


國立臺灣師範大學科技與工程學院電機工程學系

碩士論文

Department of Electrical Engineering  
College of Technology and Engineering  
National Taiwan Normal University  
Master's Thesis

基於深度強化學習之移動大型重物  
Moving Large Size and Heavy Object with  
Deep Reinforcement Learning



許哲菡

Hanjaya Mandala

指導教授：包傑奇 教授

Advisor: Prof. Jacky Baltes

中華民國 109 年 6 月

June 2020

## Acknowledgment

This work was financially supported by the ‘Chinese Language and Technology Center’ of National Taiwan Normal University (NTNU) from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan, and Ministry of Science and Technology, Taiwan, under Grant Nos. MOST 108-2634-F-003-002, MOST 108-2634-F-003-003, and MOST 108-2634-F-003-004 (administered through Pervasive Artificial Intelligence Research (PAIR) Labs) as well as MOST 107-2811-E-003-503. We are grateful to the National Center for High-performance Computing for computer time and facilities to conduct this research.



# Moving Large Size and Heavy Object with Deep Reinforcement Learning

Student: Hanjaya Mandala

Advisor: Prof. Jacky Baltes

Department of Electrical Engineering  
National Taiwan Normal University

## ABSTRACT

Humanoid robots are designed and expected to work alongside a human. In our daily life, Moving Large Size and Heavy Objects (MLHO) can be considered as a problem that is a common activity and dangerous to humans. In this thesis, we propose a novel hierarchical learning-based algorithm, which we use dragging to transport an object on an adult-sized humanoid robot. The proposed method proves robustness on a THORMANG-Wolf adult-sized humanoid robot, that manages to drag a massive object with a mass of double of its weight (84.6 kg) for 2 meters. Therefore, the algorithms consist of three hierarchical deep learning-based algorithms to solve the MLHO problem and distributed in terms of robot vision and behavior control. Based on this insight, in the robot vision control, first, we propose deep learning algorithms to 3D object classification and surface detection.

For 3D object classification, we propose a Three-layers Convolution Volumetric Network (TCVN). Input data of the TCVN model used a voxel grid representation from point clouds data acquired from the robot's LiDAR scanner. On the other hand, for surface detection, we propose a lightweight real-time instance segmentation called Tiny-YOLACT (You Only Look at Coefficients) to segment the floor from the robot's camera. Tiny-YOLACT model is adopted from the YOLACT model and utilized ResNet-18

model as the backbone network. Furthermore, for robot behavior control, as the main part of this thesis we address solving MLHO problem by an adult-sized humanoid robot using the deep reinforcement learning algorithm for the first time. At this part, we proposed a Deep Q-Learning algorithm to train a deep model for control policy in offsetting the Centre of Body (CoB) of the robot when dragging different objects named (DQL-COB). For this purpose, the offset CoB is implemented to keep tracking with the robot's center of mass. As a result, the robot can keep balance with maintaining the ZMP in the support polygon. DQL-COB algorithm was first trained on the ROS Gazebo simulator to avoid costly experiments in terms of time and real environment constraints, then it was adopted with a real robot on three different types of surfaces.

To evaluate the stability of the THORMANG-Wolf robot with the proposed methods, we evaluated two types of experiments on three types of surfaces with eight different objects. In these experiments, in one scenario we use IMU along with foot Pressure (F/T) sensor, in the second scenario we just use IMU data as learning algorithm input. In the experiments, the success rates of applying the DQL-COB algorithm on the real robot are 92.91% with using the F/T sensor and 83.75% without using F/T sensors. Moreover, the TCVN model on 3D object classifications achieved a 90% accuracy in real-time. Correspondingly, the Tiny-YOLACT model achieved a 34.16 mAP on validation data with an average of 29.56 fps on a single NVIDIA GTX-1060 GPU.

**Keywords:** humanoid robot, deep reinforcement learning, dragging object, deep learning.

## Table of Contents

Acknowledgment.....	i
ABSTRACT .....	ii
Table of Contents.....	iv
List of Figures.....	vi
List of Tables .....	viii
Chapter 1: Introduction.....	1
1.1. Background.....	1
1.2. Problem statement.....	2
1.3. The objective of the study.....	4
1.4. Limitation of the study.....	5
Chapter 2: Literature Review .....	6
2.1. Related work .....	6
2.1.1. Pushing object.....	6
2.1.2. Pivoting object .....	8
2.1.3. Teleoperation manipulation .....	9
2.1.4. Walking Balance (Learning-Based).....	9
2.1.5. Push Recovery (Learning-Based) .....	10
2.1.6. Summary of related work.....	11
2.2. Inverse Kinematic .....	12
2.3. Walking Gait.....	14
2.4. Neural Network.....	16
2.5. Deep Learning.....	17
2.6. Object Detection .....	19
2.7. Reinforcement Learning .....	20
2.8. Deep Reinforcement Learning.....	22
Chapter 3: Methodology.....	23
3.1. THORMANG-Wolf Robot .....	23
3.1.1. Hardware Description .....	23
3.1.2. Software Description.....	26

3.1.3. The Proposed Algorithm Design.....	27
3.2. Robot Vision Process .....	30
3.2.1. 3D Object Detection (Deep Learning) .....	30
3.2.2. Floor Detection (Deep Learning) .....	35
3.3. Robot Motion Control.....	38
3.3.1. Object Grasping .....	38
3.3.2. Walking Control .....	39
3.4. Robot Behavior Control .....	42
3.4.1. DQL-COB Algorithm Design .....	43
Chapter 4: Experimental Result.....	54
4.1. Experimental Setup .....	55
4.2. Experimental Result for Robot Vision.....	57
4.2.1. 3D Object Classification Result .....	57
4.2.2. Floor Detection Result .....	59
4.3. Experimental Results for Robot Behavior .....	62
4.3.1. DQL-COB Training Results .....	62
4.3.2. DQL-COB Empirical Evaluation Result.....	66
Chapter 5: Closing .....	73
5.1. Conclusion .....	73
5.2. Future Work .....	74
Bibliographies.....	75
Autobiography .....	79
Academic Achievement.....	80

## List of Figures

Figure 1-1 Comparison motion pose on the moving object. ....	3
Figure 2-1 Example of inverse kinematic on the left leg of a biped robot;.....	12
Figure 2-2 Tree structure of the humanoid links connection [3].....	14
Figure 2-3 Sagittal plane view of walking gait cycle [38]. ....	15
Figure 2-4 ZMP support polygon [3]. ....	15
Figure 2-5 Projection of the Centre of Mass on Zero Moment Point. ....	16
Figure 2-6 Neural network architecture. ....	16
Figure 2-7 Operations done by neurons on a single layer perceptron.....	17
Figure 2-8 Deeper network architecture of ANN or called Deep Learning.....	18
Figure 2-9 Convolutional Neural Network subclass of deep learning. ....	18
Figure 2-10 Various types of 2D image object detection.....	19
Figure 2-11 Types of 3D point cloud object detection by [47]. ....	20
Figure 2-12 Markov Decision Process of Reinforcement Learning.....	20
Figure 2-13 Deep Q-Network architecture [50]. ....	22
Figure 3-1 THORMANG3 adult-sized humanoid robot. ....	24
Figure 3-2 THORMANG-Wolf hardware architecture.....	25
Figure 3-3 THORMANG-Wolf electrical components system. ....	26
Figure 3-4 ROS graph architecture performing a dragging task. ....	27
Figure 3-5 THORMANG-Wolf hierarchical framework data flow diagram. ....	28
Figure 3-6 Block diagram of the proposed DL algorithms to solve MLHO problem. .	29
Figure 3-7 Flowchart DL algorithm of 3D object classification. ....	31
Figure 3-8 Example process of preprocessing 3D point cloud data.....	33
Figure 3-9 The network architecture of the TCVN model.....	34
Figure 3-10 Types of floors used in this experiment.....	35
Figure 3-11 YOLACT network architecture [46]. ....	36
Figure 3-12 Building block (residual function) of ResNet [56].....	37
Figure 3-13 Sample pre-recorded motion for grasping different types of objects. ....	38
Figure 3-14 Cart-table model .....	40
Figure 3-15 Walking pattern generation based on preview control .....	40



Figure 3-16 Walking gait pattern generation process. ....	41
Figure 3-17 Integration walking module with the DQL-COB algorithm. ....	42
Figure 3-18 IMU sensor as a state of the environment. ....	44
Figure 3-19 Torque vector on both feet.....	44
Figure 3-20 Action offset on COB X. ....	46
Figure 3-21 Reward based on robot pitch state and finished distance. ....	47
Figure 3-22 Hierarchical Q-Network architecture. ....	49
Figure 3-23 Experience replay illustration on training data.....	50
Figure 3-24 Solution to non-stationary target DQN.....	51
Figure 3-25 Block diagram of Deep Q-Network on ROS Gazebo simulator. ....	52
Figure 4-1 The 4 types of the 3D object after voxel grid filter. ....	57
Figure 4-2 The comparison of TCVN model performances during training. ....	58
Figure 4-3 The comparison of the TCVN model in a confusion matrix of the validation data.....	59
Figure 4-4 Example results of floor detection using the Tiny-YOLACT model. ....	60
Figure 4-5 Result of validation mAP and FPS using different ResNet backbone. ....	61
Figure 4-6 Snapshot of dragging in the Gazebo simulator.....	62
Figure 4-7 Comparison of accumulated reward during training. ....	63
Figure 4-8 Comparison of Euclidean error during training.....	63
Figure 4-9 Snapshot during training in the Gazebo simulator. ....	64
Figure 4-10 Recorded (states, actions) pair by the learned DQN during testing. ....	65
Figure 4-11 Snapshots testing on plywood surfaces. ....	66
Figure 4-12 Snapshots testing on green carpet surfaces.....	67
Figure 4-13 Snapshots testing on tile surfaces. ....	67
Figure 4-14 The success rate of dragging all objects per each surface. ....	69
Figure 4-15 Recorded (states, actions) pair without using the F/T sensor. ....	69
Figure 4-16 Recorded (states, actions) pair using the F/T sensor. ....	71
Figure 4-17 Example of a failure condition in dragging an empty small suitcase.....	71
Figure 4-18 Example of a failure condition in dragging a big suitcase with a human..	72
Figure 4-19 The pre-defined CoB-X to dragging a big suitcase with a human. ....	72



## List of Tables

Table 3-1 THORMANG-Wolf Specification Details. ....	25
Table 3-2 The network details of the TCVN model.....	34
Table 3-3 The adopted ResNet architecture and number of parameters [56]. ....	37
Table 3-4 Details of the Q-Network architecture.....	49
Table 4-1 Experimental surfaces. ....	55
Table 4-2 Experimental objects.....	55
Table 4-3 Foot-steps parameter. ....	56
Table 4-4 Deep-learning computer hardware specifications.....	56
Table 4-5 OPC (laptop) hardware specifications. ....	56
Table 4-6 List of hyperparameters and values of the DQN .....	63
Table 4-7 Summary and comparison of the success rate result for all experiments. ....	68



# Chapter 1: Introduction

## 1.1. Background

Humanoid robots have become important types of robots that researchers develop and improve them rapidly. In [1], a description of the possible application using a humanoid robot in real-life is provided. In [2], the authors review over last decade's application and influence of humanoid robots in the social, healthcare, and education domains. Recently, in (2019), the humanoid robot applications in a real-world scenario were chosen as the special topic issues in IEEE Robotics and Automation Magazine (RAM)<sup>1</sup>. Therefore, the development of humanoid robots offers significant potential in alleviating tedious and tough tasks that currently performed by humans.

The important question with developing a humanoid robot is “Why humanoid robot? Why not the other types of robots?”. The answer can be indicated as the functions of the humanoid robots itself. Three main fundamental functions of a humanoid robot are evaluated on [3]: (i) Humanoid robots are able to work in the human environment, (ii) Humanoid robots are capable to use humans tools, (iii) Humanoid robots are designed structurally similar to a human shape. As mentioned, a humanoid robot is designed to be similar to mankind. It should mimic a human from different aspects such as interaction, perception, locomotion, manipulation, and behavior.

Generally, humanoid robots were expected to work alongside humans, or as an alternative to humans in any circumstances. For example, in heavy-duty work such as civil engineering and hazardous environments construction, Moving Large and Heavy Objects (MLHO) is required. Moreover, in rescue applications, during the evacuation process, it is necessary to remove the large size of debris. Though biped humanoid robots have high mobility like humans, walking with moving objects has a possibility robot may fall, due to relatively disturbance in the Centre of Mass (COM) with suffering

---

<sup>1</sup> <https://www.ieee-ras.org/publications/ram/special-issues/humanoid-robot-applications-in-real-world-scenarios>

serious damage. So far, many humanoid robot development projects with a focus on the MLHO was still a challenging problem [4-12]. These challenges can be summarized into how to develop a stable walking gait on a biped robot while the robot is dragging a large size object. Admittedly, the dragging problem is more challenging than carrying because there are more uncertainties of surface friction which duplicates the complexity of the problem.

## 1.2. Problem statement

Biped walking humanoid robots may not be stable due to different real-time environment conditions even the desired walking pattern has planned to realize stable walking on the flat floor. However, in the MLHO problem, it is assumed that some objects are too heavy to lift or its shape or size is very hard to carry for a humanoid robot with limited joint torque. Therefore, to deal with this problem, we considered the humanoid robot to pull the object. For this reason, we used the pull motion and then specifically called dragging. This is significant although drag and pulls motion have a similar meaning, however, term dragging is more specific than pull.

The important question in this MLHO motion type, “Why we choose dragging the object rather than pushing the object?”. The answer is illustrated in Figure 1-1, MLHO with dragging motion has more benefit than pushing an object, which is the main target in this thesis is based on that. A study about comparison force on the push and pull an object in flat horizontal surface provided by [13, 14].

Based on Figure 1-1, it shows that there is a difference in friction and forces toward the object between those two tasks. The push motion as shown in Figure 1-1(a), shows the vertical component of the pushing force acts on the object in the vertically downward direction. Therefore, it increases the effective weight of the object and it's mathematically written in Eq (1-1). Whereas, it also affects the friction force between object and ground. The effective weight  $W$  of the object on pushing motion as follow:

$$W = m \cdot g + F \sin \theta \quad (1-1)$$

Where  $m$  is a mass of the object,  $g$  is the gravity,  $F$  is the pushing force, and  $\theta$  is elevation angle of the force given to the object.

On the other hand, the pulling motion shows the reverse way of the vertical force component acts on the object is in a vertically upward direction. Thus, it reduces the effective weight of the object proof on Eq (1-2) and it also decreases friction between the object and the ground. The effective weight  $W$  of the object on pulling motion as follow:

$$W = m \cdot g - F \sin \theta \quad (1-2)$$

Based on these two equations, dragging an object on the horizontal plane is easier than pushing. Note that, although pushing the object can be beneficial in different conditions for a humanoid robot, but it is not the objective in this research study.

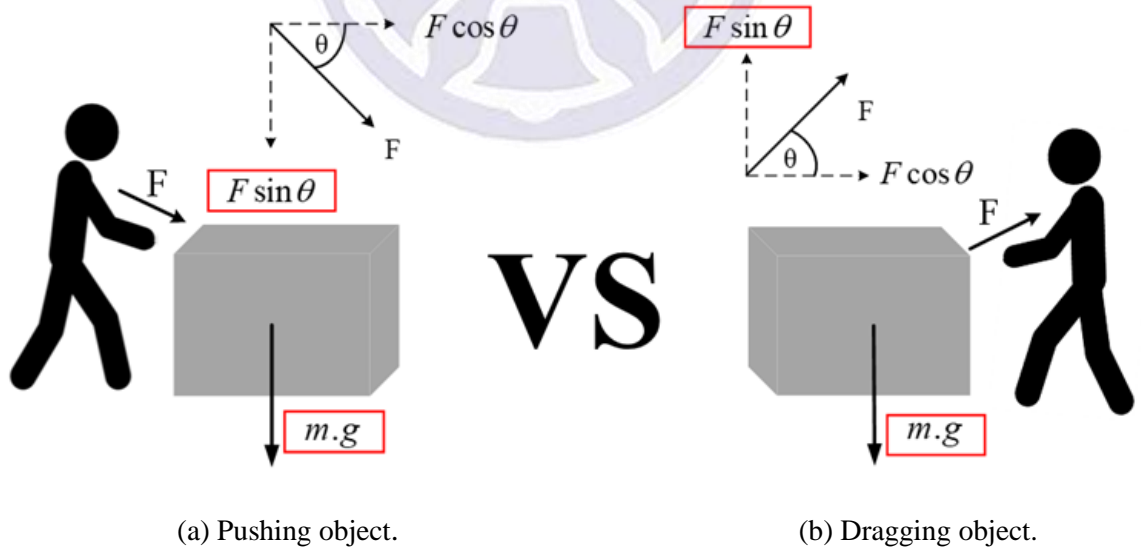


Figure 1-1 Comparison motion pose on the moving object.

### 1.3. The objective of the study

In this work, we present an adult-sized bipedal humanoid robot that is capable of moving a large and heavy object. The objectives of this project are divided into two parts. First, proposing a robot vision algorithm on 3D object detection and 2D object instance segmentation, that uses a deep-learning algorithm approached. Furthermore, in 3D object detection, the object will be acquired using a real-time LiDAR scanner on the robot's head to get the 3D data. On the other hand, the 2D instance segmentation will be expected running in real-time and used for floor detection from the robot's webcam. Second, proposing a deep reinforcement learning algorithm specifically on the Deep Q-Learning algorithm to improve the robot's behavior on whole-body manipulation to transporting large size and heavy objects. Therefore, in the training process, we used a simulated robot model and environment on Gazebo<sup>2</sup>. The advantage of using a gazebo simulator that it can simulate very close to the real environment. As a result, the training resulted can be directly applied to the real robot without any parameter adjustment. This thesis discusses a way of MLHO by a bipedal adult-sized humanoid robot, in which the robot drags different objects including a massive object on various flat surfaces, and walks in a backward direction.

The rest of the thesis is organized as follows. In chapter 2 an overview of the literature review on moving objects using bipedal humanoid robots presented. Chapter 3 explains the methodology of the algorithms to solve MLHO problem, in which the architecture of THORMANG-Wolf robot, vision on the proposed deep learning 3D object classification and floor detection, the bipedal humanoid robot walking control, and the proposed deep reinforcement learning method are presented. Chapter 4 provides the experimental result of the 3D object classification and the proposed method of Deep Q-Network (DQN) on the THORMANG-Wolf robot. Finally, chapter 5 concludes the thesis and shows future work.

---

<sup>2</sup> <http://gazebo.org/>

#### 1.4. Limitation of the study

There are four major limitations in this research that could be addressed in future research. First, the research focused on robot vision processing that is based on a deep learning approach. Also, it divided into 3D voxel object classification from LiDAR point cloud data and real-time instances segmentation for floor detection. The second limitation concern of robot manipulation control, it only used static grasp motion for grasping the object. Third, on the robot walking control, it used the original ZMP walking controller provided from ROBOTIS on the THORMANG3 robot. Finally, in robot behavior control, it specifically uses the deep reinforcement learning on the DQN algorithm to learn the control policy of the Centre of Body (CoB) parameter.



## Chapter 2: Literature Review

### 2.1. Related work

Balancing in Bipedal Humanoid Robot (BHR) systems is a challenging research problem and has been used to address a variety of issues. Hence, there are many the state-of-the-art stabilize walking in biped robot has been extensively studied [15], but walking with disturbance such as pushing [4-8], carrying [9, 10], or lifting [10-12] large or heavy objects are still an open problem. Therefore, maintain the balance of humanoid robots when transporting objects can be one of the critical problems to be investigated by adult-sized BHR. The following literature review confirms that MLHO presents a problem that goes beyond mere balancing, discusses specific and produced solutions, and concludes that specific approaches and robust initiatives are required for real widespread implementation of BHR in the real world.

In the rest of this section, the literature reviews of related works on the MLHO problems are discussed in several sub-section. As a rule, each sub-section is a group of related work in a more specific field and described briefly as follows. (i) Pushing the object, the most common method for transporting large objects. (ii) Pivoting object, an alternative motion for precise movement on moving large objects. (iii) Teleoperation manipulation, manual control of the whole-body humanoid robot to move large-size objects. Then, the humanoid robot control using a learning-based approached on (iv) walking control and (v) push recovery control.

