we will use in reinforcement learning formulation, that is the theory of Markov Decision Process (MDPs)[36]. All of the equations explanation and mathematically formulation is reference from book *"Algorithms for Reinforcement Learning"*[36].

1.3.2.1 Formulation of Markov Decision Processes

MDP defined with $\mathcal{M} = (\mathcal{X}, \mathcal{A}, \mathcal{P}_0)$, at here $\mathcal{X}$ stand for non-empty set of states. $\mathcal{A}$ stands for non-empty set of the actions. The transition probability of the state-action pair $(x, a) \in \mathcal{X} \times \mathcal{A}$ is $\mathcal{P}_0$. The transition probability of the next state with reward will belong to the set of Z from the current state x and action a was taken - $\mathcal{P}_0(Z|x, a)$, discount factor $0 \leq \gamma \leq 1$.
$\mathcal{P}$, the *state transition probability kernel* for the set of $(x, a, z) \in \mathcal{X} \times \mathcal{A} \times \mathcal{X}$ show the probability from state x to state z by action a:

$$\mathcal{P}(x, a, z) = \mathcal{P}_0(z \times \mathbb{R}|x, a).$$

The immediate reward function r: $\mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ at the action a was taken from state x, then:

$$r(x, a) = \mathbb{E}[\, R_{(x,a)}]$$

For any for any $(x, a) \in \mathcal{X} \times \mathcal{A}, |R_{(x,a)} \leq \mathcal{R}$. From that we have the *return* behavior is defined by the total discounted sum of the rewards:

$$\mathcal{R} = \sum_{t=0}^{\infty} \gamma^t R_{t+1} \tag{1.1}$$

The goal for reinforcement learning with Markov Decision Process is choose the behavior for maximize the expected of return. Only when we reach the maximize expected return, we can get the optimal behavior of agent.

1.3.2.2 The value functions

In MDP formulation, we need to find optimal behavior of the agent rely on the highest value of each state. The great idea is calculate the value function. $V^*(x)$ is called as an optimal value function of state $x \in \mathcal{X}$. Regarding to *deterministic stationary policies* show a special class of behavior in then agent. In fact, that is mapping π of states and actions from:

$$A_t = \pi(X_t) \tag{1.2}$$

We fix policy $\pi \in \prod_{stat}$, π is defined:

$$V^\pi(x) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t . R_{t+1} | X_0 = x\right], \quad x \in \mathcal{X} \tag{1.3}$$

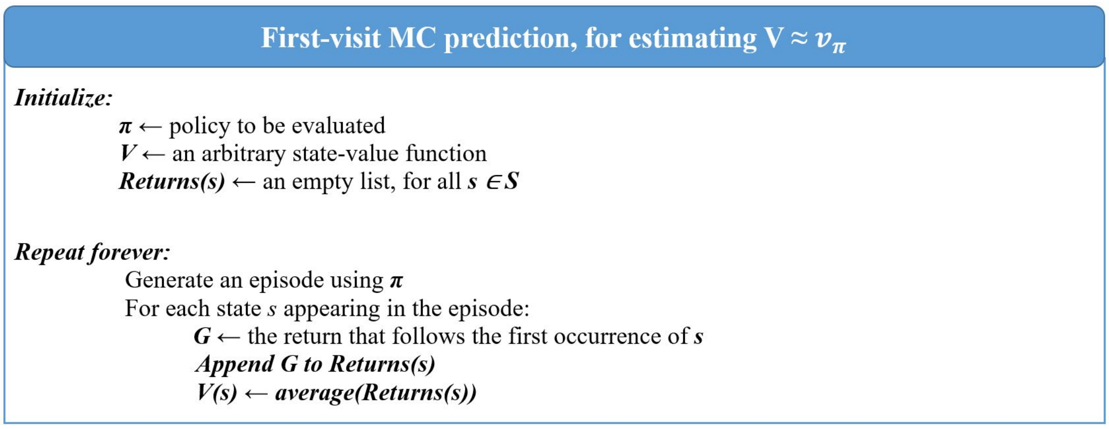
From the equation 1.3 we need to maximize the expected reward in each states for the fix policy.

1.3.3 Monte-Carlo method

From this environment I used the value iteration methods to discover optimal policies in golf-putting. Because of the simple of Monte-carlo algorithms it just require experience from states, actions and rewards by interact with the environment[32].

1.3.3.1 Monte Carlo Prediction

From the deterministic policy, the Monte-carlo methods will learning the state-value function. It is simply to average the return expected value from experience after visits from that states. That mean the more return of observed, the more average should converge in this algorithm. We looking for estimate $v_\pi(s)$ from policy π. By dint of recurrent of state s that agent visited in episode, we called that is a visit s. In the Figure1.2 below show how Monte-carlo algorithm works for optimal $v_\pi(s)$.



First-visit MC prediction, for estimating V $\approx v_\pi$

```
Initialize:
        $\pi \leftarrow$ policy to be evaluated
        $V \leftarrow$ an arbitrary state-value function
        $Returns(s) \leftarrow$ an empty list, for all $s \in S$

Repeat forever:
        Generate an episode using $\pi$
        For each state $s$ appearing in the episode:
            $G \leftarrow$ the return that follows the first occurrence of $s$
            Append G to Returns(s)
            $V(s) \leftarrow average(Returns(s))$
```

FIGURE 1.2: First-visit MC prediction, for estimating value function[32]

1.3.4 The Monte Carlo Control

In this case we analysis how Monte-Carlo use in control to approximate the policy optimal. The idea behind this is use to the same pattern of DP, that mean generalized policy iteration (GPI). By dint of keeping the approximate value function and the approximate policy. The value will be repeated to close to optimal value. Besides, it is the same with the policy repeated to improved with current value function to reach the optimal value function.

From the policy greedy with respect the current state value function, the policy will be improved to the optimal value. Therefore, for any action-value function q we can determine for chose action based on maximize value of action-value function.

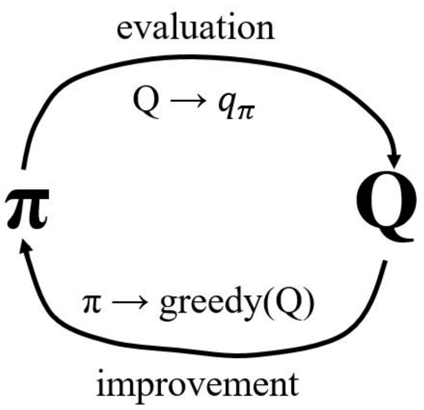


Figure 1.3: The greedy policy[32]

$$\pi(s) = \underset{a}{argmax} q(s,a). \tag{1.4}$$

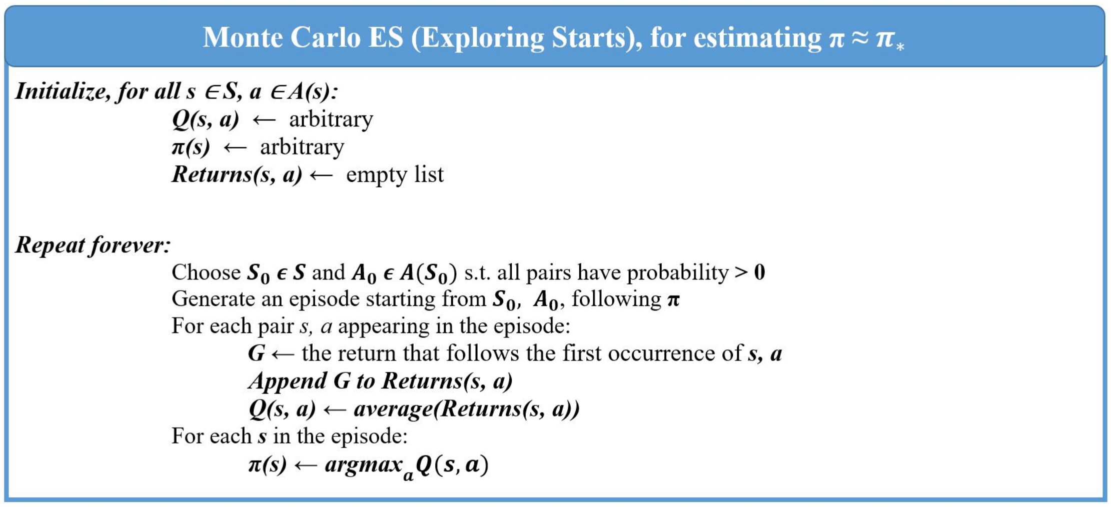
The greedy policy from q_{π_k} was improved to the π_{k+1}. This improvement will be apply to π_k and π_{k+1}, for all $s \in \mathcal{S}$:

$$q_{\pi_k}(s, \pi_{k+1}(s)) = q_{\pi_k}(s, \underset{a}{argmax} q_{\pi_k}(s,a)) \tag{1.5}$$

$$= \underset{a}{max} q_{\pi_k}(s,a) \tag{1.6}$$

$$\geq v_{\pi_k}(s). \tag{1.7}$$

For Monte-Carlo policy evaluation that is a natural for valuation on episode-by-episode basis. From observed value return for policy, the policy will be improved in all states in the episode. In the figure 1.4 shows the detail of algorithm.



Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

```
Initialize, for all $s \in S$, $a \in A(s)$:
        $Q(s, a) \leftarrow$ arbitrary
        $\pi(s) \leftarrow$ arbitrary
        $Returns(s, a) \leftarrow$ empty list

Repeat forever:
        Choose $S_0 \in S$ and $A_0 \in A(S_0)$ s.t. all pairs have probability $> 0$
        Generate an episode starting from $S_0$, $A_0$, following $\pi$
        For each pair $s$, $a$ appearing in the episode:
            $G \leftarrow$ the return that follows the first occurrence of $s$, $a$
            Append $G$ to $Returns(s, a)$
            $Q(s, a) \leftarrow average(Returns(s, a))$
        For each $s$ in the episode:
            $\pi(s) \leftarrow argmax_a Q(s, a)$
```

Figure 1.4: Monte Carlo ES (Exploring Starts)[32]

1.4 Outline of master thesis

The thesis contains of five chapters were included the current chapter. We would like to briefly summarize the thesis as follows. This thesis contains six parts. In part I contains the introduction chapter, which sets the motivation and outlines the contribution for part III and IV. In part I will show out the fundamental of reinforcement learning algorithm after that the description of the Monte-Carlo policy gradient algorithm will be implemented in this thesis belong to chapter II, after that is the formulation of the golf-putting problem under reinforcement learning mathematically. Part III presents the real hardware environment and set up the real test with golf putting stuff. Then, part IV presents for building the virtual environment based on mimicking the physic hardware and real environment by dint of ROS programming language and execute reinforcement learning algorithm. Each chapter in part III and part IV contain the problem formulation, motivation, mathematically analysis, numerical simulations. In addition, part V makes a summary and discusses some further promising directions with current problem. Finally, in chapter VI will be the summarize of this thesis and some conclusion from this thesis for student and promising of research in the future.

Chapter 2

Problem definition

2.1 Problem formulation

In sections above shown out the general definition of reinforcement learning algorithm and how to solve problems related with reinforcement learning from implement simple algorithm - Monte Carlo. In this thesis I worked with build a virtual environment of the golf putting task. In this environment has a manipulator plays as an agent and the set of action will be count the distance from end-effector of manipulator UR5 (Universal Robot) to the ball object. The golf platform and the golf ball will play as an environment to interact with the agent. When the end effector of manipulator will push the golf ball, if the golf ball move to the goal position, the agent will be received a reward, otherwise the agent will be punished. The detail of the action set has 4 parameters inside: $(l_1, l_2, \theta_1, \theta_2)$. In Figure 2.1 shown how to define the action set with 4 parameters and the set of state will be inside the blue circle to represent the initial state of golf-object.
As below is the mathematically definition of the problem in golf putting under rein-

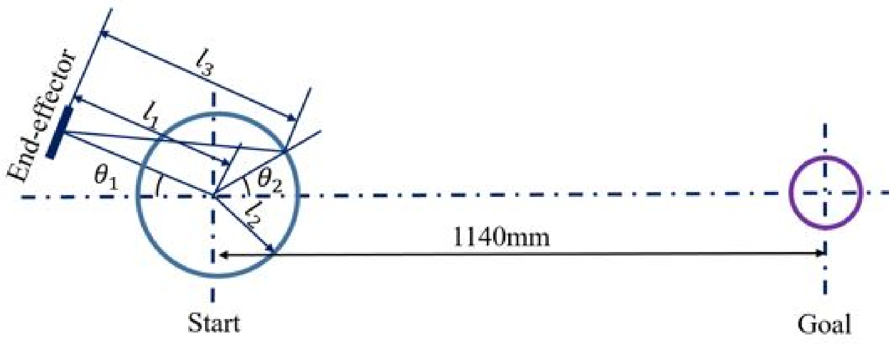


FIGURE 2.1: Illustration the set of action and state.

forcement learning formulation.

Action set

$$(l_1 = [0.3; 0.35]; \;\; l_2 = [0.15; 0.25]; \;\; \theta_1 = [-8^o; 8^o]; \;\; \theta_2 = [-8^o; 8^o])$$

The dimension of action set equal to 4 stands for 4 parameters.

Set of state

From figure 2.1, the coordinate of the initial start state object will be inside the blue circle with radius equal to l_2 with respect to (x, y). The goal state will be inner the goal circle from figure 2.1. In this case, the dimension of state equal to 2.

The distance threshold

Defined by the final position of golf-ball after push by end-effector of manipulator with the goal position. It can be called a success when the distance is ≤ 0.1(m). Thanks to this value, do we can evaluate the success of manipulator task or not.

The reward function

In order to work with reinforcement learning we need to define the reward function to evaluate how good of action in agent with the environment or not. In this case I used with sparse reward value. It will be calculated by $-d$ (d is the distance between golf ball and the target position.

2.2 Strategy and algorithm implementation

From the theoretical analysis of reinforcement learning and problem formulation of the golf putting task. I would like to propose strategy to solve with the current problem under the Monte-Carlo policy gradient algorithm as below.

From the actor-critic methods, we will have the definition as below:

- **Critic** will updates the value function of parameters w. It can be a action-value $Q_w(a|s)$ or a state-value $V_w(s)$.
- **Actor** will updates the policy parameters θ for $\pi_\theta(a|s)$, the actor function will depend on the critic to get update value.

We use two neural networks: a *target policy*, action-value function is *critic*$\mathcal{Q} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. At here, the critic will play a role for make approximate actor's value function Q^π. From the figure 2.2, we can the model mapped the states to the actions.

Network structure

In figure 2.3 we can see the dimension of input state dimension and the output of action

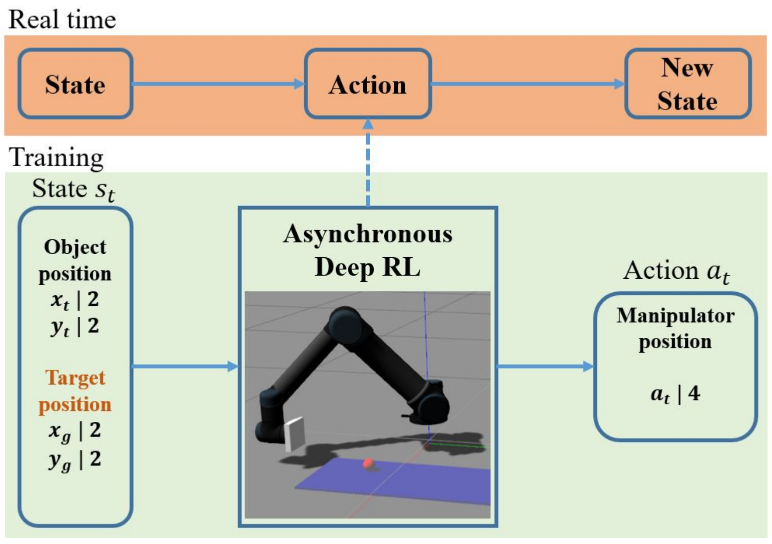


FIGURE 2.2: A map-less action was trained through asynchronous deep-RL.

dimension. In here, the dimension of state input is 2 and the dimension of action is 4. In both network also used 2 hidden layers with 512 nodes and activation function is ReLU function, it was a fully connected network. After that it was blobs into merge layer for action value are the position and angle of end effector in manipulator. At here, we used sigmoid function to calculate output of position in manipulator. For the angle we used tanh function to calculate the angle of end effector of manipulator. In the critic network is similar with the actor network for the output is used the linear function to the value of Q function.

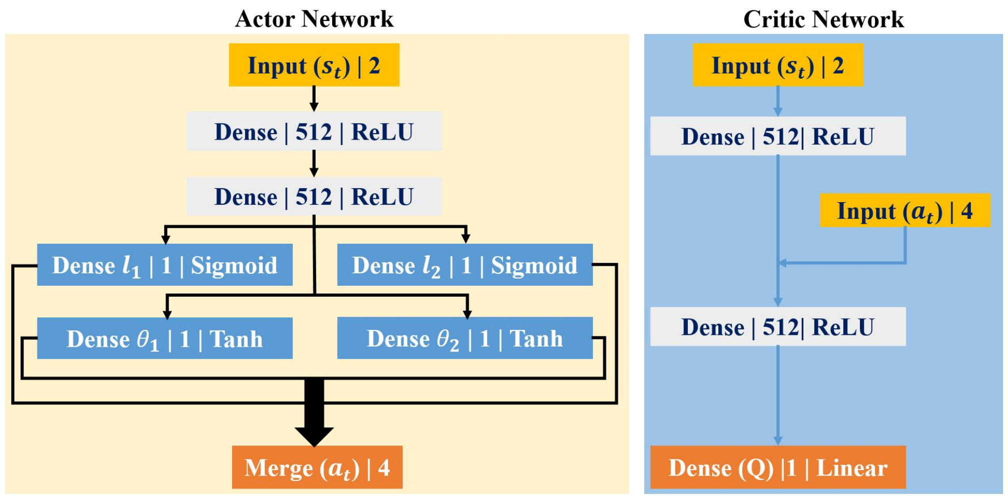


FIGURE 2.3: The network model for the DDPG model.

2.3 Summary

From the analysis of implement the Monte-carlo algorithm for optimal the value function and policies by exeprience. We will shows out some advantages by this algorithms:

- Firstly, learn optimal behavior from experience and interact with the environment. The environment's dynamics is not needed.
- Secondly, they can be used with simulation or *sample models*.
- In the last, with the small set of states and actions the Monte-carlo methods is more efficient.

In this thesis I implemented the Monte Carlo algorithm to find out the best policies for golf putting in both on the virtual environment and real hardware with the result will be shown in the following chapters.

Chapter 3

Hardware for Test-bed with Reinforcement Learning algorithms

From the problem definition with the golf putting stuff, we come up with the determine the hardware structure first after that we will mimicking that environment. In this chapter we will shows out the hardware setup of the physical environment. How to mimicking the real world will be shown in the next chapter with software structure and libraries accordingly.

3.1 Vision Based System

In order to work with determine position and detect the goal ball object, the vision system was used by a camera Intel Realsense D435[34]. A specific object recognition systems was taken. The first system operates on color images and mainly use facilities of the OpenCV library[25]. The last system operates on point clouds and mainly uses facilities of the PCL library[26]. This chapter describes the underlying algorithm of each of the two system for each solution.

3.1.1 Depth camera Intel Realsense D435

Regarding to the vision system, a depth camera Intel Realsense D435 was used to tracking the object and send information of position of object to the controller. In this camera was used stereo vision to calculate depth. It was powered by the USB-power depth camera and consist of pair depth sensors, RGB sensor. In the following is some specification of realsense camera:

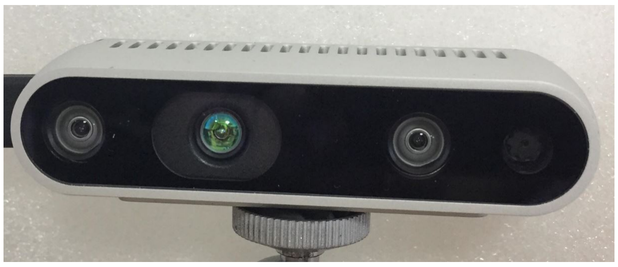


FIGURE 3.1: Camera Intel Realsense D435.

- The vision processor uses 28 nanometer (nm) and support up to 5MIPI[30]
- Advance stereo depth for accurate depth perception.
- Capturing the disparity between images up to 1280 x 720 resolution.
- Support for the cross-platform.
- The signal processor for image adjustments and scaling color.

3.1.2 HSV detection algorithm

The main idea of algorithm as in Figure 3.2. The HSV detection system detects objects with noticeable colors. The naive approach, filtering images on RGB values, since RGB values are strongly affected by the overall brightness[27]. Therefore, the images are first transformed into HSV color space are transformed into a corresponding set of Hue, Saturation, and Value values. This is done by the ***cv:cvtColor*** function[28].

The actual filtering is done with the ***cv::inRange*** function[28], which takes a lower and an upper limit for each channel of the image as its arguments. Only pixels have hue(H), saturation(S), and value(V) values satisfy the following conditions pass through the filter:

$$h\_min \leq H \leq h\_max$$

$$s\_min \leq S \leq s\_max$$

$$v\_min \leq V \leq v\_max$$

The result is a one channel image that contains the value 255 where these conditions are fulfilled and zero where they are not. An example, from the Figure 3.3 describe an image origin on the right and the result of applying the filter to it on the left.

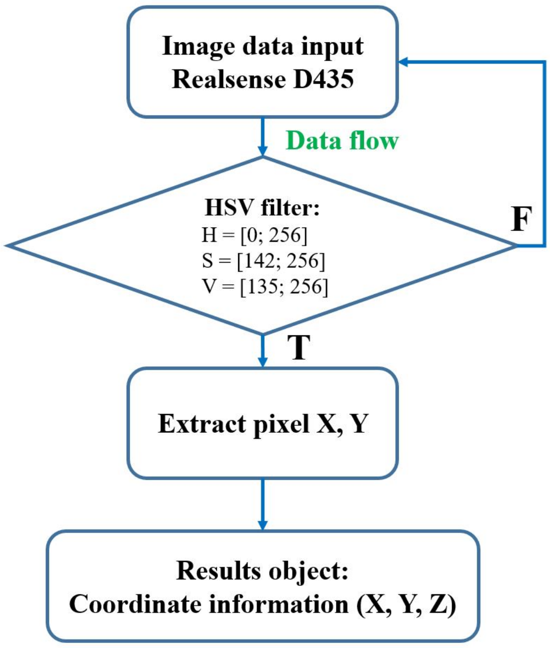


FIGURE 3.2: Object detection main algorithm.