#### 2.1.1. Pushing object

In [4], the authors studied pushing a heavy object by humanoid robot considering the reflect force acted in the end-effectors (both hands). The reaction reflects force aimed at the single support phase of walking. They proposed Dynamically Complemental Zero Moment Point (DCZMP) considering the dynamical modification position of the COM. The COM trajectory of the HRP-2 humanoid robot is modified



based on the forces acting on the robot's hands. These findings were replicated by [16], in which the authors proposed GZMP (Generalized Zero-Moment Point) that enables stability when the robot hands are in contact with objects. They use contact force without grasping to take advantage of keeping robot balance during a disturbance. The author used an HRP-2 humanoid robot in a simulation environment to push an object and proposed GZMP which enables stability when the robot hands are in contact with objects. However, these solutions were tested on large object but not with heavy weight.

In [7], the authors utilized dual-arm force control on a humanoid robot to push a heavy wheelchair. They used a zero-moment-point (ZMP) offset approach, to maintain the balance of the robot. This rectification allows the humanoid robot to dynamically stabilize against the reaction forces. In this method, a real HRP-2 humanoid robot able to push a wheelchair with weight up to 90kg without slipping. Therefore, the importance of friction forces was captured by an expensive force sensor on the robot arms. However, their maintenance is difficult and not all humanoid robot has a force sensor on the arms. Moreover, force contact of a robot can be achieved from the measurement of joint torque without using an additional force sensor. Similarly, by [9], the authors investigated whole-body pushing motion by humanoid robot considering force and balance on different contact points. They used a humanoid robot for pushing heavy objects on the sensor-less region; using both hands, forearm, or the hip. In this research, authors manually generate the posture of a robot try to push unknown mass and COG of the object. A stable pushing force equation from the feet force sensor and external force was utilized for the closed-loop feedback. In this way, the HRP-2 humanoid robot able to push a non-wheeled heavy object. They achieved the highest force from the robot by pushing backward with hip contact. However, the large external reaction force (slip) which was caused by the transported object was not discussed in this study.

In [17], the authors provided a solution for the large reaction of external forces (slip) generated on the feet and hands-on pushing a heavy object. In this study, an optimizer named quadratic programming (QP) was utilized to optimize the joint torques

for predicting the maximum value of external force. Furthermore, this research determined the problem as a free-floating model humanoid robot simulated using the OpenHRP simulator. They used virtual mass (VM) as an alternative for the high computational cost to calculate inequality friction constraint. VM was attached to the end of limbs to estimate contact force between the free-floating model robot and object. Anyway, this work presented in the simulation environment wherein a practical scenario, a QP solver cannot directly deal with the joint torque limitation, because the design variable here is the joint acceleration. In [18], the authors evaluated torque-based balancing to perform a high-force interaction task. Instead of controlling the COM, the proposed controller straight acquires information from the gravity-inertial wrench cone (GIWC) to ensures the practicability of the balancing forces. They tested on TORO humanoid robot with force up to 250N ( $\approx 1/3$  of the robot's weight) able to push the table weighing 50 kg. However, one limitation with this approached that not all humanoid robots support torque control.

### **2.1.2. Pivoting object**

Most previous studies on pushing manipulation show the range of pushing force is wide in hands pushing because the robot is easy to change COG for many joints between contact points and feet. Also, pushing the heavy and large objects in a plane requires generating large force to compensate for the ground-object friction force. This is a challenge because reaction forces from a heavy object can easily cause foot slippage or lose balance and fall. For this reason, pushing large and heavy objects may not perform well on some problems. In [19, 20], the authors validated pivoting motion as an alternative motion for pushing a large object. The robot performed whole-body manipulation of a large object by forward pivoting. Thus, this research maintains the whole-body balance using resolved momentum control (RMC) [6]. RMC was adopted for stepping motion keeping both hands in contact with the object. They tested the result on pivoting heavy objects in both simulation and real robot HRP-2 with displacement in x-direction was around 0.06[m]. The proposed motion had a good performance where there is no slipping occurs during transporting objects. However, pivoting motion took

more time to accomplished moving objects with some distances, as it slowly moves an object through a sequence of pivoting motion to the right and left.

### **2.1.3. Teleoperation manipulation**

Manipulation poses on transportation large and heavy objects, generally were generated manually by human assistance [8]; this finding shows the time completion to finds the perfect configuration is time-consuming. In [21], the authors solve this problem by using teleoperation control for controlling the HRP-2 humanoid robot through a joystick. Meanwhile, the self-balance of the robot learned from the dynamic friction model of the manipulated objects. They showed the robot able to turn, rotate, and push a table with a caster. Anyway, this research required to identified dynamic friction models [22] on every initial interaction with the new load, where solving dynamic friction modeling on a variety object is still a difficult task [23, 24].

On the other hand, another solution proposed by walking imitation of humanoid robot toward human walking recognition provided by [25]. The motion capture system was acquired by using 16 inertial measurement unit (IMU) sensors, placed on the human`s head, torso, and each limb. They achieved motion recognition successfully imitated by the humanoid robot on stance and movement direction with a time delay of 2.5 sec which is very slow. Based on this literature, considering stream a single IMU sensor requires a high-frequency process, it can be concluded that multi IMUs based approaches require a high computational cost for acquiring real-time data. In this regard, both of these approached did not take advantage of any learning algorithm.

### **2.1.4. Walking Balance (Learning-Based)**

Machine learning (ML) algorithm push the technology nowadays by presenting an artificial intelligence of computer performs a specific task without using explicit instructions. On the humanoid robots, the Reinforcement Learning algorithm empowered robot intelligence through reward and punishment from a set of actions taken by the robot. The result was tremendously changed most current research towards this approach. In general, there was no learning-based algorithm has reported for adult-

sized humanoid robots to perform transportation on a large object. So then, the problem of maintaining stability during walking and stance against disturbance, with the problem of transporting large and heavy objects on the humanoid robot are equal. In the following literature, prior work related to RL-based application on BHR that related to this paper will be described respectively.

In [26], the authors designed an RL walking balancing policy, which learns the ankle joint position of the stance leg and determines the swing foot placement during walking. In [27], the authors used Q-Learning to control dynamic walking gait balance and acceleration of biped robot without prior knowledge of the environment. In [28], the authors proposed posture self-stabilizer of a biped robot under exerts amplitude-limited random disturbances using a hierarchical stabilizer based on RL. In [29], the authors aimed a posture-based imitation with balance learning, to allow humanoid robots to imitate demonstrated motions using Q-Learning for the balance learning algorithm. In [30], the authors realized the Deep Deterministic Policy Gradient (DDPG)-based deep reinforcement learning to control the fall over of biped robot to walk steadily on the slope. In [31], the authors utilized a Q-learning algorithm to obtain a straightforward gait pattern to train a humanoid robot to walk straight, where the turning direction is viewed as a gait parameter.

### **2.1.5. Push Recovery (Learning-Based)**

The main objective of the MLHO problem is how to develop a balance system on a BHR. Therefore, likewise to a push recovery, which is also one of an essential method of maintaining the BHR stability. In general, the model-free RL method has an advantage on there is no predefined model given to the robot. The robot learns the optimum policy behavior based on the cumulative reward by trial-error. In the rest of this section, the RL applications in push recovery control problems on BHR will be reviewed to show similarity stability performing transporting large objects problem. Both of the problems should stand against perturbations from external and friction forces.

In [32], the authors applied the RL algorithm on a humanoid robot to learn arms rotation for adapting perturbation in push recovery. They used the off-line Q learning process to solve a computationally expensive problem and applied online execution on the robot. In [33], the authors solve the issue of requirement big data for learning-based approaches that are severely restricted to a physical humanoid robot. They implemented an online RL system on a full-body push recovery controller performing omnidirectional walking. In [34], the authors presented the Dynamical Movement Primitives (DMP) based push recovery for biped humanoid robot, where DMP learned bio-inspired push recovery strategies, such as hip-ankle strategy and step strategy. In [35], the authors developed a full-body push recovery system using Neural-Fuzzy (NF) controller on a general humanoid robot without specialized sensors and actuators. This method uses RL to update the parameter of the NF controller. In [36], the authors employ the Deep Q-Network (DQN) algorithm for high-level push recovery control in small-size humanoid robots, where the reward formula is based on an equation that analyzes the Linear Inverted Pendulum Model (LIPM) from the energy point of view.

#### **2.1.6. Summary of related work**

As far as we know to the best of our knowledge, overall moving large objects was done mostly using the adult-sized HRP-2 humanoid robot platform [4, 7-9, 16, 17, 19-22]. However, there's one approach employed the TORO humanoid robot [18]. Therefore, the most common approach to transporting large and heavy objects was done using the pushing motion. Based on this literature, no approaches were exploiting a learning-based algorithm on whole-body large object transportation using adult-sized BHR. Above all, the RL algorithms are applied to a humanoid robot had shown promise on stabilizing walking and stance posture (push-recovery) due to perturbation given. Reflecting that benefit, in this paper, we introduce the transporting large object on adult-sized BHR problems and propose an RL algorithm to deal with it. In this regard, the robot uses dragging motion to drag heavy and large as a novel solution for pushing problem.

## 2.2. Inverse Kinematic

Inverse Kinematics (IK) calculate corresponding joint angles of a specific link like foot or hand of the robot from a given position and orientation of the cartesian end effector [3]. An example of the IK problem is shown in Figure 2-1. The important question to solve the configuration is shown in Figure 2-1(b). Given a set of joint angles at the left foot is raised by 0.2 m and turned the pitch by 10 deg.

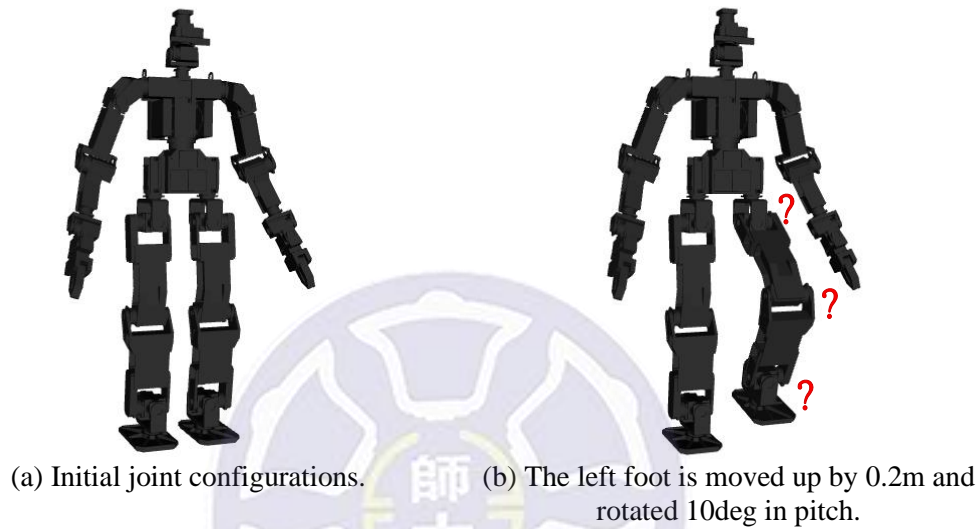


Figure 2-1 Example of inverse kinematic on the left leg of a biped robot;

A humanoid robot is a mechanism consisting of many links connected by joints. Therefore, the theory to analyze the relationship between the position and orientation of each link is called coordinate transformations and rotations. The basic rotation is the rotation around  $x$ ,  $y$  and  $z$  axes, which will call Roll, Pitch, and Yaw respectively.

A rotation point to Roll, Pitch, and Yaw an object from a given angle has to follow the following rotation matrix:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi & 0 \\ 0 & \sin \phi & \cos \phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-1)$$



$$R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-2)$$

$$R_z(\varphi) = \begin{bmatrix} \cos \varphi & -\sin \varphi & 0 & 0 \\ \sin \varphi & \cos \varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-3)$$

Roll, Pitch and then Yaw a point  $p$  around the origin, it will move the point,

$$p' = R_z(\varphi)R_y(\theta)R_x(\phi)p \quad (2-4)$$

A translation by  $a, b, c$  in the  $x, y$  and  $z$  directions respectively has the transformation matrix:

$$Trans_{(x,y,z)} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-5)$$

If we translate point  $p = (x, y, z, 1)^T$ , the translated new coordinate became:

$$p' = Trans_{(a,b,c)}p \quad (2-6)$$

In general, solving IK solutions exist on both the analytical method and the numerical method. Therefore, the position and orientation of set of links with joint angles are defined by nonlinear equations. Since the joint of most humanoid robots are rotational types, the nonlinear problem is unlikely to be solved by nonlinear equations with bunch of variables on the analytical method. However, the derivatives relationship between the position and rotation of a link and joint angles can be represented by linear



equations, and the solution of the IK problem can be solved by finding linear equations through the numerical method.

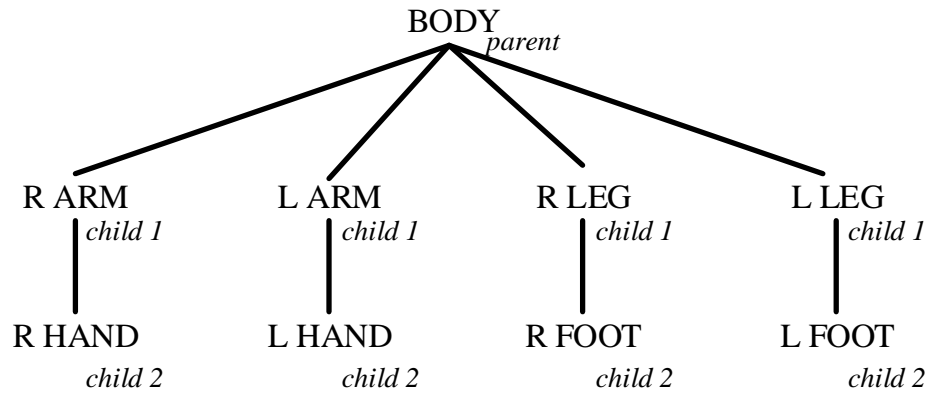


Figure 2-2 Tree structure of the humanoid links connection [3].

The humanoid robot's kinematic structure as shown in Figure 2-2 were formed a tree structure from joining of the links. This is also called the kinematic chain rule of the robot model. Nowadays, the most common way to acquires the IK solution from a kinematic chain is based on the numerical approach. Therefore, one of the famous IK solvers uses Jacobian Pseudo Inverse (JPI) (numerical method) that is available open-source and called Orocos Kinematic Dynamic Library (KDL) [37]. This approach could give an IK solution based on the kinematic chain rule that user-provided.

### 2.3. Walking Gait

Humanoid biped robot walking gait cycles consists of two phases. These phases are divided into Single Support Phase (SSP) and Double Support Phase (DSP). SSP means that the phase is defined when only one leg touches the ground. In SSP, the leg that touches ground called support foot and the leg that not touches the ground called swing foot. On other hands, DSP is defined when both of leg touches the ground. The sequences of walking are illustrated in Figure 2-3, starting by the SSP phase followed by the DSP phase and continuously [38].

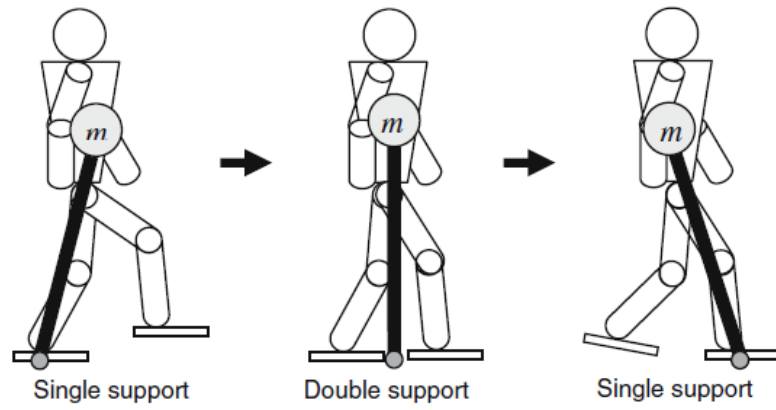
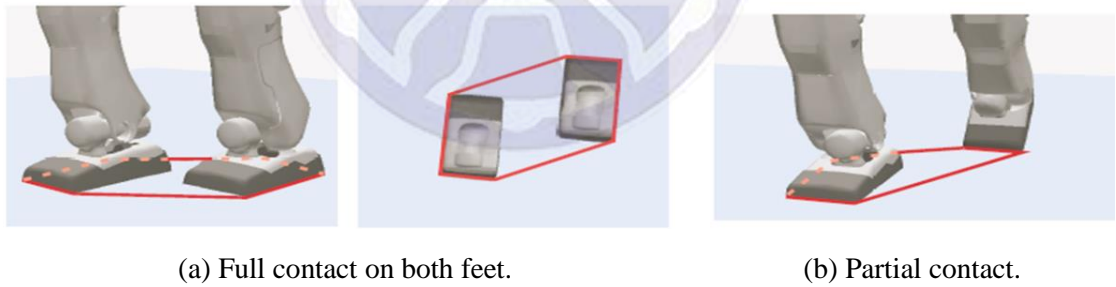


Figure 2-3 Sagittal plane view of walking gait cycle [38].

A humanoid robot is structurally the same as humans, but controlling walk on the robot is not as rigid as it looks. A humanoid robot needs to maintain its balance contact between the foot and ground while walking. For this purpose, Zero Moment Point (ZMP) is the most famous biped humanoid walking control [39]. ZMP is the reference point of the robot's combined force of gravity and ground inertial force. During the walking of the robot, if its ZMP is regularly located in the support polygon area, the robot will never fall.



(a) Full contact on both feet.

(b) Partial contact.

Figure 2-4 ZMP support polygon [3].

Figure 2-4 illustrated the region formed by enclosing all the contact points between the robot and the ground by using an elastic cord braid is called support polygon. The projection of ground with Centre of Mass (CoM) can be displayed outside of the support polygon. However, ZMP always exists inside of the polygon support. Therefore, humanoid robots can keep balance if the ground projection of CoM is located inside of the support polygon as shown in see Figure 2-5.

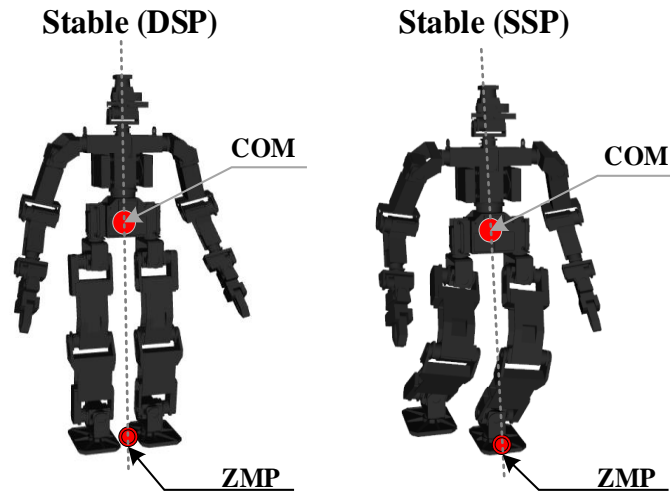


Figure 2-5 Projection of the Centre of Mass on Zero Moment Point.

## 2.4. Neural Network

Artificial intelligence (AI) has become the most famous technique in robotics applications. One of the most powerful and widely used in AI algorithms is the Neural Network (NN). The main reason behind it, because NN presents an intelligence demonstrated by a machine that works similarly to the human brain. Briefly, the architecture of NN is consists of an interconnected number of nodes called neurons, that are organized in layers to process the data information.

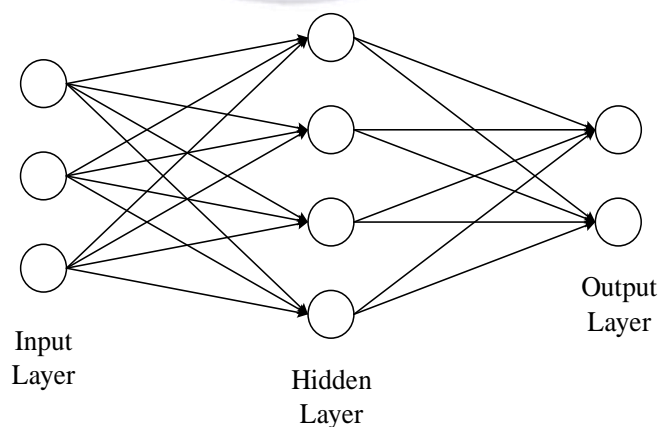


Figure 2-6 Neural network architecture.

Figure 2-6 represents a NN architecture looks like. When we zoom in to one of the hidden or output nodes, each node is called perceptron that illustrated in Figure 2-7.