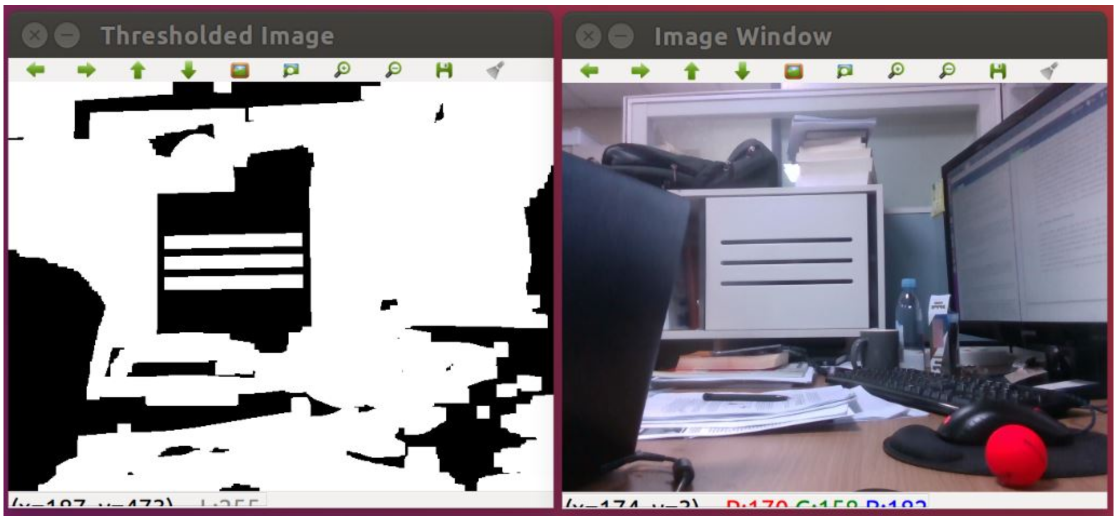
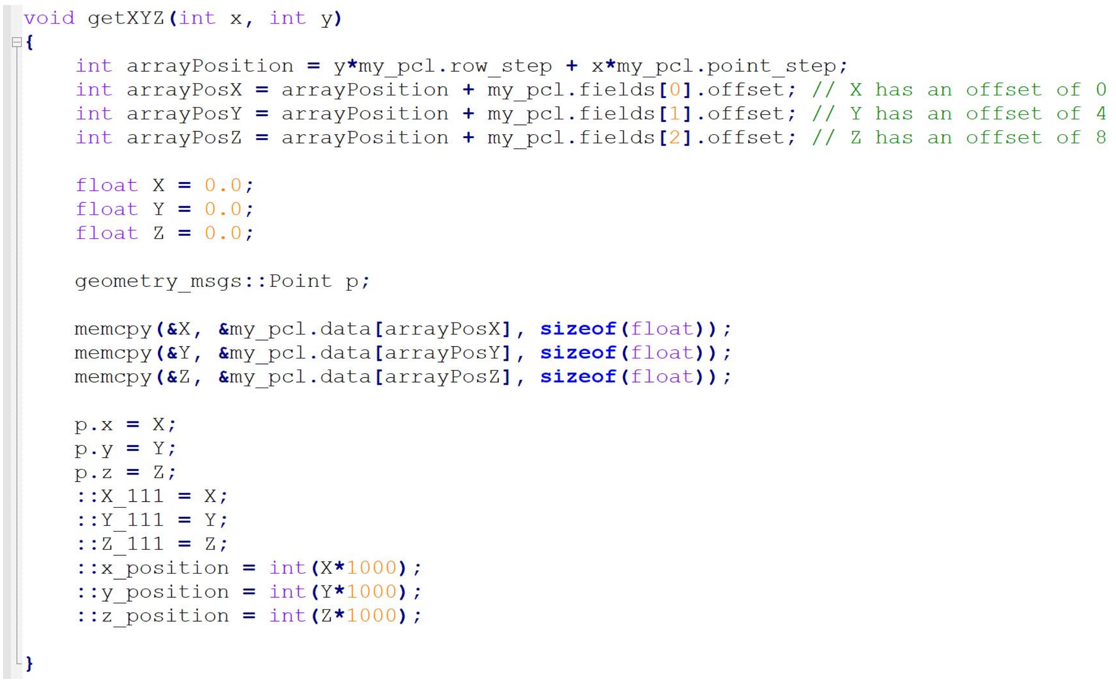


FIGURE 3.3: Image before(right) and after(left) applying the *cv::inRange* function.

Carefully inspecting the figure reveals that the result of the filter is not perfect, and adjusting the parameters reveals that it is hard to make the result much better under varying lightning conditions. To remove the white speckle noise in the image, a morphological transformation is applied to it; the ***erosion*** and ***opening*** is capable of applying these transformations are suited for this task. The *cv::morphologyEX* function[28] is capable of applying these transformations to the image. Parameters of this function, which should also be parameters of the object recognition system, are the kind of operation (erode, dilate, open, etc.).

3.1.3 Position of object extraction

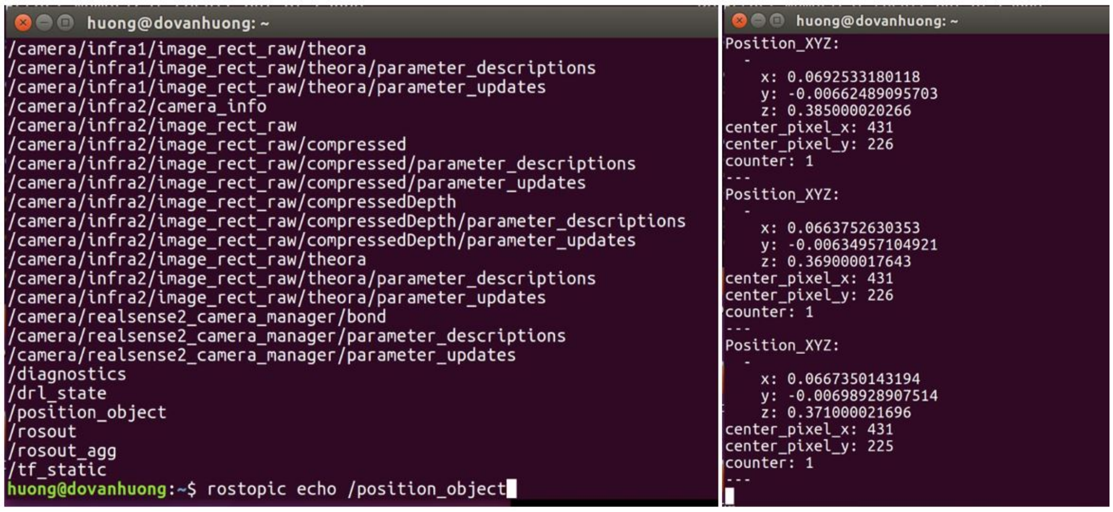
For extracting the position of the object in (X,Y,Z) coordinate, we used PCL library[29] after the object was detected from HSV algorithm. Thanks to information of the object in 2D, we found the pixel point of object in 2D image and converted that into the 3D point by the algorithm in Figure 3.4.



```
void getXYZ(int x, int y)
{
    int arrayPosition = y*my_pcl.row_step + x*my_pcl.point_step;
    int arrayPosX = arrayPosition + my_pcl.fields[0].offset; // X has an offset of 0
    int arrayPosY = arrayPosition + my_pcl.fields[1].offset; // Y has an offset of 4
    int arrayPosZ = arrayPosition + my_pcl.fields[2].offset; // Z has an offset of 8

    float X = 0.0;
    float Y = 0.0;
    float Z = 0.0;

    geometry_msgs::Point p;

    memcpy(&X, &my_pcl.data[arrayPosX], sizeof(float));
    memcpy(&Y, &my_pcl.data[arrayPosY], sizeof(float));
    memcpy(&Z, &my_pcl.data[arrayPosZ], sizeof(float));

    p.x = X;
    p.y = Y;
    p.z = Z;
    ::X_111 = X;
    ::Y_111 = Y;
    ::Z_111 = Z;
    ::x_position = int(X*1000);
    ::y_position = int(Y*1000);
    ::z_position = int(Z*1000);

}
```

FIGURE 3.4: The position 3D extract of Object algorithm.

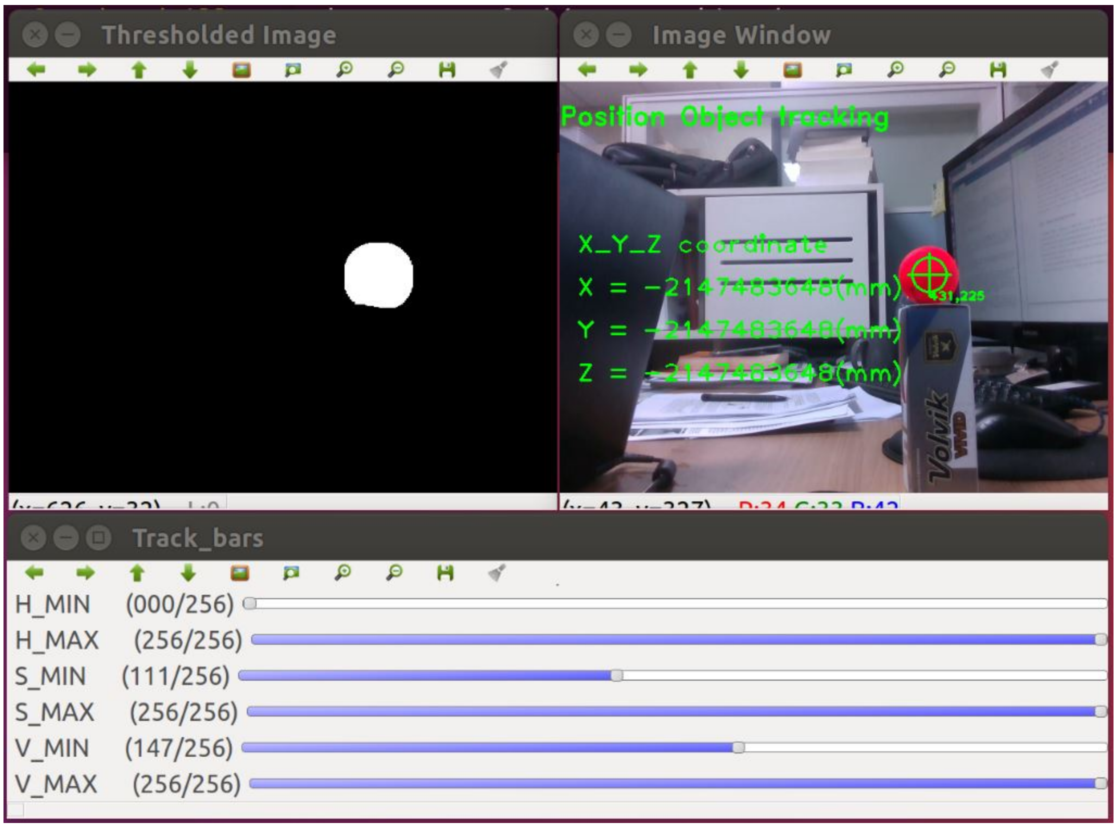
After the position of the object was found, the detail information of object with X, Y, Z coordinate will be published in a topic name ***position_object*** as in Figure 3.5.



```
huong@dovanhuong: ~
/camera/infra1/image_rect_raw/theora
/camera/infra1/image_rect_raw/theora/parameter_descriptions
/camera/infra1/image_rect_raw/theora/parameter_updates
/camera/infra2/camera_info
/camera/infra2/image_rect_raw
/camera/infra2/image_rect_raw/compressed
/camera/infra2/image_rect_raw/compressed/parameter_descriptions
/camera/infra2/image_rect_raw/compressed/parameter_updates
/camera/infra2/image_rect_raw/compressedDepth
/camera/infra2/image_rect_raw/compressedDepth/parameter_descriptions
/camera/infra2/image_rect_raw/compressedDepth/parameter_updates
/camera/infra2/image_rect_raw/theora
/camera/infra2/image_rect_raw/theora/parameter_descriptions
/camera/infra2/image_rect_raw/theora/parameter_updates
/camera/realsense2_camera_manager/bond
/camera/realsense2_camera_manager/parameter_descriptions
/camera/realsense2_camera_manager/parameter_updates
/diagnostics
/drl_state
/position_object
/rosout
/rosout_agg
/tf_static
huong@dovanhuong:~$ rostopic echo /position_object
```

```
huong@dovanhuong: ~
Position_XYZ:
  -
    x: 0.0692533180118
    y: -0.00662489095703
    z: 0.385000020266
center_pixel_x: 431
center_pixel_y: 226
counter: 1
---
Position_XYZ:
  -
    x: 0.0663752630353
    y: -0.00634957104921
    z: 0.369000017643
center_pixel_x: 431
center_pixel_y: 226
counter: 1
---
Position_XYZ:
  -
    x: 0.0667350143194
    y: -0.00698928907514
    z: 0.371000021696
center_pixel_x: 431
center_pixel_y: 225
counter: 1
---
```