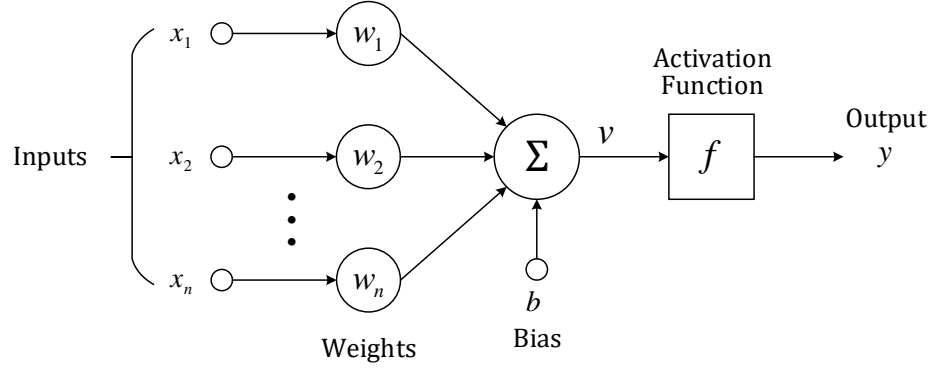


Figure 2-7 Operations done by neurons on a single layer perceptron.

The neurons process on single-layer perceptron shows in Figure 2-7 is the math calculations that denotes in the equation below:

$$y = f\left(\sum_{i=1}^n w_i x_i + b\right) \quad (2-7)$$

As shown in Figure 2-7 and denotes in Eq (2-7) the process can be described briefly as follows. (i) First, the inputs  $x_1, x_2, x_3$  are multiplied by variable weight  $w_1, w_2, w_3$  before it being sum up. Each neuron connection has its weight  $w_n$ , and during the learning process, those variables are the only parameter that will be tuned. (ii) Next, a bias  $b$  value is added to the total value calculated, it is not a value from a specific neuron. (iii) Finally, after all of those summations, the neuron applies a function called “activation function” to the obtained value.

## 2.5. Deep Learning

Deep learning (DL) is a subset of machine learning forms by artificial neural networks (ANN). The DL networks are similar to ANN but with deeper architecture (multiple hidden layers). The learning methods in DL can be supervised (labeled data) or unsupervised learning (not need labeled data). Additionally, in the DL algorithm, a large dataset is required to trains the model. An instance of the illustrated deeper network architecture of the DL model as shown in Figure 2-8.

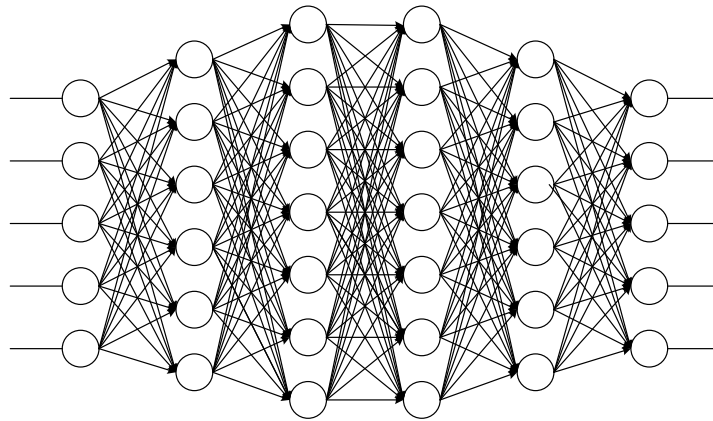


Figure 2-8 Deeper network architecture of ANN or called Deep Learning.

Despite the function of ANN, automatic data feature extraction is another function of deeper network architecture. Moreover, the feature extraction in the DL model layer is well famous applied in the image processing task. This layer is called a convolutional layer, which can obtain feature maps from several filtrations on the image (see Figure 2-9).

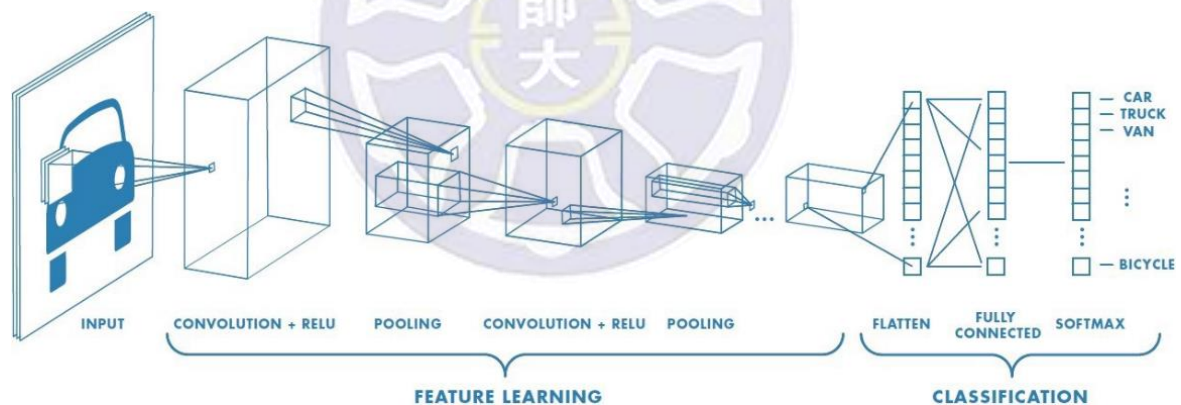


Figure 2-9 Convolutional Neural Network subclass of deep learning<sup>3</sup>.

Not only in image processing, several famous applications of DL as automatic speech recognition, visual art processing, natural language processing, recommendation systems, bioinformatics, fraud detection, mobile advertising, etc.

<sup>3</sup> <https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html>

## 2.6. Object Detection

Object detection in computer vision is a method to find a target object in a digital image or video. Target object detection can be single and also multiple. In robotics applications, object detection has become fundamental as robot perception. Therefore, object detection can be divided into different types (see Figure 2-10). Whereas most approaches in object detections are based on DL-Convolutional Neural Network (CNN) approached.

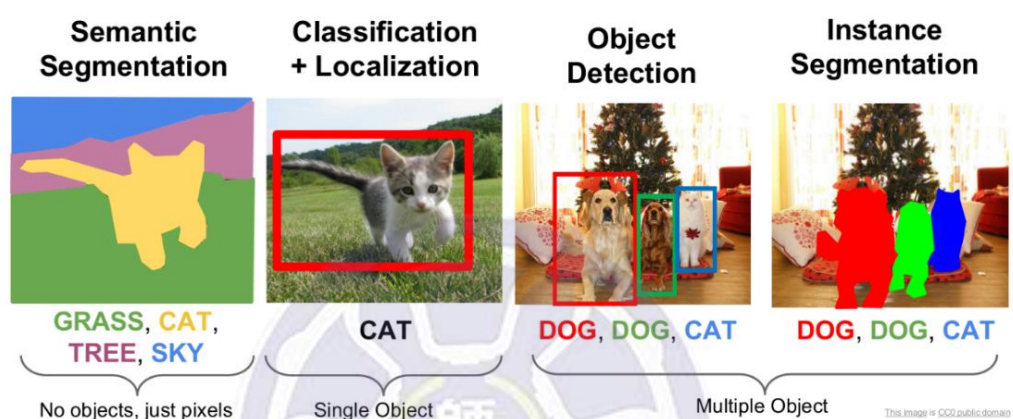


Figure 2-10 Various types of 2D image object detection<sup>4</sup>.

As shown in Figure 2-10, the types of famous object detection in the 2D frame will be briefly introduced in respectively. (i) Semantic segmentation is a technic to label each pixel in the image with a category label, it doesn't differentiate instances and only care about pixels. The most notable semantic segmentation is based on fully CNN architecture [40]. (ii) Classification and localization are the common object detection technique that finds object position and simultaneously classified the object name. There are several famous researched on this approach: Faster R-CNN [41], Single Shot MultiBox Detector (SSD) [42], You Only Look Once (YOLO) [43]. (iii) Instance segmentation is different from semantic segmentation that includes identification of boundaries of the objects at the detailed pixel level. Therefore, few works have focused on instance segmentations: Mask R-CNN [44], FCIS [45], YOLACT[46].

<sup>4</sup> [http://cs231n.stanford.edu/slides/2017/cs231n\\_2017\\_lecture11.pdf](http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf)



Object detection is not limited only to 2D frames. It has an expansive to 3D object detection. Therefore, in computer vision, 3D object detection is obtained from point clouds data that form a 3D model. See Figure 2-11, an example of 3D object detection in object classification, part segmentation, and semantic segmentation by [47].

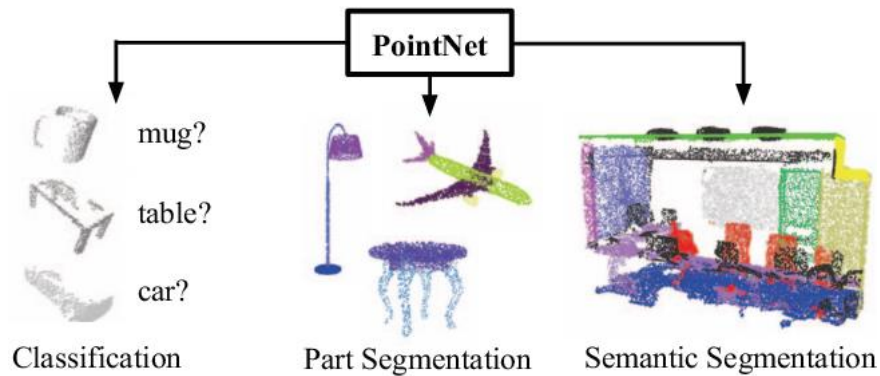


Figure 2-11 Types of 3D point cloud object detection by [47].

## 2.7. Reinforcement Learning

Reinforcement learning (RL) subset of machine learning that differs from other types of machine learning. The main difference is that based on trial and error, there is no supervisor and only depend on a reward signal. The environment is initially unknown, where time matters. During the agents interact with the environment, it also improves its policy.

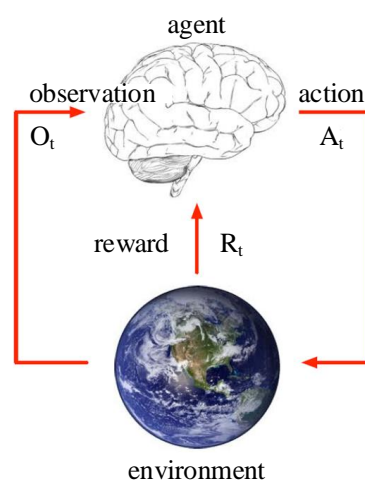


Figure 2-12 Markov Decision Process of Reinforcement Learning<sup>5</sup>.

<sup>5</sup> [http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching\\_files/intro\\_RL.pdf](http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/intro_RL.pdf)



The flow process of RL as shown in Figure 2-12, the process is divided into agents and environment interaction. Each step  $t$  by the agent: (i) executes an action  $A_t$ , (ii) receives observation  $O_t$ , and (iii) receives a scalar reward  $R_t$ . Meanwhile, in the environment: it receives an action  $A_t$ , then emits observation  $O_{t+1}$ , and finally receives a scalar reward  $R_{t+1}$ . These steps learning processes are performed periodically in episodic time-based. This means that in every single episode, the process took a set of actions based on increment at the environment step  $t$ . Therefore, the mathematical formulation of the RL problems can be defined as Markov Decision Process (MDP).

A reward  $R_t$  is a scalar feedback signal, indicates how well an agent is doing at step  $t$ . All goals can be described by the maximization of Eq (2-8) expected cumulative reward:

$$R_t = r_t + r_{t+1} + r_{t+2} + \dots + r_n \quad (2-8)$$

During the training process, an agent should care about immediate rewards to rewards in the future. This is called a discounted factor  $\gamma \in \{0..1\}$  in cumulative reward. If  $\gamma = 0$ , means the agent only cares about the first reward. On the other hand,  $\gamma = 1$ , means agents care about all future rewards.

$$R_t = r_t + \gamma \times r_{t+1} + \gamma^2 \times r_{t+2} + \dots + \gamma^{n-t} \times r_n \quad (2-9)$$

The agent's job is to maximize cumulative reward. To achieve that, RL must try to get the optimal value function, i.e. the maximum sum of cumulative rewards. Bellman equation [Eq (2-10)] helps the agent get the optimal value function.

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a') \quad (2-10)$$

In model-free RL, to learn with no prior knowledge of the environment can use the Temporal-Difference (TD) learning. The methods learn directly from episodes of experience. It can be mathematical formulate in the below equation [Eq (2-11)], observation before versus observation now:

$$TD = [r + \gamma \max_a Q(s', a)] - [Q(s, a)] \quad (2-11)$$

Moreover, to learn the optimal value-function in the off-policy (randomly explore the environment), TD learning is combined with the Bellman equation. Therefore, it also called as Q-Learning and express in the below equation [Eq (2-12)]:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_a Q(s', a)] \quad (2-12)$$

Whereas, Q-Learning algorithm required a Q table to store the Q-values based on state, action it takes, and rewards it acquires during the training process [48].

## 2.8. Deep Reinforcement Learning

In traditional RL algorithms, the major limitation of the approach is limited to small problem spaces and few possibly state in the environment [48]. This is also called Q-Table, where the size of the table depends on the numbers of action and state. However, this method is not suitable when the states of the environment are substantial. Later on, the famous Deep Reinforcement Learning (DRL) algorithm was introduced by [49]. The authors utilized deep neural network architecture into RL, to replace the Q-Table and called it Deep Q-Network (DQN). Admittedly, the benefit of DQN that agents can learn a more complex environment. It allowing to have better generalization for unknown states and able to take action that never seen before. The illustration of the DQN algorithm is shown in Figure 2-13.

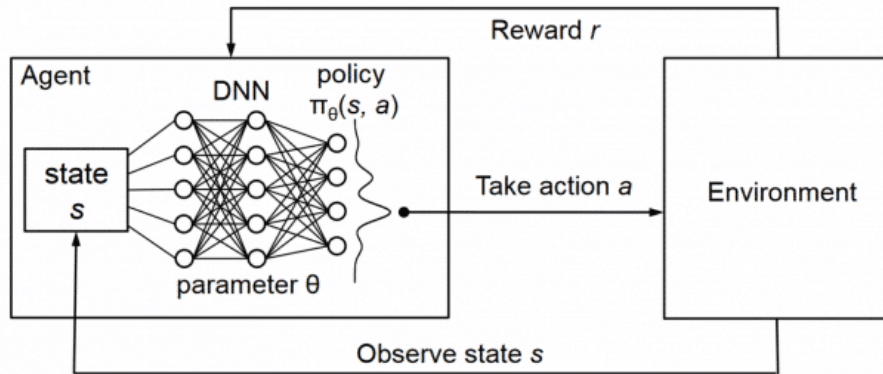


Figure 2-13 Deep Q-Network architecture [50].

## Chapter 3: Methodology

In this chapter, the methodology of the proposed algorithm will be discussed. The systems technique to approach this thesis is divided into four important parts. First, overviews of the architecture THORMANG-Wolf adult-sized humanoid into hardware, software description. Then, the proposed novel hierarchical learning method to solve the Moving Large size and Heavy Object (MLHO) problem is described. Second, explains the robot vision process. It distributed into two types of proposed DL object detection that are 3D object detection and floor detection. Third, it presents the robot walking controlling. It contains the biped robot walking controller in the ZMP walking controller. Finally, the details of robot behavior controlling. It shows the proposed DRL algorithm to achieve this dragging task. Deep Q-Learning is chosen as a type of DRL method to control the walking meanwhile drags an object.

### 3.1. THORMANG-Wolf Robot

THORMANG (Tactical Hazardous Operations Robot) is a full-size commercially available bipedal humanoid robot developed by (ROBOTIS, Inc) [51]. The main objective of the robot is to design an adult-sized humanoid robot as a researched platform. Currently, the latest version of this robot is called THORMANG3 as shown in Figure 3-1(a). However, due to application requirements in this project, we have a slight modification on THORMANG3 and named as THORMANG-Wolf. In the rest of this sub-section, the details of our THORMANG-Wolf robot are discussed into three parts including the hardware description, the software description, and the proposed hierarchical learning-based algorithm design.

#### 3.1.1. Hardware Description

The main difference in mechanical appearance between THORMANG3 robot and THORMANG-Wolf robot is on the webcam of the robot. As original THORMANG3 mechanical design has USB webcam Logitech C920 HD that has a

limited field of view (FOV), we replaced it with Logitech C930E version that has an expansive 90-degree wide FOV<sup>6</sup>. The dimensions of the robot are stayed the same as the original and illustrated in Figure 3-1(b). It has a height of 137.5 cm, a width of 42.4 cm, and its weight including the two batteries is 42 kg. The details specification of the robot is shown in Table 3-1.

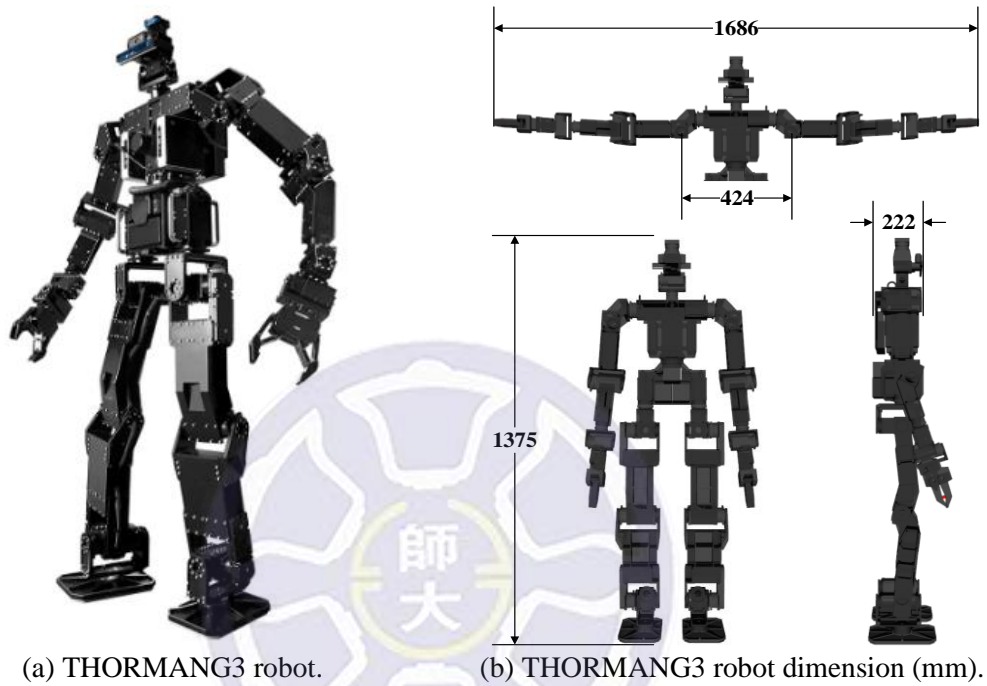


Figure 3-1 THORMANG3 adult-sized humanoid robot.

THORMANG-Wolf robot has wide ranges of manipulation and walking motions with an overall 29 Degree of Freedom (DOF) in total. The actuators for robot kinematics are consisting of three different models of Dynamixel-PRO series and 1-DOF two-fingered Dynamixel hand. Hardware components of the robot are equipped with advanced computational power and sophisticated sensors (see Figure 3-2). It has two minicomputers, one monovision webcam, one depth camera, one LiDAR scanner, one force and torque (F/T Sensors), and one speaker. The robot's electrical power is supplied through two batteries, which grouped into the 22.2 Volt for actuators and 18.5 Volt for controllers and sensors.

<sup>6</sup> <https://www.logitech.com/en-us/product/c930e-webcam>

Table 3-1 THORMANG-Wolf Specification Details.