FIGURE 3.5: Information of the object under topic name ***position_object***



```
Position Object tracking
X_Y_Z coordinate
X = -2147483648(mm)
Y = -2147483648(mm)
Z = -2147483648(mm)
```

```
H_MIN   (000/256)
H_MAX   (256/256)
S_MIN   (111/256)
S_MAX   (256/256)
V_MIN   (147/256)
V_MAX   (256/256)
```

FIGURE 3.6: Track bars from HSV filter for the object detection and information of the object detected

3.2 Manipulator and work-space platform

3.2.1 Universal Robot UR5

In this thesis, the industrial manipulator Universal Robot UR5[31] was used for learning task in both virtual and real environment in Figure 3.8[35]. UR5 is controlled by TCP/IP, 100BASE-TX Ethernet socket. It is programmed with poly-scope graphical user interface. UR5 has a working in a temperature range of $0 - 50°$. In the following part is the specification of UR5:

- **Weight:** 18.4kg/ 40.6lbs
- **Payload:** 5kg/ 11lbs
- **Repeatability:** $\pm 0.1mm/ \pm 0.0039in$ (4mils)
- **Footprint:** ϕ149mm /5.9in
- **Degrees of freedom:** rotating joints is 6
- **Control box size (W×H×D:)** 475×423×268(mm) /18.7×16.7×10.6(in)

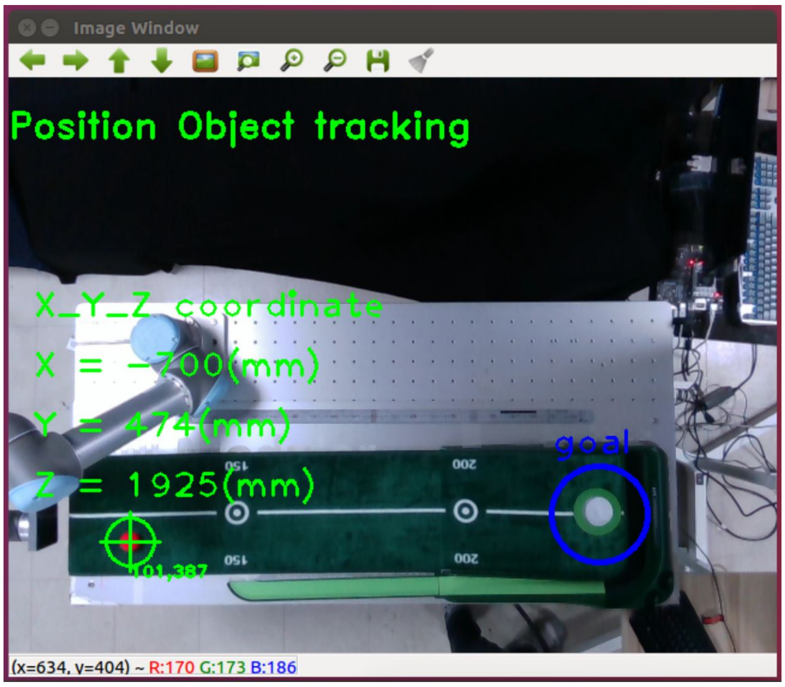


FIGURE 3.7: Output information of object tracking with Realsense D435.

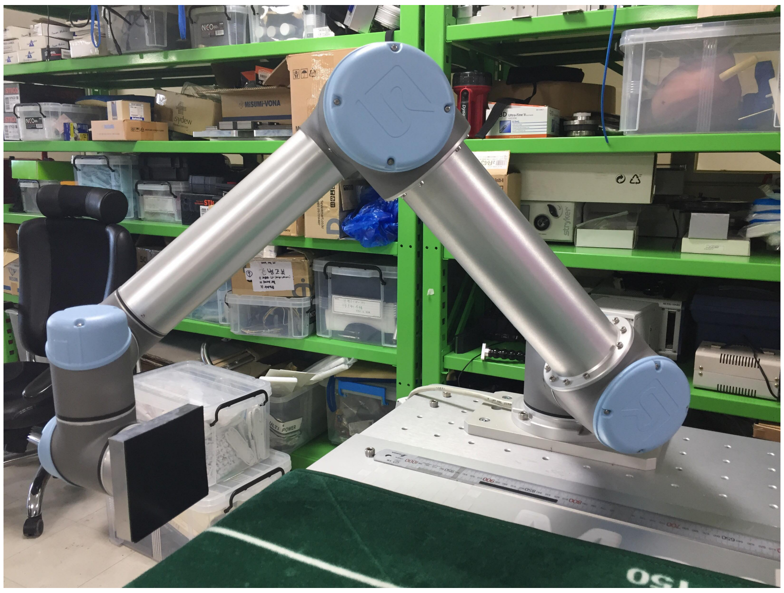


FIGURE 3.8: Universal Robot UR5.

- **I/O power supply:** 24V 2A in the control box and 12V/24V 600mA in the tool

In order to programmed UR5 we can communicate over a TCP/IP connection called URControl in the low level robot controller. That is controlled by the script-level with programming language URScript.

3.3 Golf-platform and the ball object

Base platform for putting golf-ball with dimension weight$\times$ height $\times$ length: $300 \times 1300 \times 85 (mm)$. The hardware setup environment was shown as in Figure 3.9. The ball

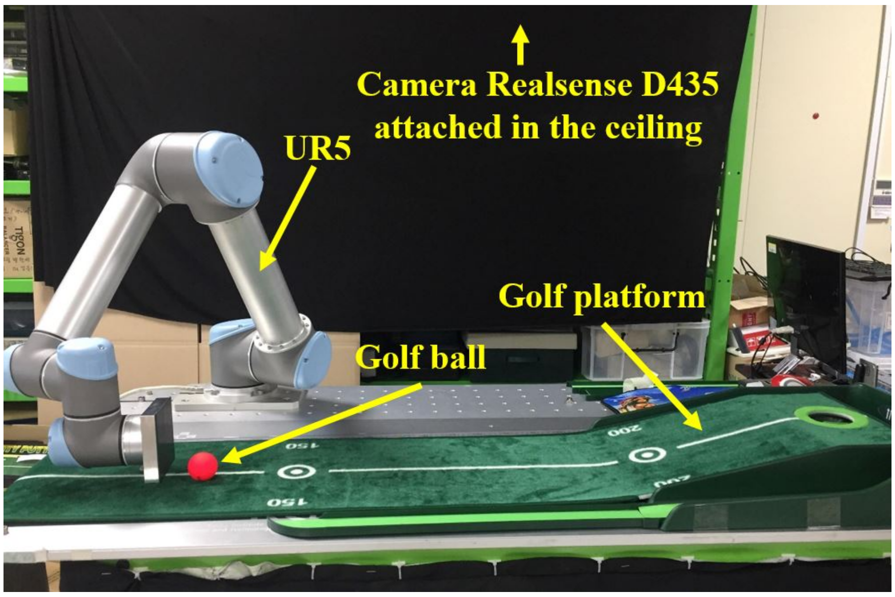


FIGURE 3.9: Hardware setup environment: *UR5 robot, golf-platform and vision based system.*

of object with a diameter is 42.67mm under the solid type object.

Chapter 4

Software libraries and Gazebo implementation

In order to use simulation to mimic the real-life of golf-putting, our system includes Gazebo simulation environment, a robot arm UR5 (Universal Robot)[31], a golf platform put in a table play as a work-space of manipulator. The target object is a golf-ball with diameter 42.67mm. All software and libraries were used in Linux LTS 16.04 version 64bit and ROS Kinetic platform system.

4.1 Robot Operating System Programming introduction

The version of ROS is Kinetic was used in this thesis. ROS is an open-source framework for developing robot software. Thanks to provide tools, conventions and libraries of ROS, we can implement and test algorithm more efficiently. With the aim to change a little in the code[14], ROS can work in other robots by adjust a little in code. Some essential elements of ROS are topics, nodes, messages and services. The core of ROS is the ROS Master because of it provide the look-up in computation graph and for registration name. Without the master, nodes or topic ROS cannot work and cannot connect each other. Besides, a message is a data structure. The topic used for identify the content of message which was sent from another topics. Thanks to service will help to define a message structure by request and reply for the other.
Furthermore, ROS provide some tools for analysis and data processing such as 3D visualization or real-time logging even playing offline with rosbag, plotting data and visualize the ROS network structure by rxgraph.

4.1.1 ROS Graph

A *ROS system* is made up of multiple simultaneously running processes, called *ROS nodes*. The ROS nodes connected together by ROS *topics*. Multiple nodes publish under the *messages*. From subscribing to the topic and other node can take information inside messages. A *ROS graph* renders these correlations for a specific ROS system. Figure 4.1 shows the ROS graph of a simple system. The two nodes *publisher1* and *publisher2* publish messages on the topic called *topic1*, to which the node *subscriber1* subscribes. The *publisher2* also publishes messages on the topic *topic2*, to which *subscriber1* and *subscriber2* subscribe.

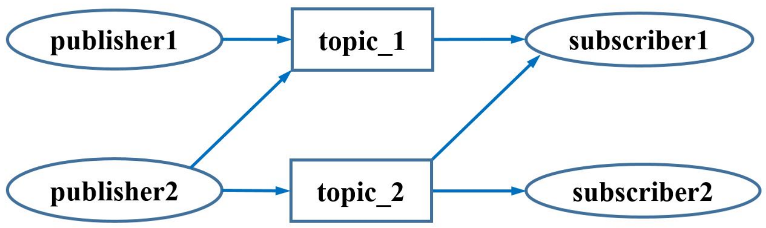


FIGURE 4.1: ROS graph of the system that comprises four nodes and two topics.

All messages that are published on and expected from a specific topic have to be of the same type. ROS provides many standard types –e.g. integers, strings, images, and point clouds. If these types are not sufficient, new types can be defined.

Another feature of ROS is the way is to remap the names of topics when launching nodes. This allows to connect nodes that use the same message type, even if, according to their source code, they publish and subscribe to differently named topics. When, in the example above, the name *topic2* is mapped to *topic3* on start-up of node *publisher2*, the ROS graph of the system looks like Figure 4.2.

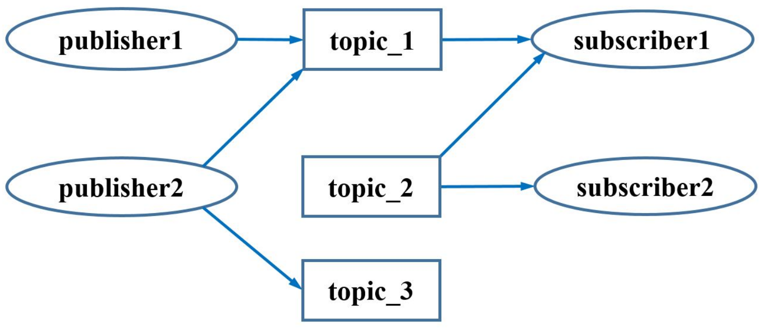


FIGURE 4.2: ROS graph of the system after mapping the name *topic2* to *topic3* on start-up of *publisher2*

4.1.2 Packages and Work-spaces

ROS organized into *packages*. In fact, package is a collection of resources –i.e. code, data, and documentation – that are built and distributed together.
New packages are created within a *work-space*. A work-space is a set of directories and contains a related set of ROS code. The complete code of this project resides in a single work-space, folders called ***ur_gazebo_test2***, ***golf_platform***, ***opencv_object_tracking***. Inside this work-space is a folder called ***src***, which contains the packages of this project, each one represented by a separate folder.

4.1.3 Facilities for Launching ROS Systems

rosrun is a command-line tool for launching single ROS nodes. The command-line arguments that are passed to the node:

$$\$\ rosrun \quad < package > \ < executable > \quad [args]$$

Separately launching every node of a large ROS system is cumbersome and error-prone. For this reason, ROS provides the ***roslaunch*** command-line tool as below:

$$\$\ roslaunch \quad < package > \ < launch - file >$$

Launch files are XML files that describes the nodes that are launched by ***roslaunch***, together with their parameters, re-mappings, and command-line arguments. Several examples of launch files are presented in this document.

4.1.4 Other Communication Mechanisms

Besides topics, ROS nodes use two main communication mechanisms.
The *ROS services* are synchronous remote procedure calls. This means that nodes can call a function that executes in another node, the service provider. This function is executed whenever the service provider receives a service request.
The *ROS actions* are asynchronous and build on top of ROS messages, i.e. implemented using topics. Actions are typically used to provide time-extended, goal-oriented behaviour. Servers and clients react to goals, feedback, results, and other incoming messages by executing corresponding callback functions. In this project, actions are used to send action control for manipulator.

4.1.5 The ROS Master

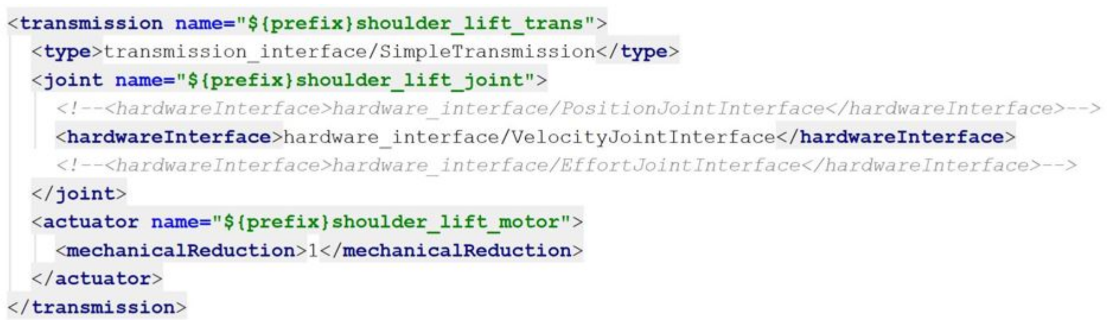
Every ROS system has a *ROS master*, which is a process with a known URI, determined by the environment variable **ROS_MASTER_URI**. Nodes register themselves with the master at start-up, and inform the master when they publish or subscribe to a topic or when they advertise or call a service.The negotiation of peer-to-peer connections are based on XML-RPC[24]. After such a peer-to-peer connection has been established, the connected nodes communicate directly with one another, normally via TCP, although other types of connections are possible.

4.1.6 Programming Languages

ROS nodes can be used in any programming language for which a so-called *client library* has been written. Client libraries for multiple languages exist, but the two best-supported APIs, which are also exclusively used in this project, are ones for C++ and Python.

4.2 Gazebo software simulation

In order to simulate the robot outdoor environment Gazebo can support us to do that. It is simulating sensors and object under three-dimensional type to reflect the real world. Gazebo supports the realistic sensor feedback and interaction between objects.
Thanks to realistically simulating the robots and the environment, robot was designed and operate on the artificial version. Nowadays, a lot of researchers are using this software to develop and run experiment in simulate environment because of the realistic manner.
In a control pipeline which is transform effort/flow variable that is a transmission. We can see the configure effort of controller transmission named ***shoulder_lift_joint*** as in Figure 4.3



```
<transmission name="${prefix}shoulder_lift_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="${prefix}shoulder_lift_joint">
    <!--<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>-->
    <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
    <!--<hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>-->
  </joint>
  <actuator name="${prefix}shoulder_lift_motor">
    <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

FIGURE 4.3: The transmission-specific code

Furthermore, we need to add a Gazebo plugin in our URDF for parsing the transmission tags which can make the appropriate hardware interfaces with the controller, from Figure 4.4 show the gazebo plugin for controller.
The robot controller is needed part for execute our manipulator work as we want. In order to do that, from YAML[18] file was loaded in configuration accordingly to the joint trajectory controller.

```
<gazebo>
  <plugin name="gazebo_joint_state_publisher" filename="libgazebo_ros_joint_state_publisher.so">
    <robotNamespace>ur5</robotNamespace>
    <updateRate>200.0</updateRate>
    <jointName>elbow_joint, shoulder_lift_joint, shoulder_pan_joint,
               wrist_1_joint, wrist_2_joint, wrist_3_joint
    </jointName>
  </plugin>
</gazebo>
```

FIGURE 4.4: Gazebo_ros_control Plugin

From the figure 4.5 we have two ways in publish commands to control of the joints. The

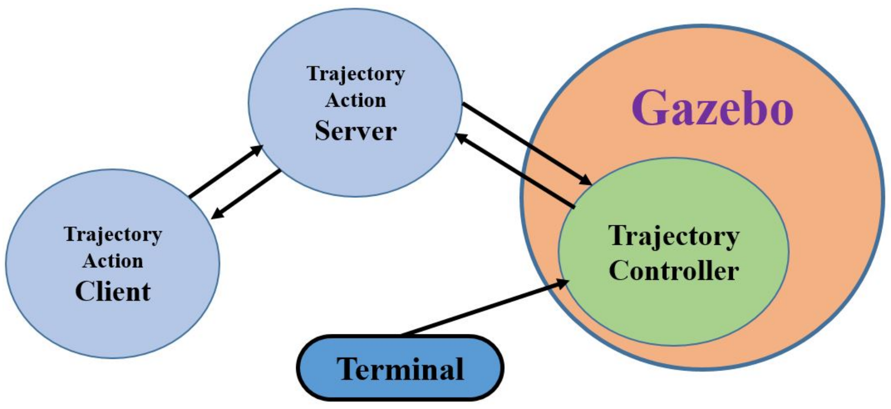


FIGURE 4.5: Methods for command the trajectory controller inside Gazebo.

first method is send command directly from the terminal as in 4.6

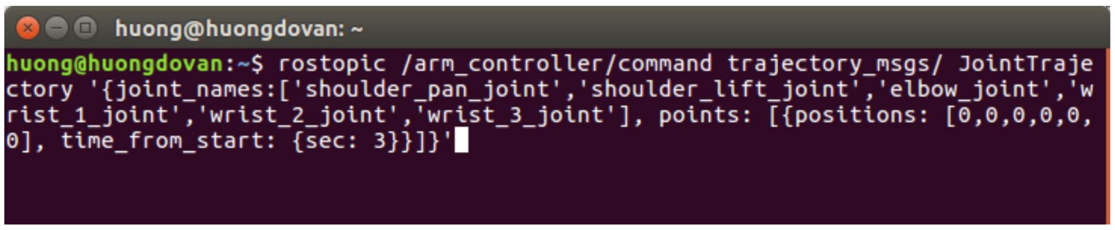