Category	Specification	Value
<b>Dimension</b>	Weight	42 Kg
	Height	137.5 cm
<b>DOF</b>	Head	2 DOF
	Arm	$2 \times 7$ DOF
	Leg	$2 \times 6$ DOF
	Waist	1 DOF
<b>Actuator</b>	H54-200-S500-R	$10 \times 200$ W
	H54-100-S500-R	$11 \times 100$ W
	H42-020-S300-R	$8 \times 20$ W
	RH-P12-RN	$2 \times 80$ W

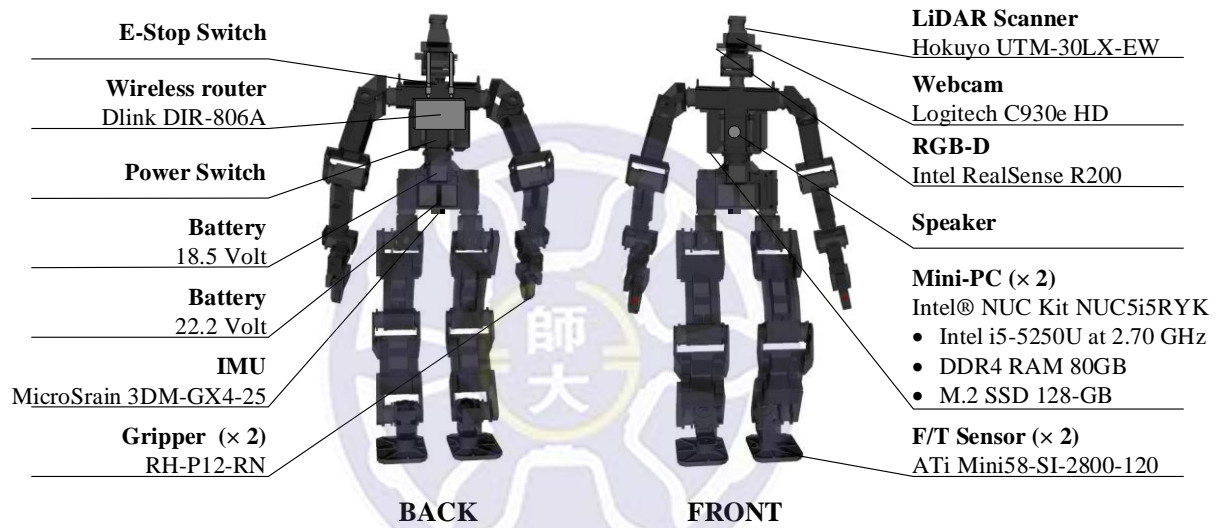


Figure 3-2 THORMANG-Wolf hardware architecture.

The electrical components of the THORMANG-Wolf robot are shown in Figure 3-3. The main controller is distributed into three computers: (I) Perception Personal Computer (PPC), (II) Motion Personal Computer (MPC), and (III) Operating Personal Computer (OPC). Two computers (MPC and PPC) are attached to the robot and one computer (OPC) is located outside of the robot. MPC handles the dynamic kinematic system of the robot that computes every joints movement by translating into positions of actuators. PPC for perception processing that is acquiring sensors data in the robot. OPC works as manager processing to integrate the MPC and PPC. Therefore, to accommodate a multi-computers communication system in this robot, a router located



on the back of the robot is employed for the ethernet connection across this collaborative computer system.

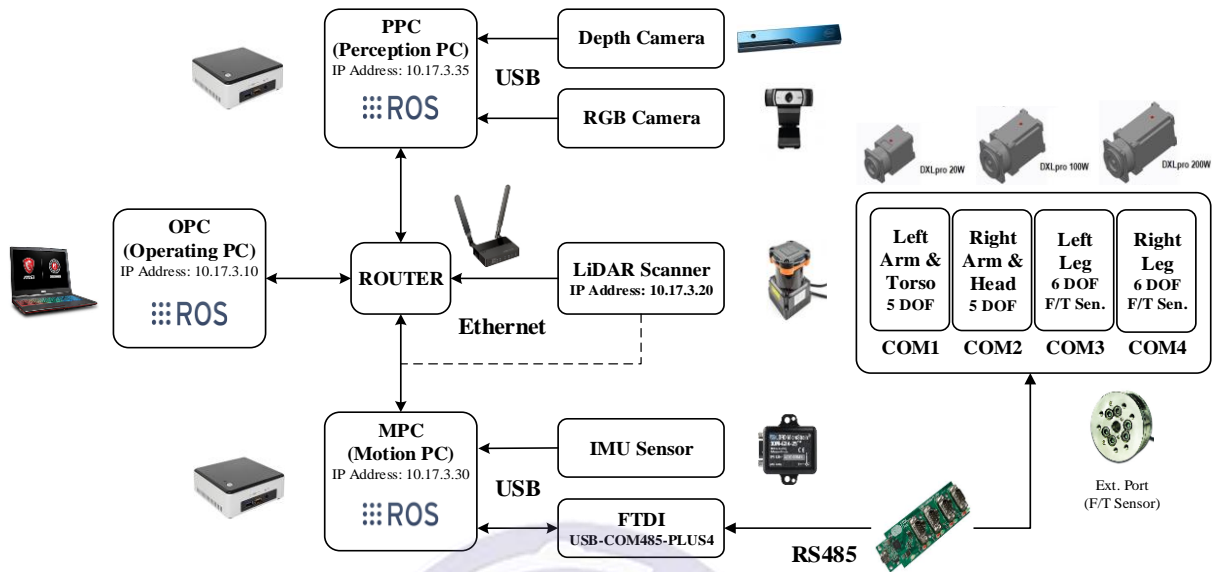


Figure 3-3 THORMANG-Wolf electrical components system.

### 3.1.2. Software Description

The software system of the THORMANG-Wolf robot was built initially using the Robot Operating System (ROS) software. Whereas, the ROS Kinetic Kame version is chosen as the compatibility version alongside with Ubuntu 16.04 (Xenial) operating system. It is well known that ROS is a set of software libraries and tools that was originally designed for robotic applications<sup>7</sup>. The main advantage of using ROS is the message passing feature that can easily be developed under a multi-computer communication system. Another benefit is the multi-language programming compatibility. Therefore, the software description of the THORMANG-Wolf robot is illustrated in Figure 3-4.

Figure 3-4 presented the simplified ROS graph architecture of the THORMANG-Wolf robot on the dragging task. The core management of the systems is distributed into three different computers. Those three types of computers are as follows respectively: (i) PPC computer preprocess sensor perception acquisition on webcam and LiDAR scanner into two different ROS topics: “/rgb\_image” and “/point\_cloud”. (ii) MPC

<sup>7</sup> <https://www.ros.org/about-ros/>

computer provides “/robot\_state” directly from the F/T and IMU sensors, then subsequently calculates the dynamic kinematic in the “/CONTROL\_MANAGER” to read and write positions of each joint. (iii) OPC computer works as a management substance inside the “/MAIN” node to manage behavior control from PPC sensor input into the MPC action movement. Overall, those multi-computers systems were done using ROS to have a synchronized system of a humanoid robot in perception, behavior, manipulation, and locomotion.

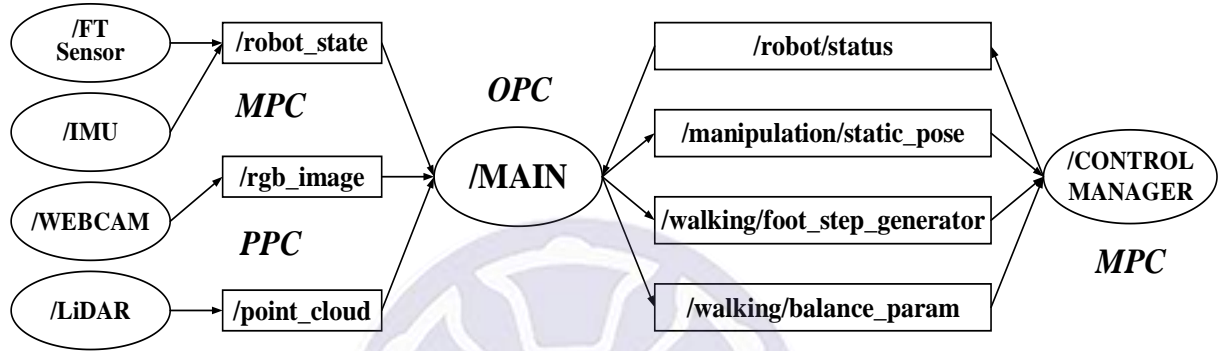


Figure 3-4 ROS graph architecture performing a dragging task.

**Note:** ROS node and topic is represented by ellipses and rectangle shape respectively.

### 3.1.3. The Proposed Algorithm Design

The algorithm design for the dragging task in the THORMANG-Wolf robot is a hierarchical independence framework at three different levels including robot vision process, behavior control, and motion control. The details of those frameworks are illustrated in Figure 3-5.

As it is illustrated in Figure 3-5, first, the vision process is divided into two sub-categories: (I) Object detection and (II) Floor detection. Object detection reads point clouds data from the LiDAR scanner and then feeds into the proposed DL for classifying object type. A result of object detection results will be used to determine pre-recorded manipulation motion for grasping the object. On the other hand, the floor detection uses RGB images obtains from the robot webcam. It feeds the images into the proposed lightweight DL algorithm for real-time instance segmentation on floor types detection. So then, the floor type result goes to DQN to adjust the offset coefficient of the Centre of Body (CoB).



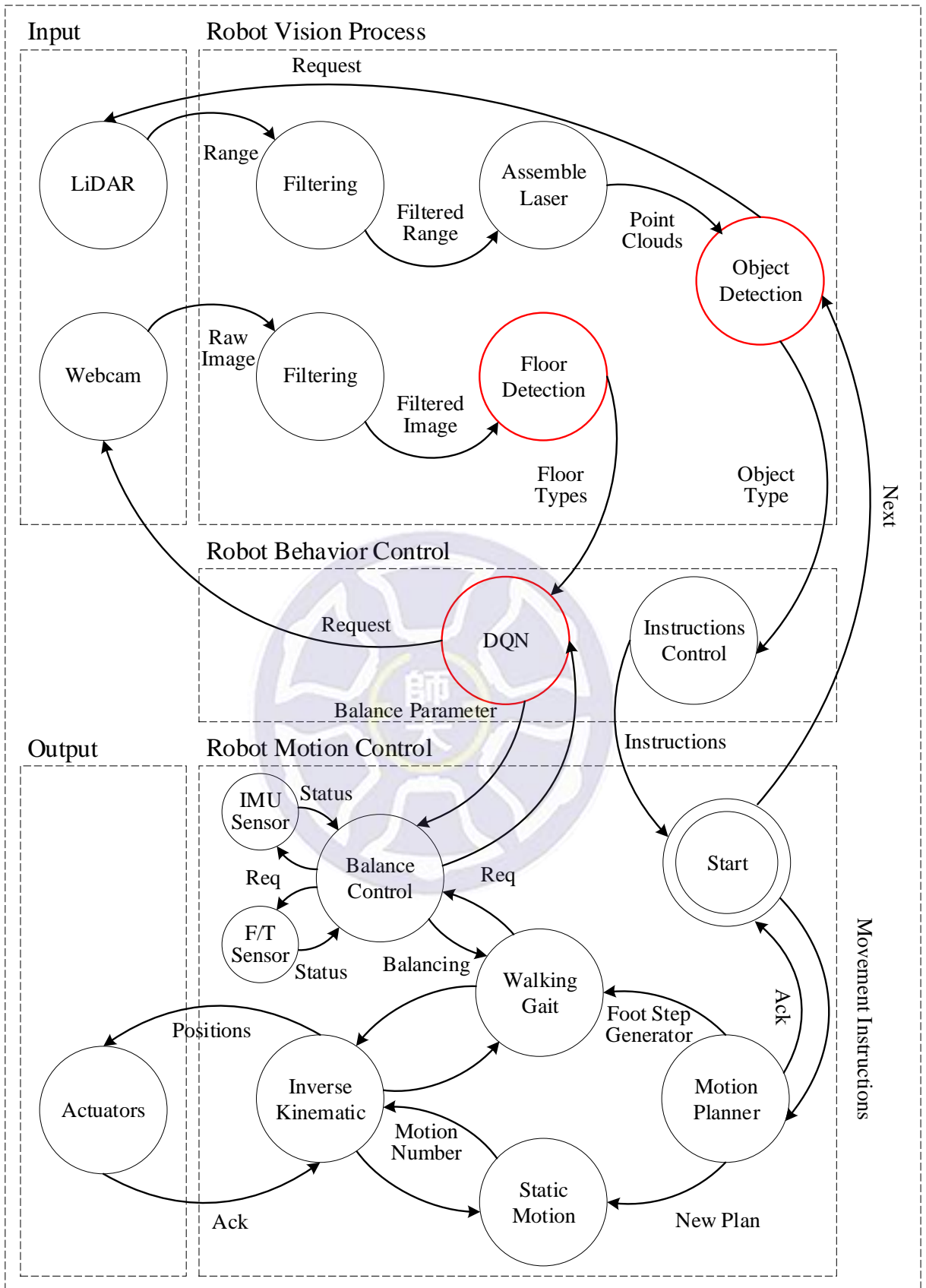


Figure 3-5 THORMANG-Wolf hierarchical framework data flow diagram.

**Note:** The term “Ack” indicating a process termination acknowledgment and red color shape points out the proposed hierarchical deep learning algorithms.

Second, the behavior control, which is also the main proposed DRL method in this MLHO problem. For this purpose, we used a DQN algorithm to learn the parameter of the walking balance control policy. This algorithm learns the behavior control based on robot states that are acquired from IMU and F/T sensors. As a result, the setpoints of  $CoB_x$  parameters were tuned automatically by the DQN algorithm in real-time during the dragging procedure.

Finally, the motion control, which handles all processes of robot movement. It comes from instructions controls to take action in sequential order. Moreover, there are two main functions of motion control are described as follows. At first, the grasping motion act as a motion manager that can store and play the recorded grasping motions for various objects. Then, followed by the walking control, it produces the walking footstep generator by solving the inverse kinematics of legs using Pseudo Jacobian Inverse and generates walking gait pattern.

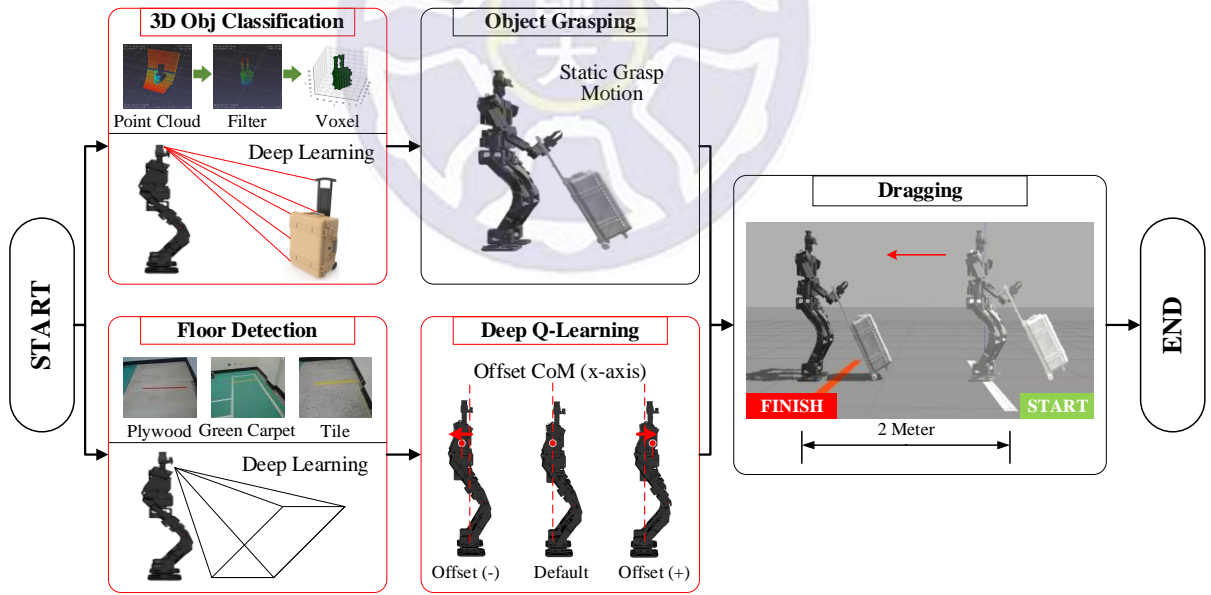


Figure 3-6 Block diagram of the proposed DL algorithms to solve MLHO problem.

**Note:** the red color blocks are the proposed algorithms.

In this regard, based on the hierarchical framework of the algorithm design illustrated in Figure 3-5, our proposed learning-based algorithms consists of three learning phases respectively to solve the problem as follows: (i) Deep Learning algorithm on 3D object classification. (ii) Deep Learning algorithm on real-time instance

segmentation for floor detection. (iii) Deep Reinforcement Learning algorithm on the walking balance control policy. Therefore, to clarify the MLHO process in sequential order, drawn a block diagram of the proposed hierarchical learning-based algorithm to solve MLHO problems more clearly as illustrated in Figure 3-6. In summary, each part of the data flow diagram in Figure 3-5 with the proposed hierarchical methods on the MLHO problems will be described individually.

## **3.2. Robot Vision Process**

It is a very significant point on a humanoid robot to have a vision system to visualize the environment and identified objects. The problem of this dragging task can be simplified to a robot need to know what kind of object it will move and on which type of surfaces. In this section, the vision processing of the robot is divided into (I) 3D object detection and (II) Floor detection (instance segmentation). However, both processes are based on DL approaches and will be described below.

### **3.2.1. 3D Object Detection (Deep Learning)**

Single two-dimensional (2D) images from a robot camera actually can provide an instance of visual information to this problem. However, information from the 2D image is limited to the two-dimension projection of length and width, whereas the three-dimension (3D) of the object's height is indistinguishable. Unlike 2D images, point clouds data contains 3D data that provide a rich source of information. Therefore, the point clouds are acquired by using the LiDAR scanner from the robot. The main objective of this approach is to use the LiDAR scanner to classified objects. After the object has been classified, the output will be used for selecting pre-recorded manipulation motion to grasp the object.

It is well-known that the state-of-the-art 2D image object recognitions were based on Convolutional Neural Network (CNN) [40-44]. The same concept also has been applied for 3D point cloud data object detection by using CNN as well [47, 52, 53]. On the other hand, in contrast to the camera, LiDAR has no interference with lighting

conditions that leads to increasing the robustness of the system. Therefore, the implementation of this object classification algorithm is based on CNN. The general flowchart of the proposed object classification is illustrated in Figure 3-7.

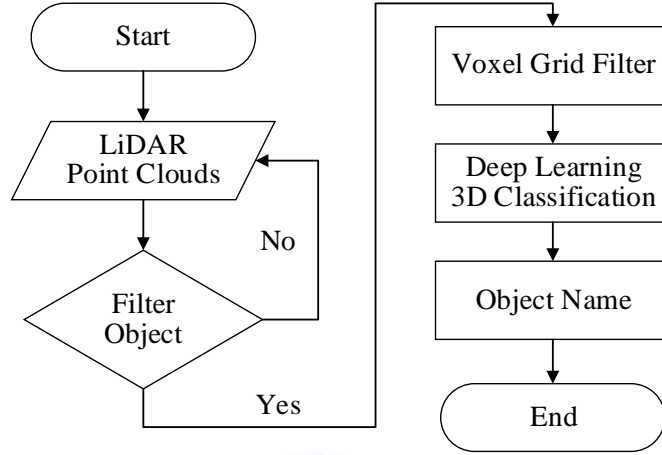


Figure 3-7 Flowchart DL algorithm of 3D object classification.

The working process on DL 3D object classification is described in the following. First, the LiDAR point cloud data were acquired from the robot head's scanning process. As shown in Figure 3-8(a) the LiDAR point clouds data includes additional information about the environment from scanned results. To tackle this issue, it is suitable to perform filtering on point cloud data. Removing additional features from raw point clouds, in other words, to extract important information, will lead to increases and robustness of DL models. For this reason, a proposed heuristic algorithm is applied to filtering and extract the object from a cluttered environment as shown in Figure 3-8(b). In this process, Euclidean distance-based filtering to extract the object from the environment is proposed and given by the following formula.

$$d_{(p,q)} = \sqrt{\sum_{i=1}^3 (p_i - q_i)^2} \quad (3-1)$$

Where  $p$ , and  $q$  are two points in Euclidean space, then the distance  $d$  from  $p$  to  $q$  is calculated by each axis  $i$ , indicate the axis of  $(x, y, z)$  respectively. So, in the

filtering process, if there was no object in front of the robot, it redoes the scanning process for collecting the point cloud data.

Based on the structure of DL algorithms in general, DL models are required a fixed amount of input size to feed into the model. Also, it is well-known that the total number of points given each time of LiDAR scanned result has no fixed shape. To deal with this, Voxel Grid (VG) filter is utilized to downsample the point cloud data into a regular voxel grid representation [54]. Therefore, in the VG process, the filtered object point clouds were discretized spatially as binary voxel at  $30 \times 30 \times 30$  volumetric occupancy grid (see Figure 3-8(c)), where each voxel is assumed to have a binary state (occupied or unoccupied). Next, a fixed size of voxels data (occupancy grids) is continued to the input of the DL model. During this process, the DL model performs mathematical calculations for processing this 3D classification task. Finally, the last process also known as the output of the DL model will give a predicted answer based on the highest probability to recognize which type of object.

Exploiting volumetric representation of the voxel data for 3D shape recognition, the empirical applications of shape recognition have become popular in the DL-based approaches [52, 53]. The most notable 3D shape recognition, that integrating a volumetric occupancy grid representation with a supervised 3D CNN provided by [52]. In [52], the authors introduced VoxNet, as a 3D CNN multi-class classification task on binary voxels data with a simple network architecture resulting in real-time performance. One other research study on volumetric 3D object multi-class classifications was presented in [53]. In [53], they proposed a lightweight Volumetric-CNN1 (V-CNN1) model. In this method, the volumetric 3D object was represented in the form of a set of spatially convoluted 2D images (known as feature maps). So, instead of using a 3D convolutional layer, the authors use a 2D convolutional layer for convoluting the 3D volumetric occupancy grid and achieved a faster training process (because of using fewer parameters). Although 2D convolutional were outperformed in 2D images classifications [40-44], the result was not as good as in the 3D occupancy grid [53]. It shows on V-CNN1, the classification accuracy was slightly decreased in comparison to VoxNet [52] (use 3D convolutional) on the same 3D data set.



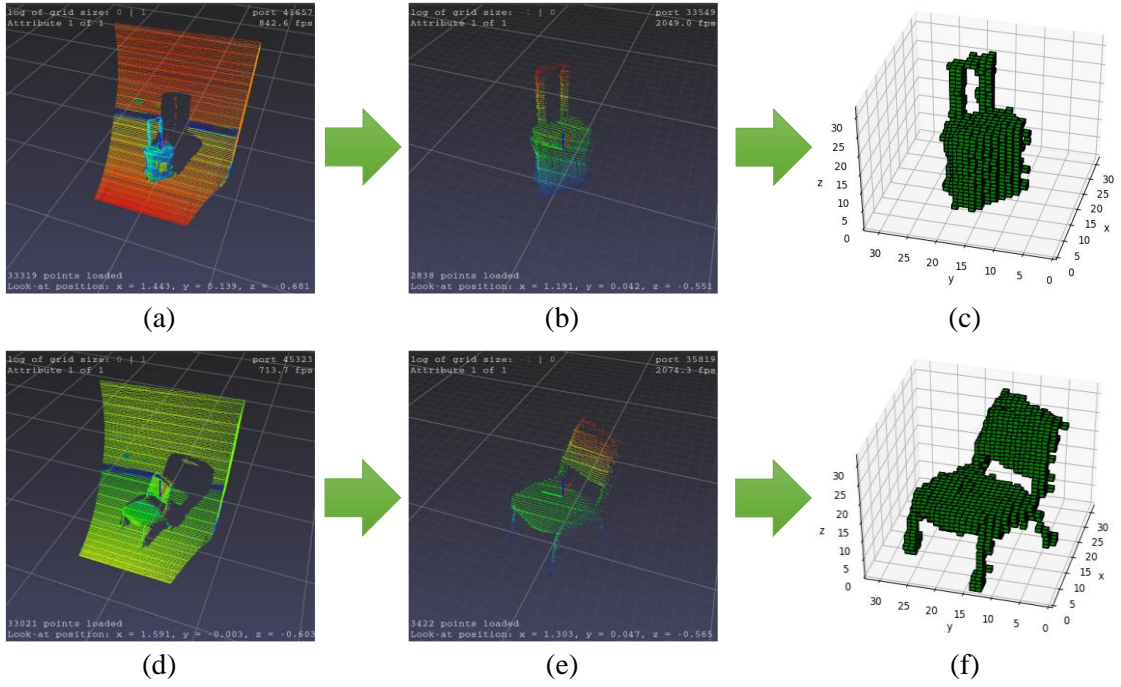


Figure 3-8 Example process of preprocessing 3D point cloud data.

**Note:** left figure: raw point cloud data, middle figure: filtered point cloud data, and right figure: after voxel grid filter.

Regarding the two mentioned models in the previous paragraph, that were outperformed on multiclass classification in the 3D volumetric occupancy grid. However, there are shortcomings of existing models described as follows. In the VoxNet model [52], the authors consider only a very small network that contains only two 3D convolutional layers and two fully-connected layers. In this regard, shallow network architecture caused the model lacks to generalize the data (learn more features on various levels) [55]. On the other hand, in V-CNN1 [53], the authors used deeper network architecture (depth-5 layers) but fail to establish a relationship between 3D data from the 3D convolution benefits. So, in this thesis, we proposed a new model called Three-layers Convolution Volumetric Network (TCVN) as a robust learning method to tackle issues of previous models. As shown in Figure 3-9, TCVN is based on the VoxNet and V-CNN1 reference concept, which is using deeper network architecture along with a 3D convolution layer.

As shown in Figure 3-9, the proposed model using a volumetric occupancy grid computed with size  $30 \times 30 \times 30$ . This model consists of three 3D convolutional layers, all with 32 filters of size 3 and stride 1. Correspondingly, the convolution layers are followed by batch normalization and three ReLU activation functions along with two



max-polling layers. The ReLU layer is to introduce non-linearity in the model by activating only positive neurons. The pooling layer following ReLU ensures that neurons do not contribute to the model from learning redundant information of spatial voxel. Also, there are two fully connected layers in the last part of the model, where the final fully connected layer used a SoftMax function to normalize the probability distribution of each class score. During the training, dropout with a probability of 0.5 is used to prevent overfitting, and an Adam optimizer with a standard base learning rate of 0.001 was employed for updating the model weights. Overall, the details of the proposed model are presented in Table 3-2.

Table 3-2 The network details of the TCVN model.

Layer type	Filter size / Dropout rate	Stride	Output Size	Number of parameters
<b>Convolution 3D</b>	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$32 \times 30 \times 30 \times 30$	896
<b>Batch Norm.</b>	-	-	$32 \times 30 \times 30 \times 30$	128
<b>ReLU</b>	-	-	$32 \times 30 \times 30 \times 30$	-
<b>Max pooling 3D</b>	$2 \times 2 \times 2$	$2 \times 2 \times 2$	$32 \times 15 \times 15 \times 15$	-
<b>Convolution 3D</b>	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$32 \times 13 \times 13 \times 13$	27680
<b>Batch Norm.</b>	-	-	$32 \times 13 \times 13 \times 13$	128
<b>ReLU</b>	-	-	$32 \times 13 \times 13 \times 13$	-
<b>Convolution 3D</b>	$3 \times 3 \times 3$	$1 \times 1 \times 1$	$32 \times 11 \times 11 \times 11$	27680
<b>Batch Norm.</b>	-	-	$32 \times 11 \times 11 \times 11$	128
<b>Max pooling 3D</b>	$2 \times 2 \times 2$	$2 \times 2 \times 2$	$32 \times 5 \times 5 \times 5$	-
<b>Dropout</b>	0.5	-	$32 \times 5 \times 5 \times 5$	-
<b>Fully connected</b>	-	-	2048	8194048
<b>Batch Norm.</b>	-	-	2048	8192
<b>Fully connected</b>	-	-	5	10245

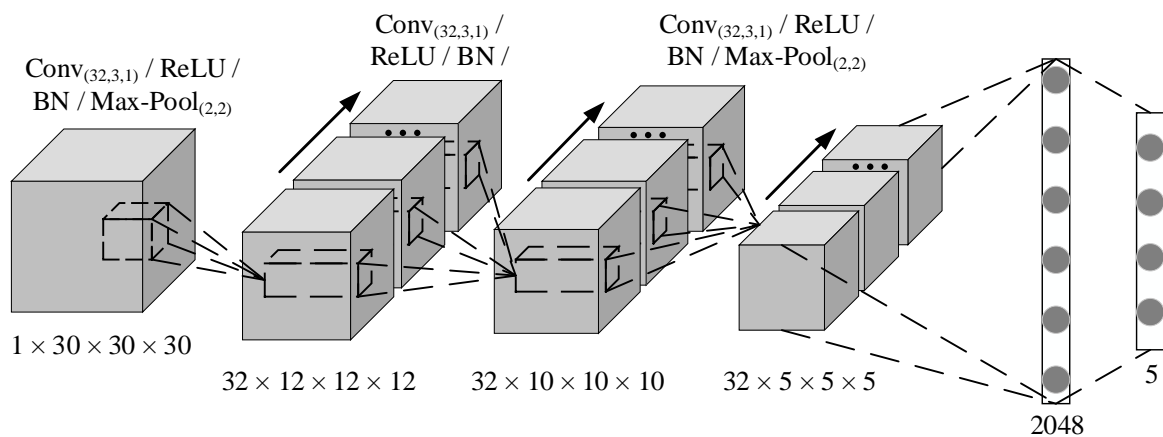


Figure 3-9 The network architecture of the TCVN model.

### 3.2.2. Floor Detection (Deep Learning)

One of the important features in MLHO tasks is a humanoid robot should know in what type of floor it performs the dragging task. In this section, the object instance segmentation technique is employed for the floor detection from the robot camera. Therefore, this algorithm will identify partitioning pixels image into a segmentation mask of floor area. As a result, the segmented pixels provide the information in which type of ground it performs the dragging task. Figure 3-10 shows three different types of floors where the humanoid robot will be evaluated on the dragging task.

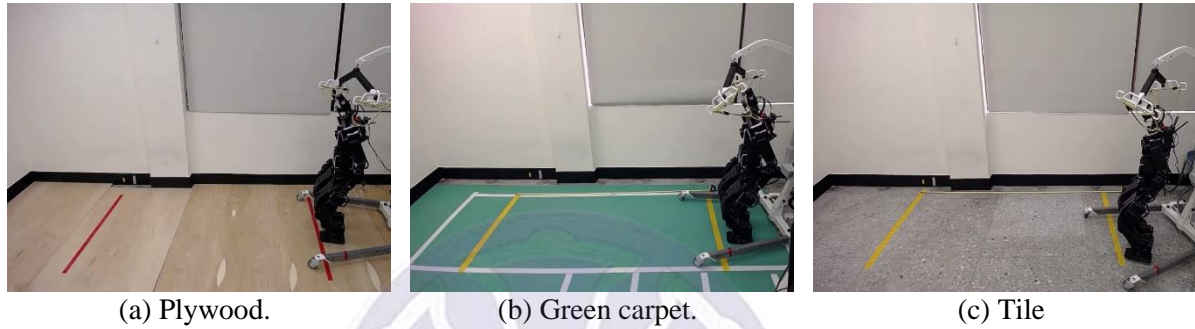


Figure 3-10 Types of floors used in this experiment.

The state-of-the-art instances segmentation was introduced by [44]. In [44], the authors introduced Mask-RCNN that was built with focuses on performance. However, these instance segmentations are accurate but it only runs on 5 frames per sec (fps) on modern computer hardware. The main reason, because they force using expensive re-pooling operation in the ROI-align. Also, the Mask-RCNN model used a two-stage detector, which means the computational in the model happened in sequentially.

Later on [45], the authors introduced a real-time instance segmentation called YOLACT (You Only Look at Coefficients) that beats state-of-the-art instance segmentation in terms of speed. They used a one-stage detector and produce two parallel parts solution to split mask computation. First, they create a set of “prototype” mask for the whole image. Second, linearly combine those prototypes using coefficients from the prediction head.

As shown in Figure 3-11, the design of YOLACT network architectures is presented in detail. First, the model uses the standard Residual Network (ResNet) with

Feature Pyramid Net (FPN) as the backbone network. Then, FCN (“Proto Net”) is attached to the largest FPN layers to produce this whole prototypes masks. Second, in parallel, the standard “Prediction Head” predicts the linear combination coefficients for each anchor box. Finally, the models do some minimal post-processing (crop and threshold) to obtain the final mask.

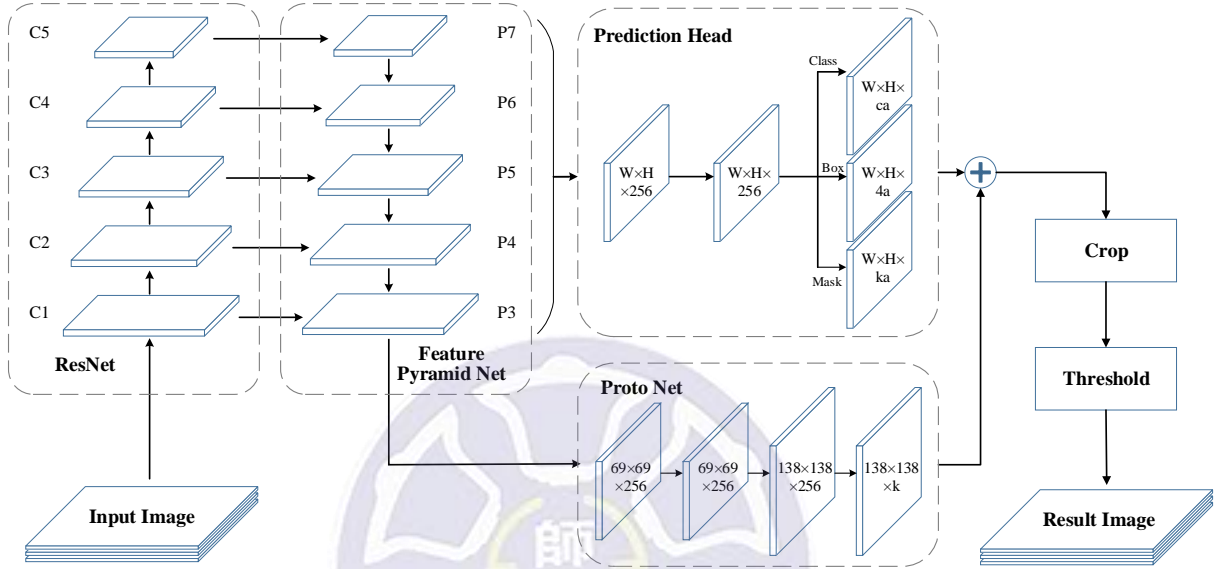


Figure 3-11 YOLACT network architecture [46].

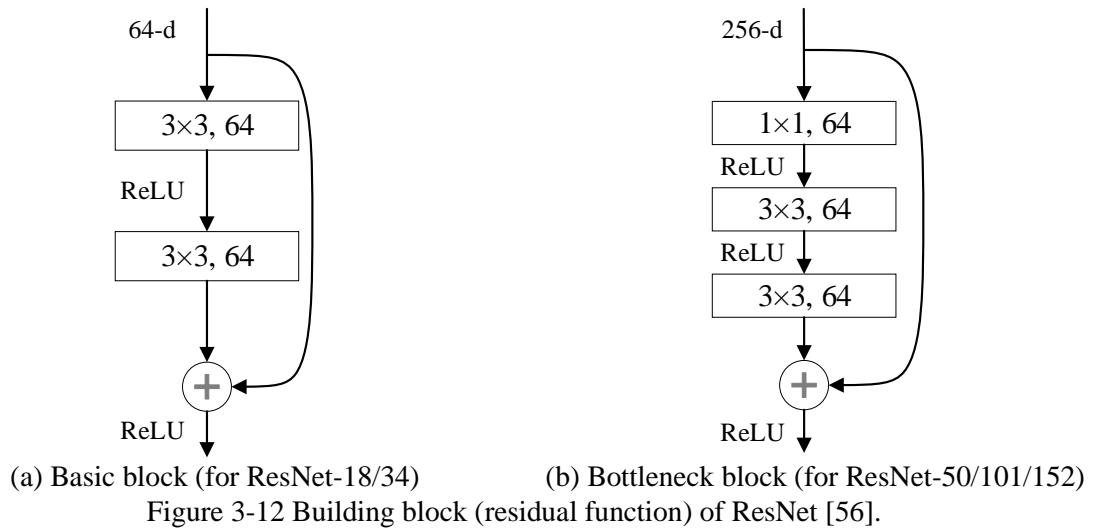
For training the mask branch, a pixel-wise loss is applied only on the final assembled mass. Thus, the prototypes and linear combination coefficients only get downstream supervision from the mask loss. This means the combination is not constraining of any semantic. Therefore, the leads to the prototypes taking on some various translation variants in a fully convolutional network.

Furthermore, as stated in the original of ResNet [56], the authors validated deeper residual network lead to lower loss value that improved the accuracy. Therefore, in [46], the authors use ResNet-101 as the default backbone in the YOLACT model. They trained the model with a base size image  $550 \times 550$  on advanced Microsoft-Common Object in Context (COCO) dataset. Furthermore, their method achieved results above 30 fps on the COCO dataset by using the ultimate graphic processing unit (GPU) NVIDIA Titan XP.

As shown in Figure 3-12, the ResNet building blocks are categorized into two types. (i) Figure 3-12(a) illustrated the two sequential convolutional layers in the basic block for building ResNet-18 and 34 block function. (ii) Figure 3-12(b) shown a deeper network of three sequential convolutional layers to build a bottleneck block function of Resnet-50, 101, and 152.

Table 3-3 The adopted ResNet architecture and number of parameters [56].

layer name	output size	18-layer	34-layer	50-layer	101-layer
<b>conv1</b>	112×112	7 × 7, 64, stride 2			
<b>conv2_x</b>	56×56	3 × 3 max pool, stride 2			
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
<b>conv3_x</b>	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
<b>conv4_x</b>	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$
<b>conv5_x</b>	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax			
<b>FLOPs</b>		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$
<b>Number of Parameters</b>		11.176.512	21.284.672	23.508.032	42.500.160



Generally, the original YOLACT model was tested to predict 80 class categories in the COCO dataset. However, the objective in this floor detection was simplified only to detect three types of objects, which is much simpler compared to the COCO dataset. Therefore, in this section proposed a new simple model called Tiny-YOLACT. This proposed model is fully based on YOLACT architecture with modification only on the backbone network. The main advantage of the Tiny-YOLACT model that it uses ResNet-18 & ResNet-34 which has a smaller number of weight parameters. As a result, this model can also run on a moderate level NVIDIA GPU. The details of each network architecture of ResNet and total numbers of parameters presented in Table 3-3.

### 3.3. Robot Motion Control

In this section, the development of robot motion controlling as shown in Figure 3-5 is centralized in the motion planner. Wherefore, it consists of a grasping motion and the main walking control itself. However, both will be described below.

#### 3.3.1. Object Grasping

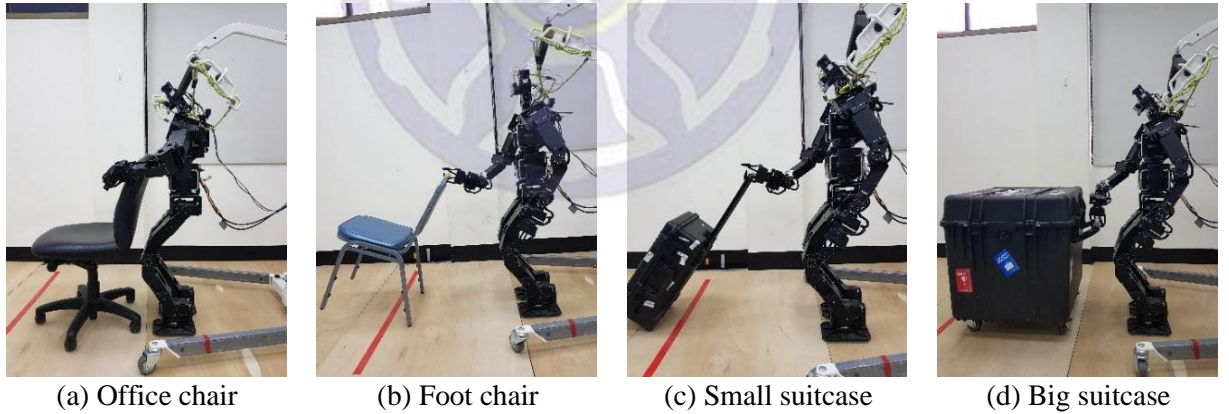


Figure 3-13 Sample pre-recorded motion for grasping different types of objects.

The first step to drag an object is to grasp it properly. There are many researched focuses on manipulation grasping different types of objects. However, the objective of this work is to focus on designing a balanced walking on BHR to solve the MLHO problem. Therefore, to grasp the experimental object, whole-body motion capture is

employed to record the robot gesture. Moreover, it is natural for a humanoid robot with a design similar to human structure, for imitating human motion on grasping an object. As shown in Figure 3-13, the robot postures on different objects were recorded. So then, those motions will be used when the robot has identified the object from the output of 3D object classifications.