```
huong@huongdovan: ~
huong@huongdovan:~$ rostopic /arm_controller/command trajectory_msgs/ JointTraje
ctory '{joint_names:['shoulder_pan_joint','shoulder_lift_joint','elbow_joint','w
rist_1_joint','wrist_2_joint','wrist_3_joint'], points: [{positions: [0,0,0,0,0,
0], time_from_start: {sec: 3}}]}'
```

FIGURE 4.6: The content of command send directly from terminal.

The second method is by communicate between action server and action client as in figure 4.7.

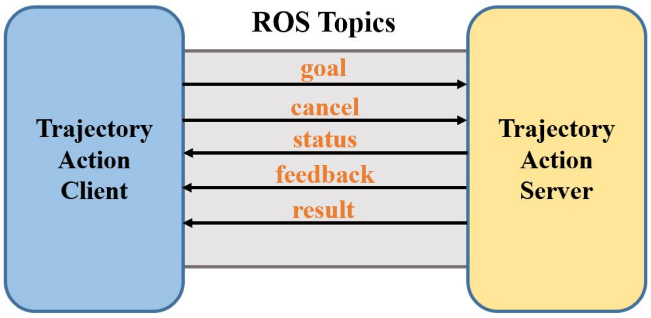


FIGURE 4.7: Communication between action server and action client by ROS Topics.

From the figure 4.7 we have some messages which clients and server communicate as below:
Goal: to take the accomplish task by using action.
Feedback: in order to tell the progress to action server from the Action client.
Result: is information or data will be sent from action server to action client for the task is complete or not.

Chapter 5

Virtual environment for Reinforcement Learning algorithms

From the definition of hardware, problem and the software libraries, in this chapter will describe how to build a virtual environment for mimicking the real environment. The detail of manipulator modeling and golf platform will be shown in this part. In addition, the method to demonstrate the consistence between the real-world and the virtual environment will be shown too.

5.1 Manipulator modeling and golf platform

In order to describe our robot, we need to have the Unified Robot Description Format (URDF)[20]. This file was created with the support from Computer Aided Design (CAD), design software as Blender, Solidworks, Pro-engineer, ...
In the figure 5.1 shows the basic structure of a robot. In the robot description model, regard to describe the link of robot will be used tag $< link >$ and tag $< /link >$. In addition, describe the joint of robot will be used tag $< joint >$ and tag $< /joint >$.
We can import model as meshes by STereoLithography (STL)[21] and Collada[22] files into URDF.

The robot UR5 environment contains all the functions to the specific UR5 robot we want to train. In this case, we implement to use a joint trajectory controller to let our UR5 robot work.

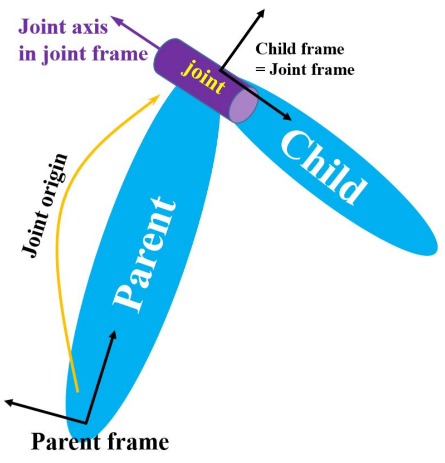


FIGURE 5.1: Connection of two links by one re-volute joint.

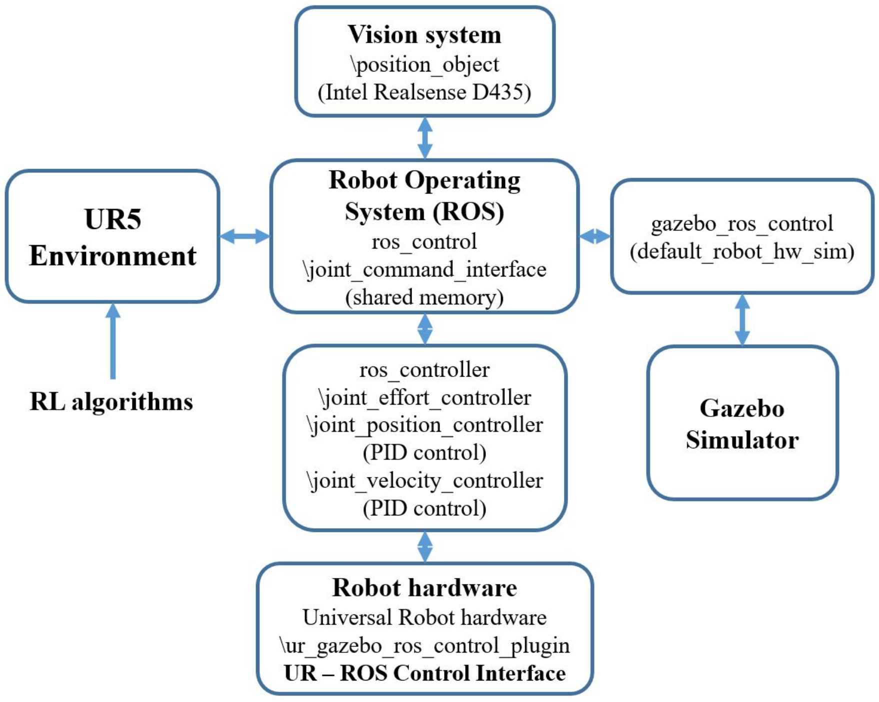


FIGURE 5.2: Block communication between simulation and hardware from Gazebo to ROS and test-bedded.

5.2 Calculation friction value between surface and the ball

In this environment, we should consider the friction value between the surface and the golf-ball. There is some research to evaluate the friction of golf-ball and surface[16]. In this thesis, I proposed a solution on how to calculate the friction value. In this case, we count the golf ball rotating in the surface an angle is α(radian), the radius of the ball is R(m) with massive m(kg) of the golf ball. We measured the distance of the object traveled until that stop with d_1, d_2 sequence. According to Figure 5.4 we can see the angle of the golf platform is θ(radian), the force analysis was shown in the same graph.

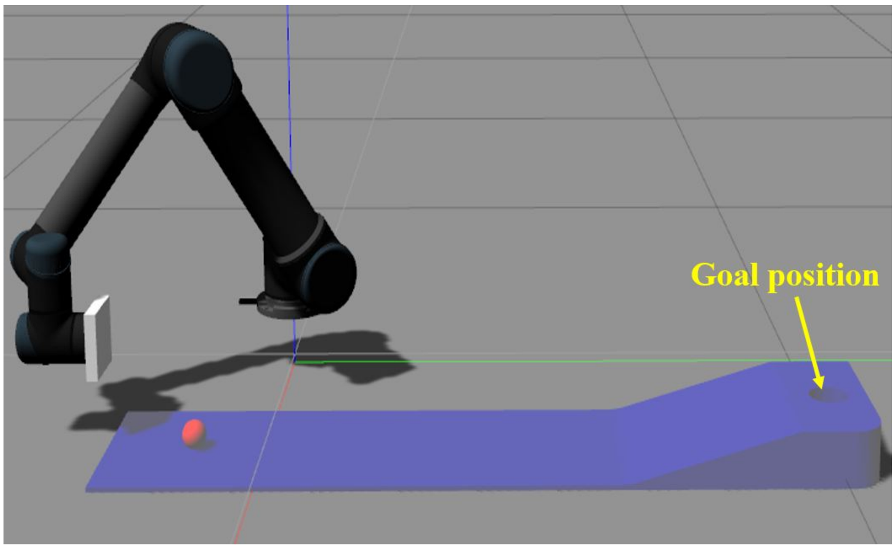


FIGURE 5.3: The virtual environment of golf-putting and UR5.

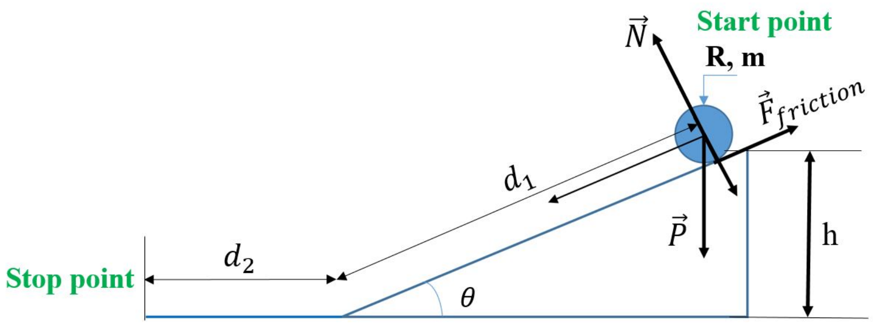


FIGURE 5.4: Friction between the golf ball and surface analysis.

From that, we will have equations following the rule of energy:

$$mgh = F_{friction_1} d_1 + F_{friction_2} d_2$$

We assume that the value of friction will be same in all surface of platform. From the force analysis in fig 5.4 we will have the energy in distance d_1, d_2 will be:

$$F_{friction_1} d_1 = mg\mu cos\theta d_1$$

$$F_{friction_2} d_2 = mg\mu d_2$$

After the calculation we will have friction value:

$$\mu = \frac{h}{d_1 cos\theta + d_2}$$

From the fig 5.6 show how we measure the distance of ball after free drop from the top of platform until that ball stopped. After take the experiment with 109 times to measure

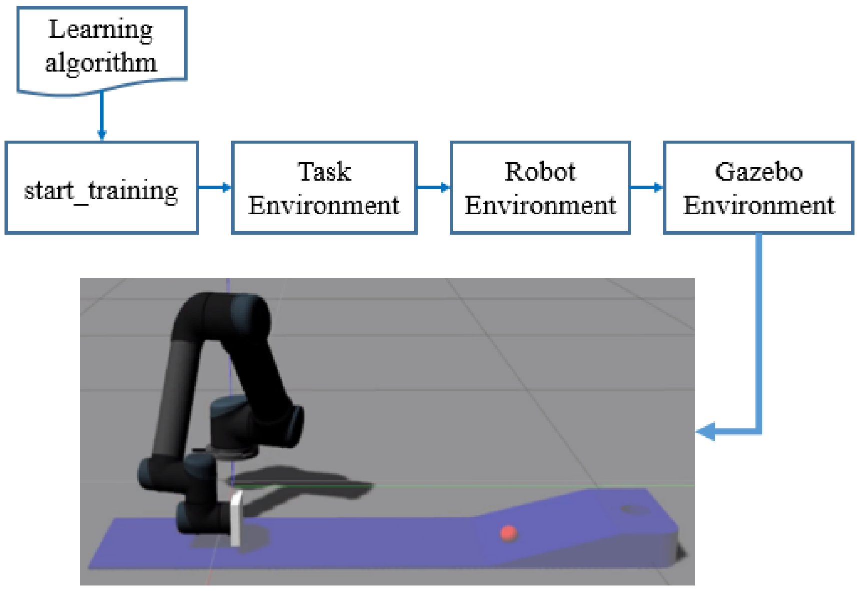


FIGURE 5.5: Reinforcement Learning algorithm training in the virtual environment

the distance d_1 and d_2 sequence, we have the mean of friction value $\mu = 0.10378$. The details of data for experiment in the appendix B of this thesis.
In order to validate the value of friction from the experiment. We input that value into simulation environment and test the same method with experiment. After that we measure the value of distance d_1 and d_2 after release the ball from the top of platform until it stops. We will make the comparison between the mean of d_2 in the real world with the value of d_2 in the virtual environment. In the fig 5.7 shows the position of the ball will be free drop in the virtual environment. After that, the fig 5.8 describes the position stopped of ball after free drop from the top of platform with the detail of coordinate in the virtual environment.
First, from the real environment, we measured the mean of distance $d_2 = 484.11(mm)$.
Second, from the virtual environment we take the value of distance $d_2 = 471.98(mm)$.

Therefore, the different between simulation and physic will be: $\Delta d_2 = 0.01214$. From that we will have the error between simulation and physic will be described as below:

$$d_2 = 0.4841 \pm 0.01214(mm)$$

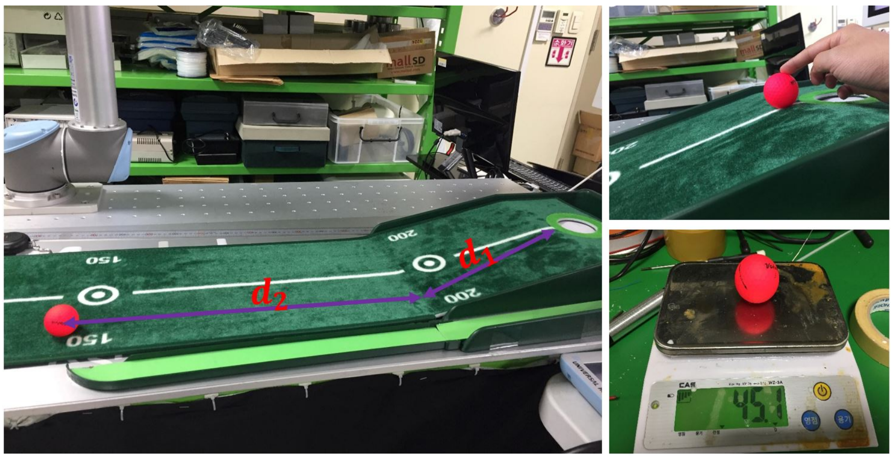
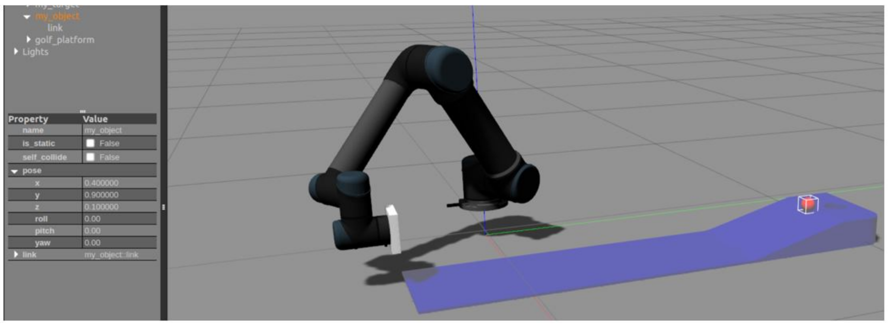
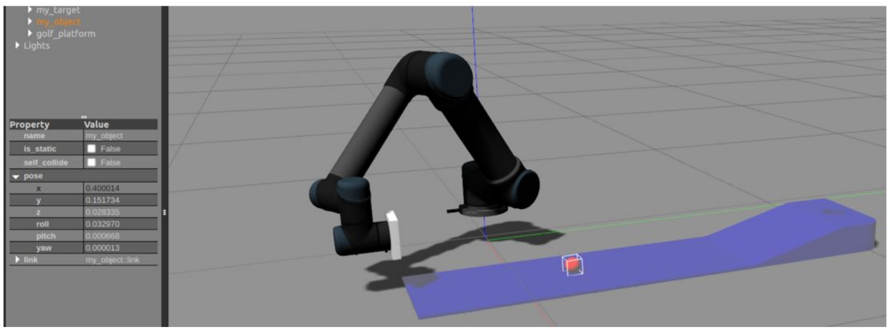


FIGURE 5.6: The real measurement from experiment with d_1, d_2 and the weight of ball



Property	Value
name	my_object
is_static	False
self_collide	False
pose	
x	0.400000
y	0.900000
z	0.100000
roll	0.00
pitch	0.00
yaw	0.00
link	my_object::link

FIGURE 5.7: Ball drop from start position in virtual environment. The coordinate of ball: (x,y,z) = (0.4, 0.9, 0.1)



Property	Value
name	my_object
is_static	False
self_collide	False
pose	
x	0.400014
y	0.151734
z	0.028335
roll	0.032970
pitch	0.000668
yaw	0.000013
link	my_object::link

FIGURE 5.8: Ball stopped position in virtual environment. The coordinate of ball: (x,y,z) = (0.4, 0.1517, 0.02)

Chapter 6

Simulation result

6.1 The results in simulation and the real hardware

6.1.1 Result from simulation in the virtual environment

From the Gazebo environment, we completed to mimicking the virtual environment from physical data. The tasks of manipulator were completed execute in Gazebo environment. The Figure 6.1 was shown the process of UR5 task with golf putting in the virtual environment.

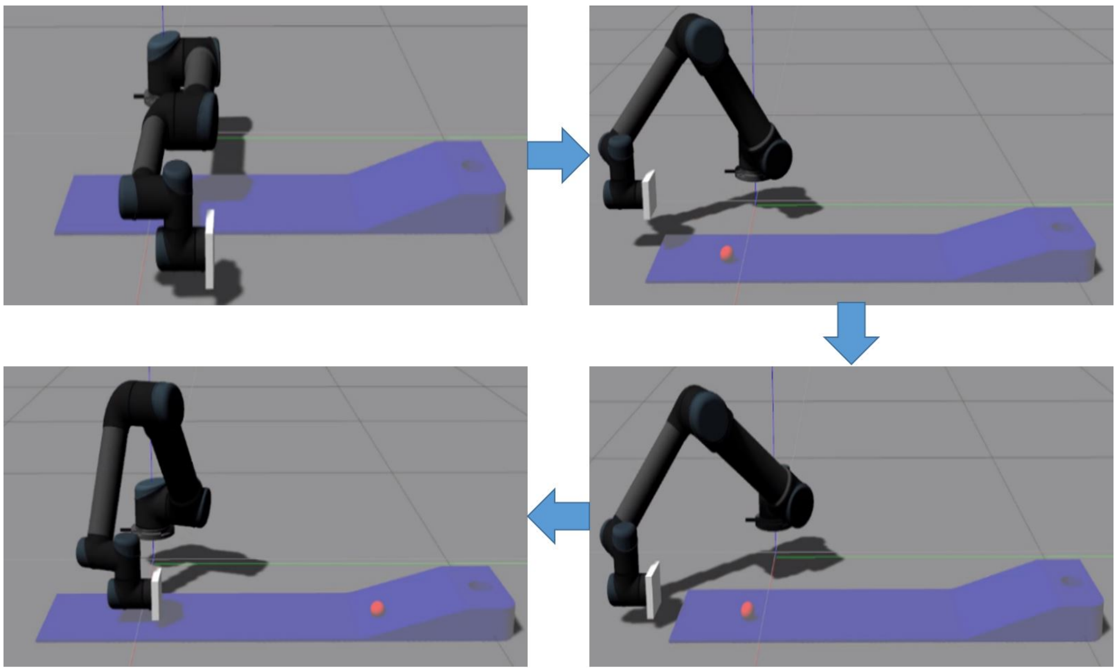


Figure 6.1: Task simulation in the virtual environment.

After implemented with Monte-Carlo algorithm the result of training was improved with the success rate for golf putting increased as in the graphs as in below.

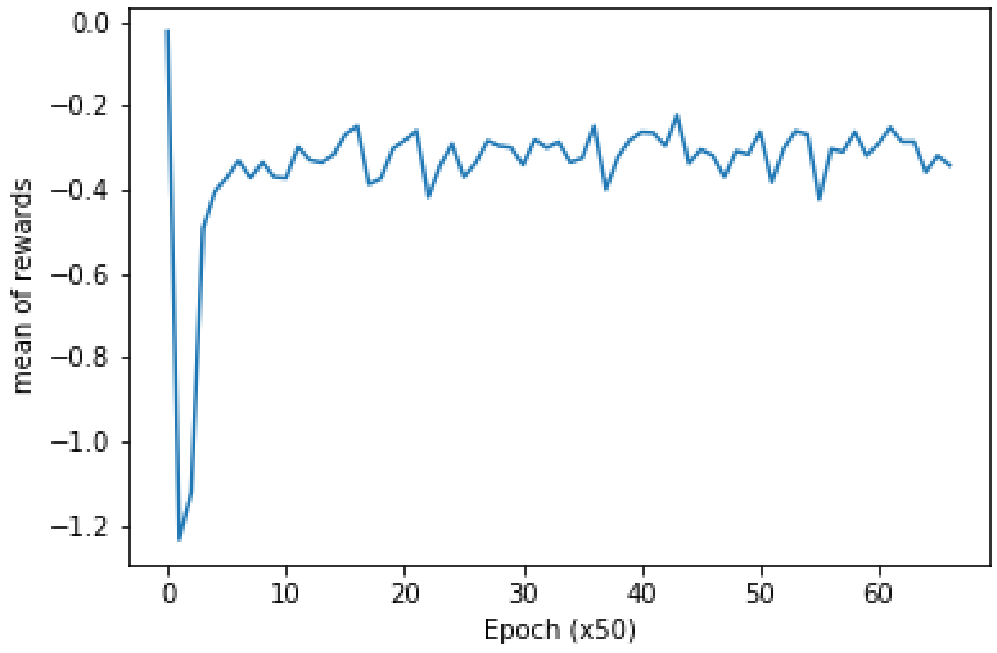


FIGURE 6.2: Normal of rewards over epoch.

After training with 14000 epochs, we had the result of rewards as in Figure 6.2 with accumulate of reward increased over time. In addition, the mean of success rate in Figure 6.3 also increase however the average is lower than 68(%).
Furthermore, in the link as below is the result from simulation: `https://youtu.be/HVbtnGaIi-s`

6.1.2 Result in test-bed with hardware setup.

From the result in the simulation and training for UR5 how to putting golf. We transfer that policy result in the real hardware and make a comparison between the simulation and the real. The Figure 3.9 was shown the detail of the real hardware setup with a vision system, a golf platform environment, and a ball. All of the hardware was setup in a table as a work-space of robot UR5.
We are working on the task of training by Monte-Carlo algorithm for UR5 and after that, we will transform the result into the real hardware setup. From the Figure 3.9 and Figure 6.1 were shown how consistence between the simulation and real hardware.