### 3.3.2. Walking Control

The THORMANG-Wolf robot used a closed-loop walking engine based on Zero Moment Point (ZMP) walking control [57]. The closed-loop walking engine of biped humanoid robots can be illustrated like a cart-table model as shown in Figure 3-14. First, a cart with mass  $M$  running on a lightweight table with mass is insignificantly small. Then, the size of the table foot is too narrow to stabilize the cart on the table edge. Therefore, the table-cart models show by keeping cart runs on certain accelerations it can keep a spontaneous balance of the table. Same like biped humanoid robot walking on the ground, the position of ZMP is given as:

$$p = x - \frac{z_c}{g} \ddot{x} \quad (3-2)$$

Where  $x$  is the moving coordinate,  $z_c$  is the constant height of the Center of Gravity (CoG),  $g$  is the gravity and  $\ddot{x}$  is the acceleration. Walking pattern generation based on a cart-table model takes the trajectory of CoM as input and resulting ZMP as output. Therefore, ZMP based walking pattern generation is an inverse form of a 3D linear inverted pendulum. Regarding a cart-table model as a dynamical system, the CoM motion starts before the changes of the ZMP. This means the cart must move before the change of input in the system.



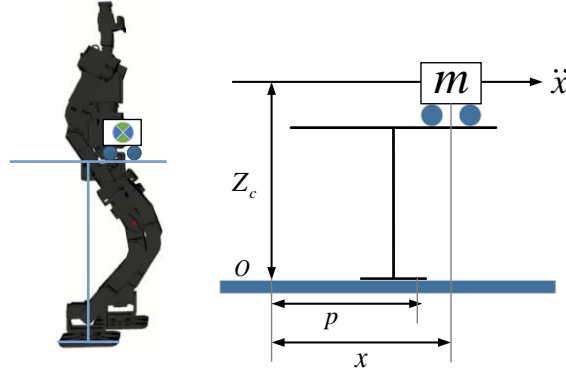


Figure 3-14 Cart-table model

**Note:** Robot walking behavior is close to a cart moving on a massless table. The condition of the moving cart decides the center of pressure operating from the floor (the cart is changing ZMP).

In [58], the authors used cart-table model as a dynamical system and designed a pattern generator, which is called preview-control as shown in Figure 3-15.

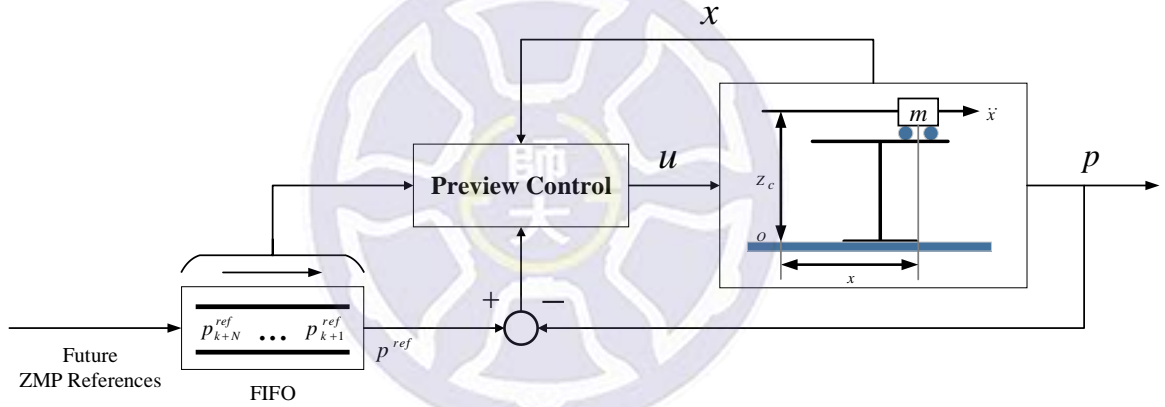


Figure 3-15 Walking pattern generation based on preview control

Figure 3-15 shows a block diagram for the walking pattern generator based on the ZMP preview control. First, we regard the reference ZMP and the cart status calculation in FIFO buffer as the input of the preview control system. Then, the control vector  $u_k$  can be obtained by Equation (3-3) based on the ZMP reference and the cart current state. While according to the system equation, the position on  $x$  axis and  $y$  axis of CoM is the result of pattern generation. With this information, the CoM trajectory is taken from the position of the cart.

$$u_k = -K_s \sum_{i=0}^k (p_i^{ref} - p_j) - K_x x_k + \sum_{j=1}^n G_j p_{p_k}^{ref} \quad (3-3)$$

Where  $K_s, K_x, G_j$  are the gains for preview controller,  $p^{ref}$  is the ZMP reference,  $p_j$  is the ZMP output,  $x_k$  is the state of the cart including position, velocity, and acceleration.

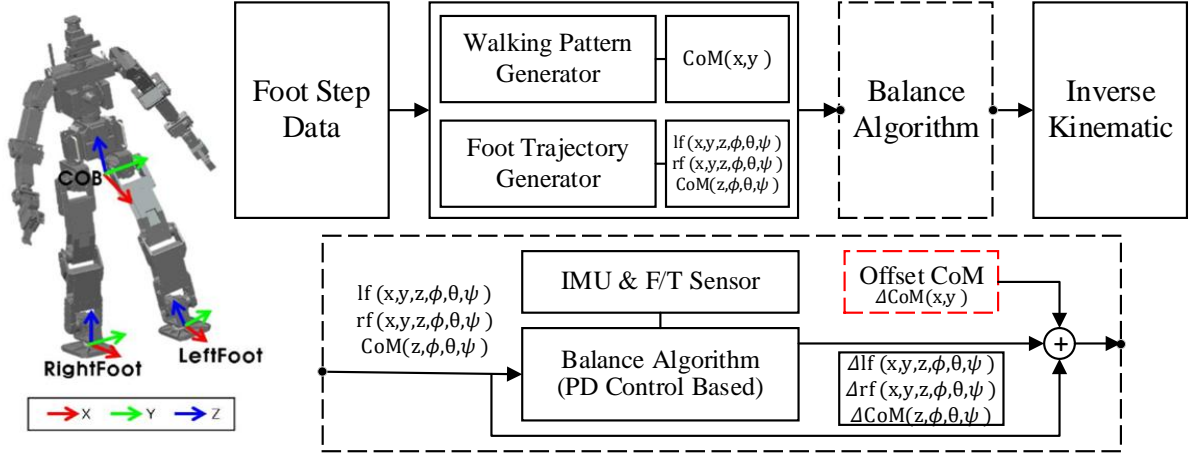


Figure 3-16 Walking gait pattern generation process.

**Note:** Red block is the parameter to be controlled by our proposed methods.

The process of walking gait generation processes of THORMANG-Wolf robot shown in Figure 3-16 is described as follows. First, the footstep data takes an input of step-number, step-time, step-length, side-step-length, and step-angle-degree to generates footstep data array. Then, the footstep data planning is used to produce a walking pattern and foot trajectory. Therefore, as described in Figure 3-15, the walking pattern generation generates the CoM trajectory in  $x, y$  axis for the ZMP tracking control system. Meanwhile, on the foot trajectory generator, we use a sigmoid pattern to generate the gait pattern and obtain trajectories of the CoM and ankles. As a result, it generates a smooth gait pattern when lifting a leg during the walking phase. Next, we use the balance algorithm to adjust the walking trajectory for keeping the robot stable during walking.

The balancing algorithm calculates the difference between the reference and actual pose of the robot based on the robot state. Therefore, an Inertial Measurement Unit (IMU) sensor with Force and Torque (F/T) sensor is used as feedback to read the robot state. Moreover, the additional offset of CoM in  $x, y$  axis will be used for the behavior control on the deep reinforcement learning. As a result, the balance algorithm and offset CoM are combined. Thus, the equation become

$$\begin{cases} lf_g = lf + \Delta lf \\ rf_g = rf + \Delta rf \\ CoM_g = CoM + \Delta CoM \end{cases} \quad (3-4)$$

Where  $lf_g, rf_g, CoM_g$  are defined as goal coordinate of the left foot, right foot, and Centre of Mass,  $\Delta$  is the difference offset resulting from the balance algorithm. Finally, we utilized an inverse kinematics solver to calculate all the angles of joints on the legs from the result of Equation (3-4).

### 3.4. Robot Behavior Control

As illustrated in Figure 3-5, the THORMANG-Wolf robot software system is comprised of a behavior-controlling level. At this level, we proposed a Deep Q-Learning to Learn the Centre of Body (DQL-COB) algorithm for the behavior control of THORMANG-Wolf robot on this dragging task. It is worth mentioning that the proposed DQL-COB is part of the proposed hierarchical DL algorithm design in terms of the DRL algorithm on the walking balance control policy. Therefore, the output from the DQL-COB algorithm resulting in a Centre of Body in X-Axis ( $CoB_x$ ) offset value, as the parameter to the robot walking module. The integration of the walking module with the DQL-COB algorithm is illustrated in Figure 3-17.

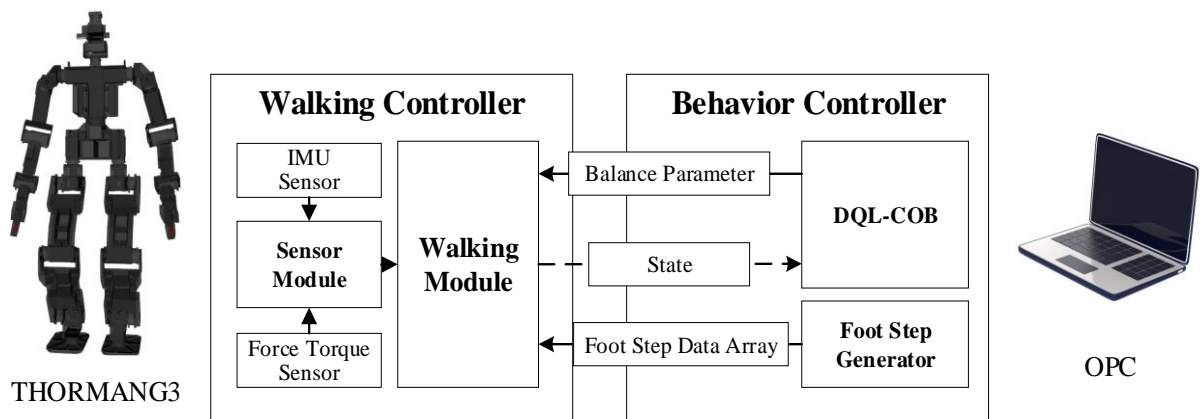


Figure 3-17 Integration walking module with the DQL-COB algorithm.

Based on this figure, the flow process of unified systems is divided into two central control and described as follows. First, on behavior control where the array footstep data generated from the footstep generator will be given to walking module.

Then, walking control will generate the walking gait pattern and also activate the balance control (see Figure 3-16). Next, the robot starts to walk derived by walking module. Finally, the DQL-COB algorithm will provide a balance parameter based on the closed-loop feedback state of the robot continuously and in real-time.

One of the objectives of this thesis problem is to have a balance backward walking meanwhile dragging large size and heavy objects. Therefore, the initial step for applying the RL-based algorithm is to create or set up an environment for the agent to interact. Hence, to avoid costly experiments in terms of time and physical training, the environment and training process was done in the Gazebo<sup>8</sup> simulator that also integrated with ROS. As shown in Figure 2-12, MDP is the mathematical formulation of the RL problems. Also, Markov's property represents the current state completely characterizes the state of the world. Therefore, MDP also defined by a tuple object  $(S, A, R, P)$ .

### 3.4.1. DQL-COB Algorithm Design

In the rest of this section, the proposed DQL-COB algorithm including five phases to solve the MLHO problems are described respectively.

#### 3.4.1.1. States space

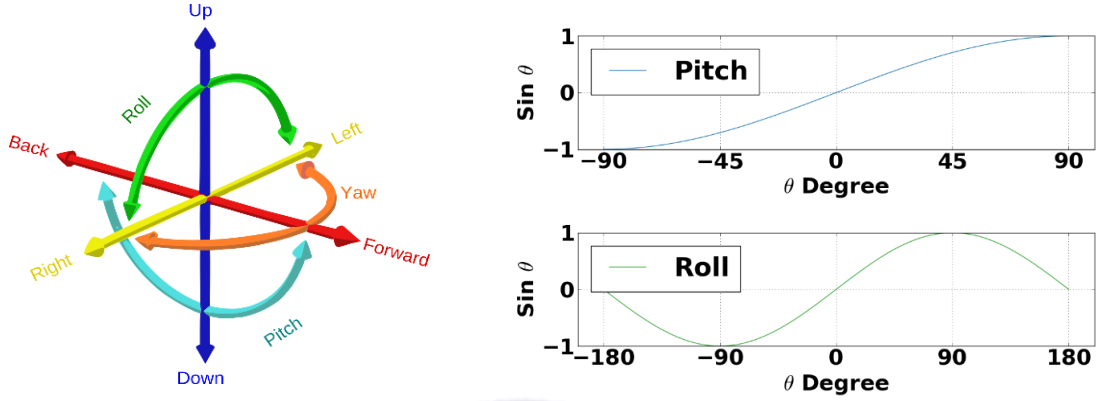
The first tuple object of MDP is the set of possible states  $S$  in an environment. Since the aim is to have balance dragging actions. Therefore, utilizing data from the Inertial Measurement Unit (IMU) with Force and Torque (F/T) sensors for reading the robot state whether robots in stable condition or not. Generally speaking, the IMU sensor has a characteristic of angular value output to represent each of the axes. The IMU sensors in the THORMANG-Wolf robot illustrated in Figure 3-18 (a) consists of 6 DOF. However, in this environment, only the pitch and roll axis were selected as an important feature to provide into the agent.

It is well-known that to normalized the data before feed data into a deep neural network. Therefore, in the NN literature, normalizing also often refers to rescaling by

---

<sup>8</sup> <http://gazebosim.org/>

the minimum and range of the vector, to make all elements lie between 0 and 1. For this purpose, IMU pitch and roll angles are normalized to comply with that. It can also achieve an efficient training process by having a faster training process (due to the mathematical structure of calculation in a neural network).



(a) 6-DoF axis IMU sensor (b) Normalize theta of IMU Pitch & Roll using  $\sin \theta$ .

Figure 3-18 IMU sensor as a state of the environment.

In contrast to the IMU sensor, the F/T sensor located in the robot's ankle also provide the robot state. It acts as a perceiving robot's foot state whether on the ground or air. Particularly, the torque sensor gives output in vector product of the force magnitude and the perpendicular distance of the action force line (see Figure 3-19). Due to the complexity number given by torque sensors, those values are preprocessed using the binarization method before feed into the deep neural network. The binarization process is explained in Algorithm 1.

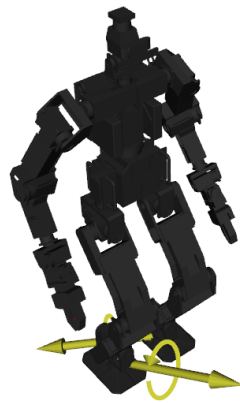


Figure 3-19 Torque vector on both feet.

The algorithm 1 details are explained in the following. First, initializing constant threshold value for each foot. Due to torque values are in a three-dimension vector, therefore obtaining the length of the torque vector was done by calculating Euclidean distance towards the origin axis (0,0,0). Finally, after vector distance is acquired, thresholding the values for each foot to binarized it: 0 if foot on the air and 1 if a foot on the ground. These processes are repeated in a loop for each foot.

---

**Algorithm 1:** Binarization torque sensor

---

1. Initialize threshold  $T$  for each foot  $M$
  2. **For**  $j = 1, M$  **do**
  3. Calculate Euclidean distance  $d_j = \sqrt{\sum_{i=1}^3 (p_i - 0)^2}$
  4. Set  $y_j = \begin{cases} 0 & \text{if } d_j > T_j \\ 1 & \text{otherwise} \end{cases}$
  5. **End For**
- 

So, the total states in this environment are 4 states. Given by normalized IMU pitch and roll, also the binarized torque value of each foot. Which is the state  $S$  can be denoted as  $(Pitch, Roll, Lfoot, Rfoot)$ .

### 3.4.1.2. Actions space

The second tuple object of MDP is the set of possible actions  $a$  in an environment. As illustrated in Figure 3-20, the action in this DQN approached is simplified to offsetting the parameter of the balance control algorithm in the Center of Body in  $X$  axis ( $COB_x$ ). The main concept to achieve stable walking meanwhile receiving disturbance is by keeping ZMP point in the support polygon. Therefore, the offset of the Center of Body in the x-axis is the most critical point to achieve that. So, in total there are 3 actions to be learned by the DQN agent. Those actions are for increment, decrement, and do nothing towards the offset parameter  $COB_x$ .



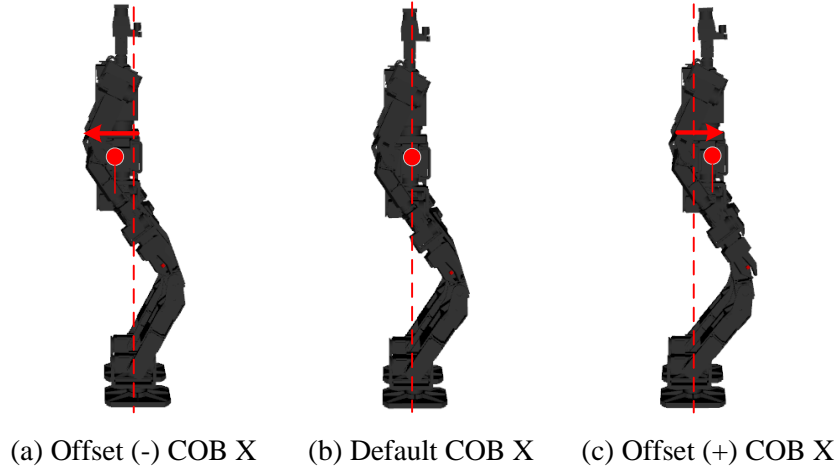


Figure 3-20 Action offset on COB X.

### 3.4.1.3. Rewards function

The third tuple object of MDP is the value of the reward  $R$  from distribution given by (state, action) pair. In this environment, reward functions were calculated from two main factors. Figure 3-21 illustrates those factors that are from the IMU pitch sensor and traveled distance toward the goal target. The details of reward functions are presented in Algorithm 2.

First, rewards based on the balance factor, the reward is calculated on how a robot could maintain its pitch angles during the dragging task. Therefore, the robot's pitch angle denoted as  $\theta$ , if the  $\theta$  degree within the threshold range, means robot in stable condition then  $r_1$  is 1. Conversely, when  $\theta$  it is out of the stable threshold range then  $r_1$  it will less than 1, it calculated from the division of the error differences of the threshold  $t$ . Consequently, if the robot falls then the agent will get a -1 reward or called punishment, it also terminates the step.

Second, the reward is based on accomplished walk distance. Additionally, in this study, the target distance that the robot requires to walk is to drag an object for 2-meter and walking backward. Therefore, another 2D Euclidean distance utilized to measured walk distance for reward  $r_2$  calculation. It means, the closer to the finish line, the distance reward  $r_2$  will be close to 1. Furthermore, when robots finished dragging or

successfully walk in 2-meter the  $r_2$  will also be 1. Overall, the total reward function  $r$  is obtained  $r_1 \cdot r_2$ .

---

**Algorithm 2:** Reward function

---

- |   |   |
|---|---|
| 1. Initialize threshold $t$ for a stable state  | 9. Initialize $f$ for finish distance     |
| 2. <b>if</b> $t$ is fall <b>then</b>            | 10. <b>if</b> $d$ is finished <b>then</b> |
| 3. $r_1 = -1$                                   | 11. $r_2 = 1$                             |
| 4. <b>else if</b> $t > \theta > -t$ <b>then</b> | 12. <b>else</b>                           |
| 5. $r_1 = 1$                                    | 13. $r_2 = \frac{1}{f - d}$               |
| 6. <b>else</b>                                  | 14. <b>end if</b>                         |
| 7. $r_1 = \frac{1}{\theta - t}$                 |   |
| 8. <b>end if</b>                                | 15. $r = r_1 \cdot r_2$                   |
- 

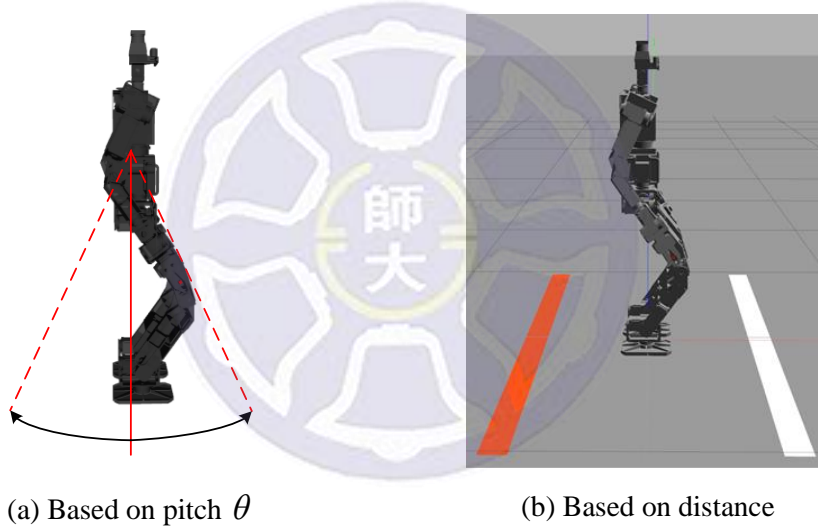


Figure 3-21 Reward based on robot pitch state and finished distance.

Finally, the total cumulative reward will be obtained from cumulating this reward  $r_t$  in each step. Then, the RL algorithm agent tries to find an optimal policy to receive maximum rewards by keeping robot stables in the desired pitch angle and walk-in backward direction as close as to finish line target.

#### 3.4.1.4. Deep Q-Network

The main objective of the RL algorithm is to find an optimal policy  $\pi^*$  that maximizes the expected sum of rewards. Therefore, an agent acts in the environment

and receives a reward, where the optimal Q-value function  $Q^*$  is the maximum expected cumulative reward achievable from a given (state, action) pair of the following policy.

$$Q^*(s, a) = E \left[ \sum_{t \geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi \right] \quad (3-5)$$

Solving for that, a Deep Q-Learning uses a deep neural network as a function approximator to estimate the action-value function. Therefore, the Q-value function is determined by those neural network data parameter  $\theta$  weights. Given this function approximation, finding the optimal policy it requires to calculate the Q-function that satisfies the Bellman equation.

$$Q^*(s, a) = E_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right] \quad (3-6)$$

DQN algorithm enforces the Bellman equation to have neural network approximating Q function. Therefore, it can be accomplished by training the network where the loss function is going to minimize the error of the Bellman equation. However, the forward pass of the neural network is given by the following equation.

$$L_i(\theta_i) = E_{s, a \sim \rho(\cdot)} \left[ (y_i - Q(s, a; \theta_i))^2 \right] \quad (3-7)$$

Where,

$$y_i = E_{s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a'; \theta_{i-1}) \mid s, a \right] \quad (3-8)$$

and the backward pass (gradient updates concerning Q-function parameters)

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s, a \sim \rho(\cdot); s' \sim \mathcal{E}} \left[ r + \gamma \max_{a'} Q^*(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \nabla_{\theta_i} Q(s, a; \theta_i) \right] \quad (3-9)$$

Furthermore, the proposed deep neural network architecture for estimating Q-function is shown in Figure 3-22. In this situation, the goal is directly predicting Q-

function over a network (concept: doing regression towards Q-value). Therefore, the proposed model consists of 3 fully connected layers. The first layer or also the input layer receives the state from the robot. Then, the second and third layer consists of 128 and 64 neurons with ReLU activation respectively. Finally, the last layer has a vector of output, calculates Q-value corresponding to each action.

Table 3-4 Details of the Q-Network architecture.

Layer type	Neurons	Number of parameters
Input	4	-
1 <sup>st</sup> Hidden Layer	128	640
ReLU	128	-
2 <sup>nd</sup> Hidden Layer	64	8256
ReLU	64	-
Output Layer	3	195

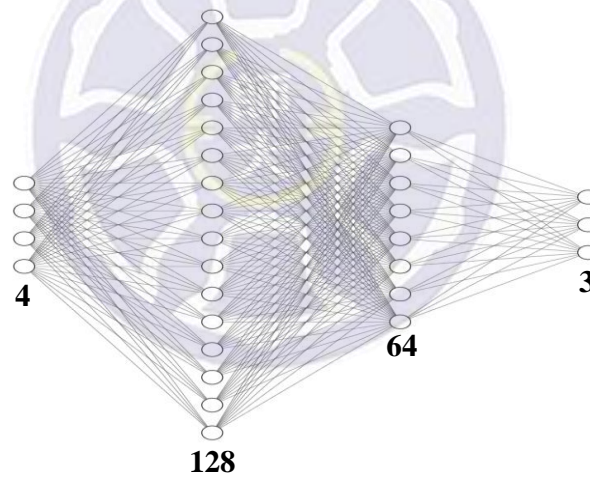


Figure 3-22 Hierarchical Q-Network architecture.

Since the proposed network has 3 actions, there is 3 scalar value (Q-values) given by  $Q(s_t, a_1), Q(s_t, a_2), Q(s_t, a_3)$ . By using neural network structure, efficiently a single feedforward pass can compute Q-values for all actions from the current state. Overall, the details of the proposed model are presented in Table 3-4.

### 3.4.1.5. Training Process

As described before, based on loss function on Eq (3-7), the aim is to iteratively make Q-values as close to target values. However, learning from batches of consecutive

samples is problematic. In [49], the authors evaluated learning directly from consecutive samples is inefficient, it has highly correlated data. Because current Q-network parameters determine the next training samples, it leads to bad feedback loops. Therefore, addressing that problem by using experience replay. Storing the transition of history states  $(s_t, a_t, r_t, s_{t+1})$  in replay memory and minibatch randomizing the samples breaks these correlations and reduces the variance of the updates.

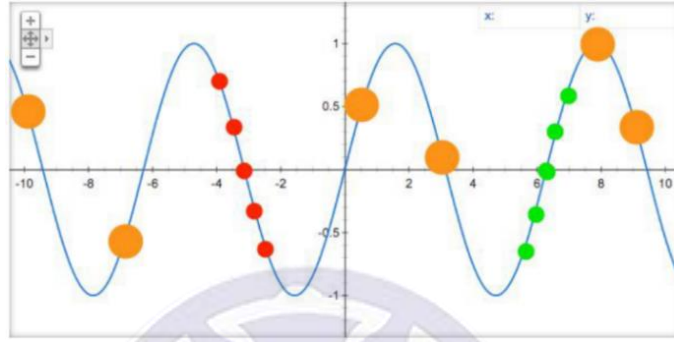


Figure 3-23 Experience replay illustration on training data.

As shown in Figure 3-23, red and green dots present the correlated data. On the other hand, oranges dot breaks the correlation by randomized sample the experience. During training, the Q-learning uses the Eq (3-10) as the loss function with sample experience replay memory on the forward pass.

$$L_i(\theta_i) = E_{(s,a,r,s')} \sim U(D) \left[ \left( r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right] \quad (3-10)$$

Another problem listed on [49], RL algorithm is considered non-stationary distributions. It is unstable or even divergent when a nonlinear function approximator (e.g. neural network) is used to represent the action-value (Q-function). Therefore, using an iterative update to adjusts the Q-values towards target values within periodically update, it can reduce correlations with the target. This solution also illustrated in Figure 3-24.

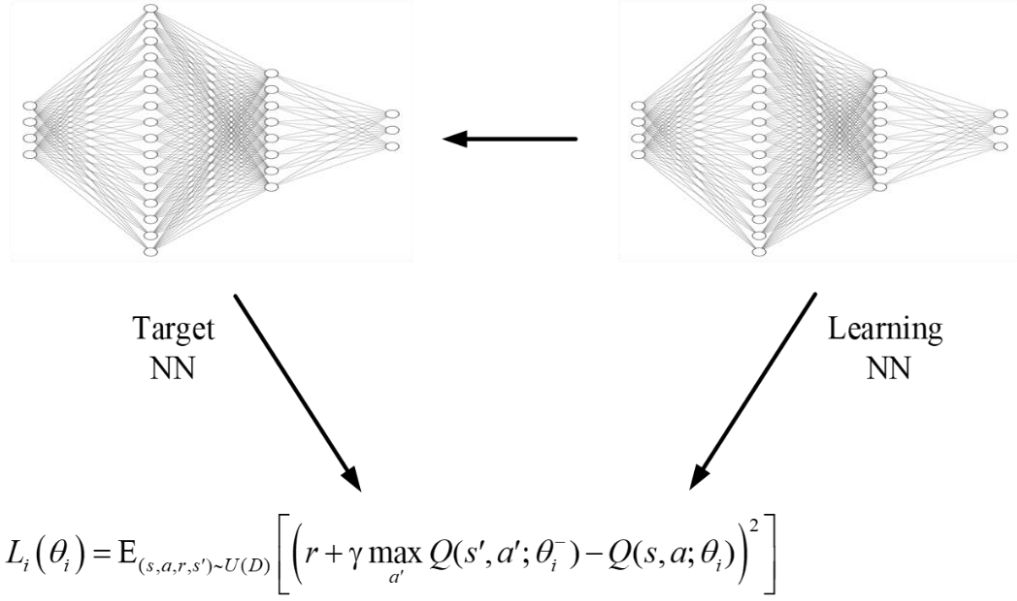


Figure 3-24 Solution to non-stationary target DQN.

As shown in Figure 3-24, target values are calculated by using a target network, which is the duplicate network of the learning network. Whereas, the weight parameters of the target net using an older set of parameters and periodically update. It made effects on targets  $y_i$ , which make oscillations and divergence more unlikely. During training,  $\epsilon$  – greedy algorithm shown in [Eq (3-11)] is used for the exploration and exploitation problem.

$$a = \begin{cases} \text{optimal } a^* & 1 - \epsilon \\ \text{random } \epsilon \end{cases} \quad (3-11)$$

Moreover, in [49], the authors also evaluated clipping the error from the update  $r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)$  to be between -1 and 1. The benefit of error clipping leads to stability training. Since the absolute value loss function  $|x|$  has a derivative of -1 for all negative values and a derivative of 1 for all positive values. Therefore, for error values outside of the range (-1,1), this clipping error is similar to using an absolute value loss function. Identically, the approach in this dragging task is similar to the approached done by [49]. Whereas, the whole training process explained above is illustrated in Figure 3-25.



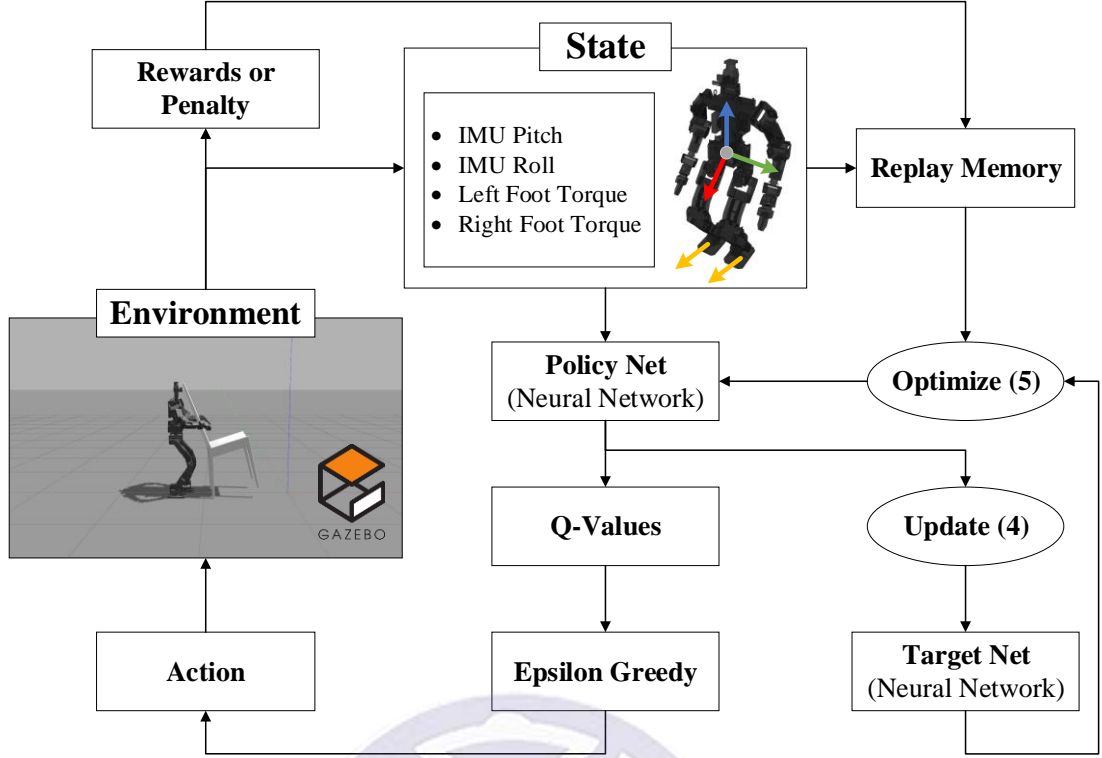


Figure 3-25 Block diagram of Deep Q-Network on ROS Gazebo simulator.

Putting all those together, the block diagram in Figure 3-25 also represented in Algorithm 3. The details of those are described briefly. First, initialize replay memory  $D$  with some capacity  $N$  and also initialize Q-Network with some random initialize weight  $\theta$ . Then, start to train the dragging task with the episode  $M$ . This loop for training the whole episode. Afterward, initializing the state  $S$  by resetting the simulator and acquired robot initial state at the beginning of each episode. Note that, the state  $S$  is the preprocessed robot state ( $Pitch, Roll, Lfoot, Rfoot$ ).

During, each time step  $t$  of the training, it will generate a small probability for select random action. In this process, the algorithm requires to have sufficient exploration to sample enough state space. Otherwise, it acts base on greedy action from the current policy. Therefore, most of the time it will take greedy action based on the best knowledge of the type of action and desired state.

The next step, acting  $a_t$ , observe reward  $r_t$ , and the next state  $s_{t+1}$ . Then, store transition  $(s_t, a_t, r_{t+1}, s_{t+1})$  in replay memory  $D$ . Currently, it will train the network

meanwhile sample random mini-batch of transition  $(s,a,r,s')$  from  $D$  and perform gradient descent step. So, these are full training loop and continuously drag the object in the Gazebo simulator and also sampling minibatch using experience replay to update Q-network weight  $\theta$ .

---

**Algorithm 3:** Deep Q-Learning with Experience Replay

---

1. Initialize THORMANG-Wolf PPC & MPC modules
  2. Initialize replay memory  $D$  to capacity  $N$
  3. Initialize action-value function  $Q$  with random weights  $\theta$
  4. Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$
  5. **For** episode = 1,  $M$  **do**
  6.   Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$
  7.   **For**  $t=1, T$  **do**
  8.     With probability  $\varepsilon$  select a random action  $a_t$
  9.     Otherwise select  $a_t = \arg \max_a Q(\phi(s_t), a; \theta)$
  10.    Execute action  $a_t$  in the emulator and observe reward  $r_t$  and state  $s_{t+1}$
  11.    Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$
  12.    Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$
  13.    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$
  14.    Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
  15.    Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  for the network parameters  $\theta$
  16.    Every  $C$  step reset  $\hat{Q} = Q$
  17.   **End For**
  18. **End For**
-

## Chapter 4: Experimental Result

As mentioned in the introduction, the objective of this thesis was divided into robot vision and behavior methods. Therefore, in this chapter, to prove the accuracy of the proposed methods first, we describe the experimental setup. Then we conducted and evaluated proposed DL algorithms into robot vision and behavior sections. To this end, in the robot vision, first, we show experiments on 3D object classification, where the accuracy of the proposed TCVN model are compared to classified 4 types of a 3D object. Second, we show experiments of the floor detection on three different surfaces to show the performance of the proposed lightweight Tiny-YOLACT (real-time instance segmentation model).

Accordingly, in the robot behavior, first, we show an evaluation of the simulation result, where the training result of the proposed DQL-COB algorithm is compared in two scenarios. Hence, in one scenario we use IMU along with foot Pressure (F/T) sensor, in the second scenario we just use IMU data as learning algorithm input. Second, we show an empirical evaluation of the proposed DQL-COB algorithm on the THORMANG-WOLF robot. In this experiment, we demonstrate for the first time, a novel implementation of the deep reinforcement learning method that utilizing training results in simulation to a real humanoid robot for solving the MLHO problems.

As far as we know to the best of our knowledge, the DQL-COB algorithm solves for the first time in this MLHO problem. Therefore, we evaluated two types of experiments on three types of surfaces with eight different objects. In these experiments, there are two schemes of data inputs (the same as the simulation) are compared. Finally, it is worth mentioning that the robustness of the walking controller is not being analyzed because designing walking control is not the focus of this thesis.

## 4.1. Experimental Setup

To evaluate the proposed method, we implemented, conducted, and validated different experiments on the THORMANG-Wolf robot (section 3.1). The hierarchical method consists of three proposed algorithms including the TCVN model, Tiny-YOLACT model, and DQL-COB algorithm. The entire code was implemented in the Python programming language. In this regard, the standard python open-source neural-network libraries Keras<sup>9</sup> and machine learning libraries PyTorch<sup>10</sup> were used to implement the deep learning models on the proposed methods.

Our experiments are conducted on the common objects and surfaces available in human daily life. Table 4-1 and Table 4-5 are presented the experimental surfaces and objects respectively.

Table 4-1 Experimental surfaces.

No.	Surface
1.	Plywood
2.	Green Carpet
3.	Tile

Table 4-2 Experimental objects.

No.	Object Type	Weight
1.	Default (No Object)	0 Kg
2.	Office Chair	12.3 Kg
3.	Office Chair with Load	33.5 Kg
4.	Foot Chair	1.3 Kg
5.	Small suitcase <sup>11</sup>	6.2 Kg
6.	Small suitcase with Load	26.2 Kg
7.	Big Suitcase <sup>12</sup>	18.6 Kg
8.	Big Suitcase with Human	84.6 Kg

As shown in Table 4-2, the overall 8 objects are a set of collections from a few similar objects with different loads that are invisible in 3D object visual. Therefore, in the conducted experiments for the 3D object classification, there are only 4 objects selected to represent those 8 objects.

<sup>9</sup> <https://keras.io/>

<sup>10</sup> <https://pytorch.org/>

<sup>11</sup> <https://www.pelican.com/us/en/product/cases/carry-on-case/protector/1510>

<sup>12</sup> <https://www.pelican.com/us/en/product/cases/cube-case/protector/0370>

Moreover, in the conducted walking for the dragging task, the robot is required to walk in a straight-line trajectory with a backward direction for 2 meters. Therefore, the configurations of the foot-step generator for walking are shown in Table 4-3.

Table 4-3 Foot-steps parameter.

Parameter	Value
Step Number	12
Step Time	1.0 s
Step Length	0.1 m
Side Step Length	0.05 m
Step angle degree	5 deg

Generally, the whole training process of the three proposed algorithms are trained on a single computer. The details of the deep-learning computer are shown in Table 4-5. Therefore, the proposed approaches are tested on the OPC (a laptop to operate the THORMANG-Wolf robot). In this situation, OPC is used for running the inference of those trained models and integrating it into the THORMANG-Wolf robot. Table 4-5 presented the details of laptop hardware (OPC) used in this whole evaluation test.

Table 4-4 Deep-learning computer hardware specifications.

Name	Information
<b>Processor</b>	Intel Core i7-8700H (8th Gen) / 3.2 Ghz
<b>Graphic</b>	NVIDIA GeForce GTX 1080Ti / 11 GB
<b>Memory / Type</b>	32 GB / DDR4 / 2666 Mhz
<b>Storage</b>	240 GB 2.5in SATA SSD

Table 4-5 OPC (laptop) hardware specifications.

Name	Information
<b>Processor</b>	Intel Core i7-8750H (8th Gen) / 2.2 Ghz
<b>Graphic</b>	NVIDIA GeForce GTX 1060 / 6 GB
<b>Memory / Type</b>	16 GB / DDR4 / 2666 Mhz
<b>Storage</b>	256 GB NVMe PCIe Gen3

To evaluate the performance of the 3D object classifier of the TCVN model, the statistical table called confusion matrix is utilized to draw the error of prediction in the test dataset, in which the true values are known. Moreover, to evaluate the Tiny-YOLACT model performance on floor detection, the mean Average Precision (mAP)

value is used as the model benchmark during training. Additionally, frame per sec (fps) rate is utilized to evaluate the speed of the models in terms of video processing times. Also, to evaluate the robustness of the DQL-COB algorithm, the error in this experiment was based on robot traveled distance and stability. Therefore, error values were calculated by Euclidean distance of robot final position and related to the starting position tracked by a global camera with the ArUco marker attached on top of the robot.