`https://youtu.be/HdxvACGRTwI`

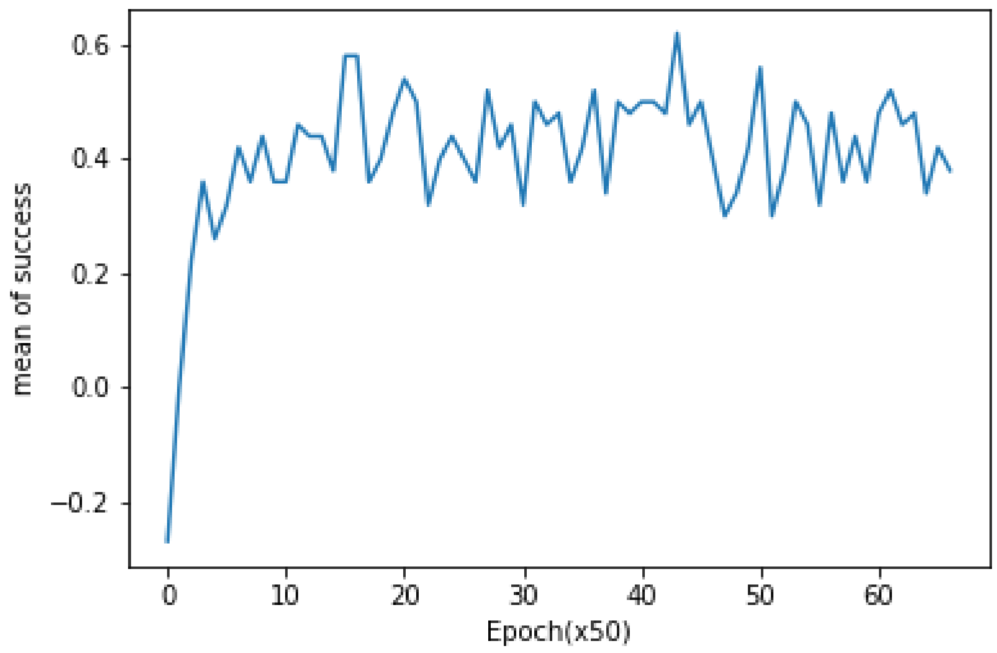


FIGURE 6.3: Mean of success rate.

6.2 Sum up from simulation result

Thanks to Gazebo simulator. It is simple to simulate our robot by tasks in daily life. Thanks to that, the time spent in robotics research will be reduced significantly.In this thesis proposed an approach helps to reduce time to train in hardware with reinforcement learning algorithm by the virtual environment and transfer policy into real hardware. However, the consistence between reality and virtual environment still exist some gap such as the impact of wind in the environment and the friction value. Because of these factors can impact a lot on the result of agent. In order to solve current issues, simulation tools should be increase accuracy and high consistence with impact factor from the reality. The current result from simulation still low, it should be improved for future works and increase the performance of algorithm for have a better result.

Chapter 7

Conclusions

In this thesis presents method to mimicking the real environment with the aiding of software and libraries. After that, we implemented the state of the art of reinforcement learning algorithm in the virtual environment to training agent how to reach the good results.
In chapter 1 and 2, we provided the introduction of reinforcement learning algorithm. After that we determine our problem formulation under reinforcement learning problem and mathematics accordingly. The Monte-carlo algorithm was used in this problem to training agent in the virtual environment.

7.1 Hardware and test-bed

In chapter 3, we determined our target hardware structure will be implemented with the combination of manipulator UR5, golf platform and the vision system. In each hardware part plays unique roles to determines in the whole system: UR5 will generate actions from the policies input from the virtual environment, golf platform provide environment task for UR5 and the vision system will determine how success of UR5 by send the feedback signal of object position as a reward function from reinforcement learning algorithm.
The detail of algorithms and control implementation with the hardware system was shown in chapter 3.

7.2 Software libraries, Gazebo and the virtual environment

In chapter 4, we shown the software and libraries were used in this thesis to mimicking the hardware system. With the support of ROS programming framework and Gazebo libraries we completed to mimicking the real-hardware in the virtual environment.
From chapter 5, we generated UR5 and golf-platform model in the virtual environment. In addition, to demonstrate the consistence between the virtual environment and the real-hardware, we calculated the friction value between golf-ball and the surface of platform as in detail from this chapter.
Following that is the reinforcement learning algorithm was used in the virtual environment to generate the best policies. The detail of result under the Monte-carlo algorithm was shown in chapter 6. After that in the real-hardware will receive the best policies from the virtual environment and the result of test-bed was shown in this chapter too.
In overall, this thesis was success to mimicking the hard-ware environment by support of software and libraries to build the virtual environment. Moreover, the implementation of reinforcement learning algorithm was used in the virtual environment for agent generate the best policies before transfer to the real hardware. Some results was shown in this thesis.
However, with the current of simulation about 66% of success rate over 3000 episodes of training still in the low performance. In opposite, by dint of apply the virtual environment we can utilize time in training from hardware. Thanks to simulate reinforcement learning algorithms in the virtual environment do we save a bunch of time in real hardware of training.
The future work of this thesis will be focus on how to implement the better algorithms for reaching the higher success rate of training.

Appendix A

Source code for execute algorithm and connection

A.1 Source code of thesis project

https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019

A.2 Virtual environment for Reinforcement Learning tasks

Universal Robot UR5 description
https://github.com/ros-industrial/universal_robot
Virtual Golf platform description
https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019/tree/master/golf_platform
Virtual golf ball object
https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019/blob/master/ur_gazebo_test2/urdf/golf_ball_object.sdf

A.3 Virtual environment with python

UR5 virtual environment with python definition
https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019/blob/master/ur_gazebo_test2/scripts/ur5_env.py

Golf putting task in virtual environment
https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019/blob/master/ur_gazebo_test2/scripts/slide_puck.py

A.4 Reinforcement learning algorithm implementation

DDPG algorithm
https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019/blob/master/ur_gazebo_test2/scripts/ddpg_ur5_20190416.py
The Monte-Carlo policy gradient algorithm
https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019/blob/master/ur_gazebo_test2/scripts/q_learning_20190503.py

A.5 Vision system code with Realsense D435

https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019/blob/master/opencv_object_tracking/src/object_filter.cpp

A.6 Hardware connection from virtual to real environment

https://github.com/dovanhuong/master_thesis_hci_robotics_june_2019/blob/master/modman_comm/src/ur_comm.cpp

Appendix B

Data from experiment measurement for friction

No.	Experiment with platform height = 80 mm					
	Distance d_1 (mm)	Distance d_2 (mm)	Weight of golf ball (gram)	Theta (degree)	Height (mm)	muy (µ)
1	300	475	45.1	15.46601	80	0.104693
2	300	460	45.1	15.46601	80	0.10679
3	300	480	45.1	15.46601	80	0.104013
4	300	525	45.1	15.46601	80	0.098264
5	300	480	45.1	15.46601	80	0.104013
6	300	510	45.1	15.46601	80	0.100108
7	300	510	45.1	15.46601	80	0.100108
8	300	550	45.1	15.46601	80	0.095336
9	300	555	45.1	15.46601	80	0.094771
10	300	500	45.1	15.46601	80	0.101377
11	300	510	45.1	15.46601	80	0.100108
12	300	560	45.1	15.46601	80	0.094213
13	300	520	45.1	15.46601	80	0.098871
14	300	520	45.1	15.46601	80	0.098871
15	300	533	45.1	15.46601	80	0.097307
16	300	522	45.1	15.46601	80	0.098627
17	300	525	45.1	15.46601	80	0.098264
18	300	540	45.1	15.46601	80	0.096486
19	300	520	45.1	15.46601	80	0.098871
20	300	542	45.1	15.46601	80	0.096254
21	300	535	45.1	15.46601	80	0.097071
22	300	542	45.1	15.46601	80	0.096254
23	300	540	45.1	15.46601	80	0.096486
24	300	555	45.1	15.46601	80	0.094771
25	300	540	45.1	15.46601	80	0.096486
26	300	560	45.1	15.46601	80	0.094213
27	300	510	45.1	15.46601	80	0.100108
28	300	442	45.1	15.46601	80	0.109419
29	300	460	45.1	15.46601	80	0.10679
30	300	460	45.1	15.46601	80	0.10679
31	300	455	45.1	15.46601	80	0.107507
32	300	475	45.1	15.46601	80	0.104693
33	300	450	45.1	15.46601	80	0.108234
34	300	500	45.1	15.46601	80	0.101377
35	300	450	45.1	15.46601	80	0.108234
36	300	545	45.1	15.46601	80	0.095908
37	300	525	45.1	15.46601	80	0.098264
38	300	465	45.1	15.46601	80	0.106082
39	300	500	45.1	15.46601	80	0.101377
40	300	382	45.1	15.46601	80	0.119201
41	300	510	45.1	15.46601	80	0.100108
42	300	545	45.1	15.46601	80	0.095908

43	300	475	45.1	15.46601	80	0.104693
44	300	500	45.1	15.46601	80	0.101377
45	300	523	45.1	15.46601	80	0.098506
46	300	525	45.1	15.46601	80	0.098264
47	300	475	45.1	15.46601	80	0.104693
48	300	415	45.1	15.46601	80	0.113614
49	300	475	45.1	15.46601	80	0.104693
50	300	440	45.1	15.46601	80	0.109719
51	300	520	45.1	15.46601	80	0.098871
52	300	485	45.1	15.46601	80	0.103341
53	300	510	45.1	15.46601	80	0.100108
54	300	475	45.1	15.46601	80	0.104693
55	300	400	45.1	15.46601	80	0.116087
56	300	410	45.1	15.46601	80	0.114427
57	300	460	45.1	15.46601	80	0.10679
58	300	445	45.1	15.46601	80	0.108972
59	300	460	45.1	15.46601	80	0.10679
60	300	492	45.1	15.46601	80	0.102415
61	300	400	45.1	15.46601	80	0.116087
62	300	480	45.1	15.46601	80	0.104013
63	300	500	45.1	15.46601	80	0.101377
64	300	484	45.1	15.46601	80	0.103475
65	300	485	45.1	15.46601	80	0.103341
66	300	522	45.1	15.46601	80	0.098627
67	300	505	45.1	15.46601	80	0.100738
68	300	380	45.1	15.46601	80	0.119557
69	300	475	45.1	15.46601	80	0.104693
70	300	480	45.1	15.46601	80	0.104013
71	300	455	45.1	15.46601	80	0.107507
72	300	510	45.1	15.46601	80	0.100108
73	300	416	45.1	15.46601	80	0.113453
74	300	490	45.1	15.46601	80	0.102678
75	300	429	45.1	15.46601	80	0.111399
76	300	445	45.1	15.46601	80	0.108972
77	300	435	45.1	15.46601	80	0.110476
78	300	445	45.1	15.46601	80	0.108972
79	300	488	45.1	15.46601	80	0.102942
80	300	445	45.1	15.46601	80	0.108972
81	300	480	45.1	15.46601	80	0.104013
82	300	486	45.1	15.46601	80	0.103208
83	300	502	45.1	15.46601	80	0.10112
84	300	489	45.1	15.46601	80	0.10281
85	300	480	45.1	15.46601	80	0.104013
86	300	486	45.1	15.46601	80	0.103208
87	300	502	45.1	15.46601	80	0.10112
88	300	489	45.1	15.46601	80	0.10281
89	300	480	45.1	15.46601	80	0.104013

90	300	418	45.1	15.46601	80	0.113132
91	300	482	45.1	15.46601	80	0.103743
92	300	442	45.1	15.46601	80	0.109419
93	300	455	45.1	15.46601	80	0.107507
94	300	428	45.1	15.46601	80	0.111555
95	300	441	45.1	15.46601	80	0.109569
96	300	446	45.1	15.46601	80	0.108823
97	300	479	45.1	15.46601	80	0.104148
98	300	450	45.1	15.46601	80	0.108234
99	300	490	45.1	15.46601	80	0.102678
100	300	423	45.1	15.46601	80	0.112338
101	300	500	45.1	15.46601	80	0.101377
102	300	490	45.1	15.46601	80	0.102678
103	300	487	45.1	15.46601	80	0.103075
104	300	501	45.1	15.46601	80	0.101248
105	300	488	45.1	15.46601	80	0.102942
106	300	459	45.1	15.46601	80	0.106932
107	300	463	45.1	15.46601	80	0.106364
108	300	504	45.1	15.46601	80	0.100865
109	300	492	45.1	15.46601	80	0.102415
	Mean of d_2	**484.119266**		**Mean of friction**		**0.103738**

Bibliography

[1] *"Introductory Survey to Open-Source Mobile Robot Simulation Software"*. Pizarro, Arredondo, and Torriti. IEEE, 2010. 2010 Latin American. Robotics Symposium and Intelligent robotic meeting.

[2] *"Manipulation Task Simulation using ROS and Gazebo"*. Qian, Xia, Xiong, Gan, Guo, Weng, Deng, Hu and Zhang,2014 Bali Indonesia. International Conference on Robotics and Biometics, IEEE, 2014.

[3] *"ROS: an open-source Robot Operating System,"*.ICRA Workshop on Open Source Software, 2009 by authors Quigley, Gerkey, Conley,Faust, Foote, Leibs, Berger, Wheeler and A. Ng.

[4] *"Design and Use Paradigms for Gazebo, An Open-source Multi-Robot Simulator,"*.at International Conference on Intelligent Robots and Systems, Sendal, Japan, 2004. by authors Koenig, Howard.

[5] *"Human-level control through deep reinforcement learning,"*.Nature, 2015. by authors Mnih, Kavukcuoglu, Silver, Rusu, Veness, Bellemare, Graves, Riedmiller, Fidjeland, Ostrovski.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., *"Mastering the game of go with deep neural networks and tree search",* Nature, 2016.

[7] S.Levine, C. Finn, T. Darrell, and P. Abbeel, *"End-to-end training of deep visuomotor policies,"* J. Mach. Learn. Res., 2016.

[8] N.Heess, S.Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z.Wang, A. Eslami, M. Riedmiller, et al., *"Emergence of locomotion behaviours in rich envionments,"* arXiv, 2017.

[9] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W.Zaremba, *"Transfer from simulation to real world through learning deep inverse dynamics model."* arXiv, 2016.

[10] J.Tobin, R. Fong, A. Ray, J.Schneider, W.Zaremba, and P.Abbeel, *“Domain randomization for transferring deep neural networks from simulation to the real world,”* in Proc. IEEE-RSJ Int. Conf. Intell. Robot. Syst., 2017.

[11] *“DARPA awards simulation software contract to open source robotics foundation,”* from source https://spectrum.ieee.org/automaton/robotics/robotics-software/darpa-robotics-challenge-simulation-software-open-source-robotics-foundation

[12] *“Hindsight experience replay,*,in Proc. NIPS, 2017. By authors Andrychowicz, Wolski, Ray, Schneider, Fong, Welinder, McGrew, Tobin, Abbeel, Zaremba.

[13] *“Continuous control with deep reinforcement learning,*.in Proc. ICLR, 2016. By authors Lillicrap, Hunt, ritzel, Heess, Erez, Tassa, Silver and Wierstra.

[14] “*ROS: Robot Operating System,*”http://www.ros.org. Accessed 29th May,2019. by W. Garage, 2011.

[15] *“Towards semantic robot description languages.”*2011 IEEE International Conference on. By authors Kunze, Roehm, Beetz.

[16] *“To evaluate the relative influence of coefficient of friction on the motion of a golf ball (speed and roll) during a golf putt.”*.Advances in Intelligent Systems and Computing, 2016. By authors Dr Griffiths, Mckenzie, Stredwick, Hurrion.

[17] *“Massively multi-robot simulation in stage.”* by authors Vaughan, Swarm. Intelligence 2.2-4 (2008).

[18] V.Sinha, F.Doucet, C.Siska, R. Gupta, S. Liao, and A. Ghosh, *“YAML: a tool for hardware design visualization and capture.”* Proceedings of the 13th international symposium on System synthesis. IEEE Computer Society, 2000.

[19] *“The SMACH high-level executive [ROS news].”*. By authors Bohren, Cousins, Robotics and Automation Magazine, IEEE 17.4 (2010).

[20] *“Towards semantic robot description languages.”*. By authors Kunze, Roehm, Beetz, in Robotics and Automation (ICRA), 2011 IEEE.

[21] *“Data reduction methods for reverse engineering.”*. By authors Lee, Woo, and Suk, in The International Journal of Advanced Manufacturing Technology. 17.10 (2001).

[22] *“COLLADA-based File Format Supporting Various Attributes of Realistic Objects for VR Applications.”*. By authors Miyahara, Katsunori, and Okada, in Complex, Intelligent and Software Intensive Systems, 2009.

[23] *"tf: The transform library."* By authors Foote,at Technologies for Practical Robot Applications, 2013 IEEE International Conference.

[24] *"Technical Overview."*. From the Open Source Robotics Foundation, Inc. `www.wiki.ros.org/ROS/Technical_Overview.` Accessed by: 26th May 2019.

[25] OpenCV team. *"OpenCV"*. `https://opencv.org`. Accessed: 26th May 2019.

[26] Open Perception, Inc. *"PCL"*.`www.pointclouds.org.` Accessed: 26th May 2019.

[27] *"Programming Robots with ROS."*.Sebastopol: O'Reilly, 2015. By authors Quigley, Morgan, Gerkey, Brian, Smart, Willian.

[28] Kaehler, Adrian; Bradski, Gary. *"Learning OpenCV 3"*. Sebastopol: O'Reilly, 2017.

[29] *"3D is here: Point Cloud Library (PCL)."*. By authors Radu Bogdan Rusu, Steve Counsins. In proceedings IEEE International Conference on Robotics and Automation. May, 2011. Shanghai, China.

[30] `https://en.wikipedia.org/wiki/MIPI_Alliance`.

[31] `https://www.universal-robots.com/media/50588/ur5_en.pdf`.

[32] Authors Richard S. Suton, Andrew G.Barto. *"An introduction to Reinforcement Learning"*. MIT Press, Cambridge, MA, 2018.

[33] *"On the Convergence of Optimistic Policy Iteration"*.. By authors John Tsitsiklis. At Journal of Machine Learning Research 3 (2002). Massachusetts Institute of Technology, USA.

[34] `https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d435.html` Accessed 29th May,2019.

[35] `https://www.universal-robots.com/products/ur5-robot/`

[36] Authors Morgan, Claypool Publishers, *Csaba Szepesvári "Algorithms for Reinforcement Learning"*, June 9, 2009.

[37] `https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html`.

[38] Lillicrap, Jonathan Hunt, Pritzel, Heess, Erez, Tassa, Silver, Wierstra. *"Continuous control with deep reinforcement learning"*, Google Deepmind, London, UK. Published as a conference paper at UCLR 2016.