## 4.2. Experimental Result for Robot Vision

The evaluation results of the robot vision-based deep learning methods are described in this section. Moreover, all data used in this evaluation are captured manually using the THORMANG-Wolf robot's webcam and LiDAR scanner.

### 4.2.1. 3D Object Classification Result

In this subsection, collected 1200 data of 4 objects that are illustrated in Figure 4-1. Each data is labeled according to the object's name. Then, the model's performance was compared using VoxNet [52], V-CNN1 [53], and TCVN on the collected dataset. Likewise, an object classification task, the last output layer of VoxNet and V-CNN1 were modified to follow the number of objects in this dataset.

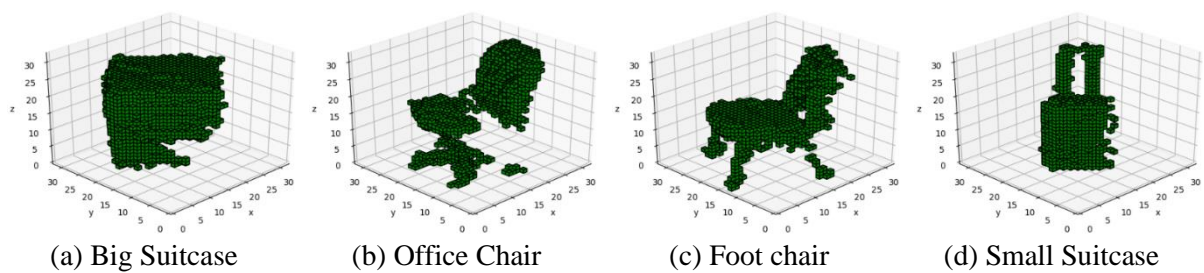


Figure 4-1 The 4 types of the 3D object after voxel grid filter.

Figure 4-2 presented the evaluation of each model during training. The key performance to evaluate these models are based on loss and accuracy value. Therefore, an interpretation of loss value is based on how bad the model is doing in these two sets prediction after every epoch iteration. Hence, accuracy is a number indicating how good



was the model's predicted on the validation test compared to the ground-truth data. Whereas, a key factor for measuring rank of loss value is by the lowest value, so the model with the lowest value rating, means that the best it is. Conversely, the highest value category is used for evaluating the rating of the model's accuracy. As a result, models with the highest accuracy means the better it is.

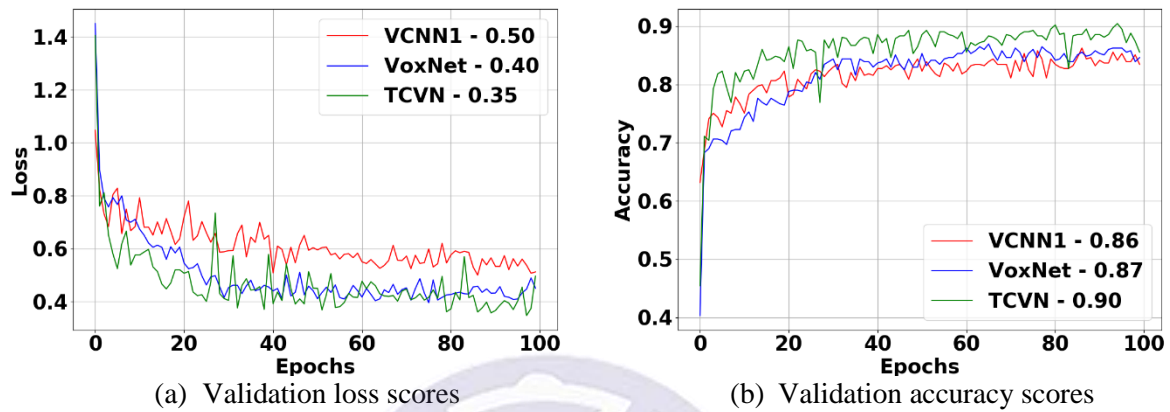


Figure 4-2 The comparison of TCVN model performances during training.

The summarized ranks of each model presented in Figure 4-2 are described respectively. (i) Figure 4-2(a) - loss scores, it presented that loss values for all models were decreasing during training. Therefore, the poorest performance was shown on the VCNN1 model that achieved the lowest error value on (0.5). Next, followed by the VoxNet model with the best (minimum) value on (0.4). Although there were overlapping performances drawn by VoxNet and TCVN model, the proposed model achieves the lowest loss as it goes down to (0.35). (ii) Figure 4-2(b) – accuracy scores, the rank proof on the result drawn in the loss value graph, where the TCVN model achieved the highest accuracy value on 90% accuracy of all validation data. Meanwhile, the VoxNet model only produces 87% accuracy and the VCNN1 model reaches the plateau on 86% accuracy.

Generally, the proposed TCVN model's performance is superior in terms of both accuracy and loss value. To clarify, the ranking results described above, a drawn confusion matrix in Figure 4-3 shows the performance result on every model of the validation data. Therefore, according to 1200 data that have been collected, it split into

33% for the validation data and 77% for the training data. In Figure 4-3, a total of 396 validation data were used to draw the confusion matrix for each model. Additionally, the blue color information of the confusion matrix is separated into two. (i) The darker blue color represents the correct predicted number of true positive and true negative in the validation data. (ii) The lighter blue color shows the incorrect predicted data, which known as a false positive and false negative. Furthermore, the TCVN model outperforms both of the models, indicating the importance of network depth and 3D convolution layer on the 3D data.

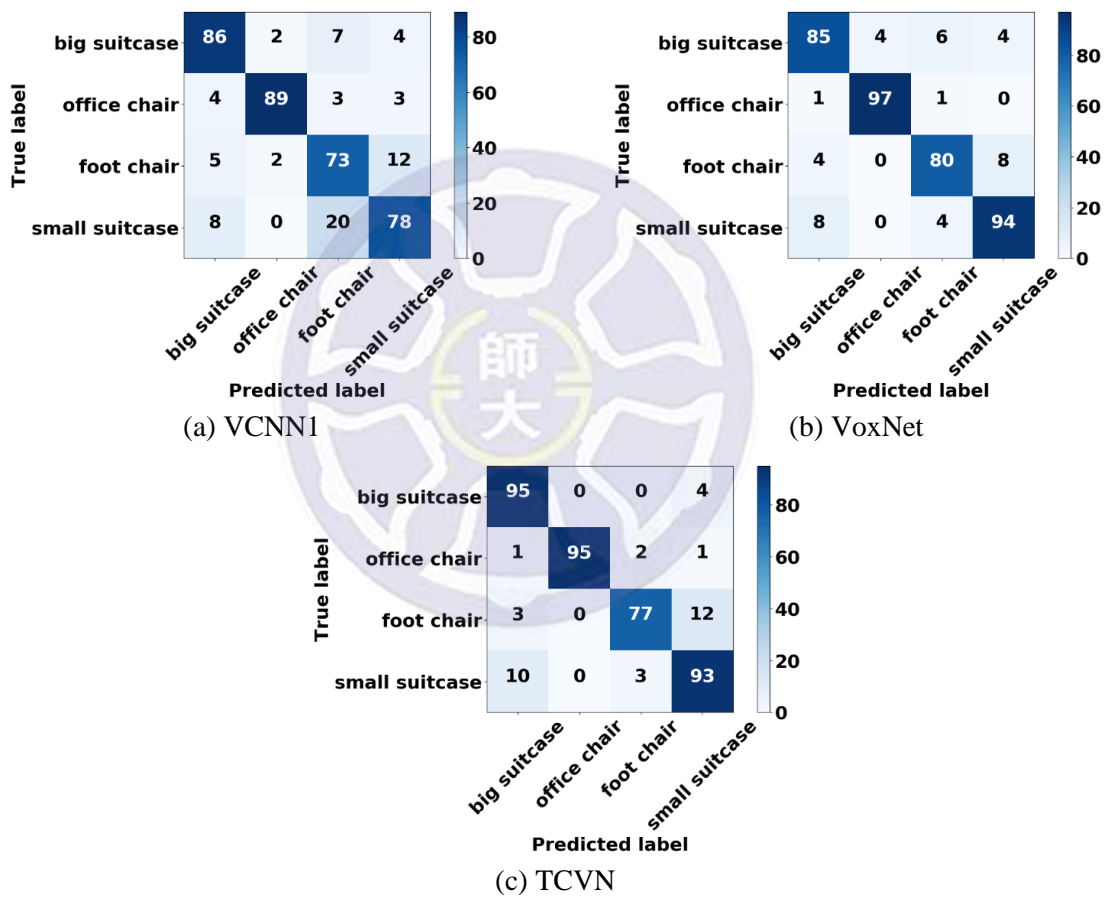


Figure 4-3 The comparison of the TCVN model in a confusion matrix of the validation data.

#### 4.2.2. Floor Detection Result

In this subsection, the floor detection algorithm is evaluated using the original YOLACT model and the proposed Tiny-YOLACT model on the custom floor images dataset. Therefore, the custom dataset is consisting of 850 train and 400 test images. Moreover, the collected train images were captured using the robot webcam. Whereas,

the test images were acquired from an external webcam with a side view of the experimental floor.

The training process is conducted with a batch size of 8 on the training computer (see Table 4-4). There were four models compared in these experiments, which are YOLACT with ResNet-18, 34, 50, and 101 as the backbone network for each model. Moreover, the pre-trained ResNet models on the ImageNet dataset were utilized from the PyTorch server<sup>13</sup>. As a result, the transfer learning method from the pre-trained weights was used to speed up the training process.

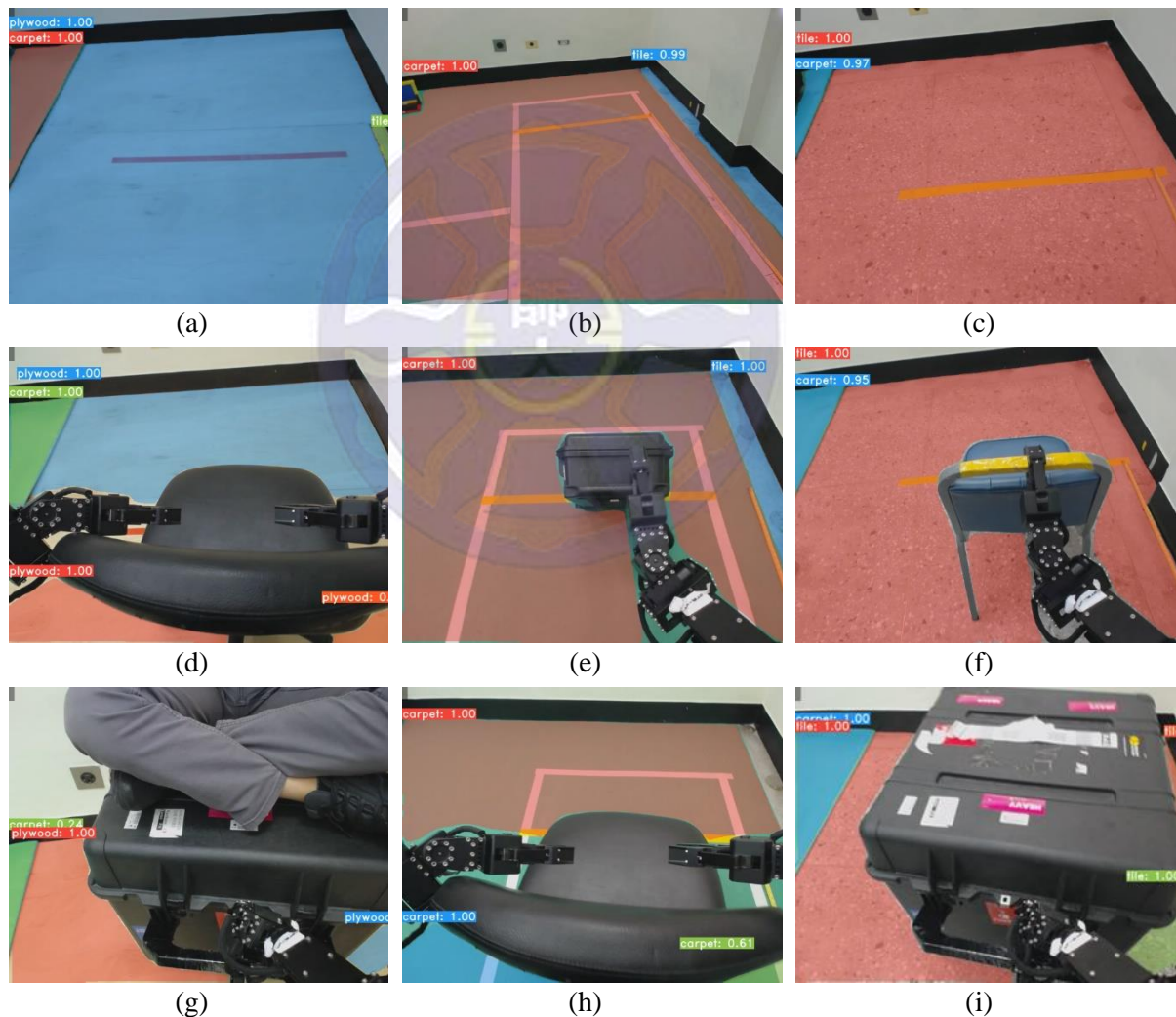


Figure 4-4 Example results of floor detection using the Tiny-YOLACT model.

**Note:** (a, d, g) plywood, (b, e, h) green carpet, and (e, f, i) tile.

<sup>13</sup> [https://pytorch.org/hub/pytorch\\_vision\\_resnet/](https://pytorch.org/hub/pytorch_vision_resnet/)

With looking at Figure 4-4, the result of the proposed Tiny-YOLACT model shows that the algorithm successful to segment the mask on each type of floor. Also, the mask result is smoother compared to other instance segmentation models. The reason is due to advantages of the “Proto Net” that produce temporal stability, although the model predicts different boxes cross frames, it’s not affected the prototypes and even resulting in a much more temporary stable mask [46].

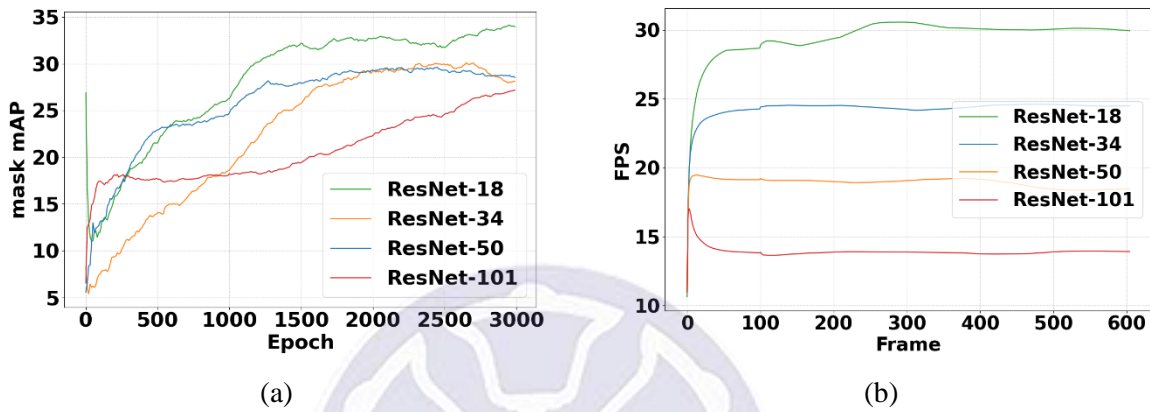


Figure 4-5 Result of validation mAP and FPS using different ResNet backbone.

Figure 4-5(a) presented the mAP value during the training of each model on validation data. It can be seen that every ResNet model converges with an average of more than 25 mAP on validation data. Therefore, due to the proposed Tiny-YOLACT model with ResNet-18 has a smaller number of parameters, the model achieves the fastest in terms of training time. Whereas, the model achieved 34.16 mAP on the validation data of the custom dataset within 1500 epoch.

In addition, Figure 4-5(b) shows the video processing time of all trained models. This shows that the YOLACT model can perform real-time instance segmentation in moderate GPU with specification similar to Table 4-5. Overall, the models were attaining more than 10 fps. In this case, the proposed Tiny-YOLACT model also obtained the fastest video processing times with an average of 29.56 fps.

### 4.3. Experimental Results for Robot Behavior

The performance of the proposed dragging object learning framework is described in this section. The walking balance parameter of offset  $CoB_x$  was learned using the ROS gazebo simulator. Then, the learned DQN algorithm will be used directly in the real THORMANG-Wolf robot.

#### 4.3.1. DQL-COB Training Results

The experiment scenario was on straight walking backward with foot step parameter shown in Table 4-3 and at the same time the robot is also grasping the object. This is illustrated in Figure 4-6, with denoted as white and red lines as the start and finish line for the dragging distance. Hence, the large size and heavy objects are simulated by using a 3D CAD big and heavy chair object. Moreover, the mass of an object is unknown to the robot. Therefore, as described in the introduction, the aim of this project is a robot learn by trial and error to drag the object (model-free RL).

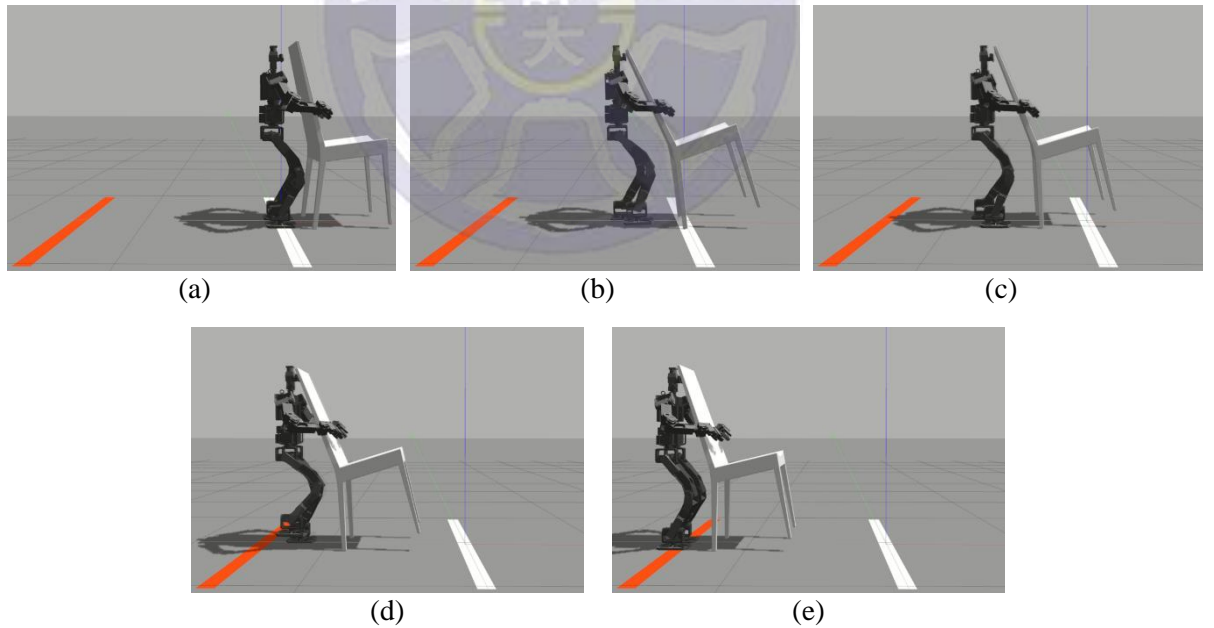


Figure 4-6 Snapshot of dragging in the Gazebo simulator.

One of the challenges in the machine-learning approach is selecting the correct hyperparameters for training the DNN. Therefore, a list of hyperparameters values used in this DQN algorithm is shown in Table 4-6.



Table 4-6 List of hyperparameters and values of the DQN

Hyperparameter	Value	Description
episode step	2000	number of sequences of steps set.
minibatch size	64	number of training cases over which each SGD update is computed.
replay memory size	50000	RMS optimizer updates are sampled from experienced by the agent that is given as input to the Q network.
target network update frequency	100	the frequency number of parameter update (this corresponds to parameter C from Algorithm 3)
discount factor	1	discount factor gamma used in the Q-learning update.
learning rate	0.01	the learning rate used by the RMS optimizer.
initial exploration	0.9	initial value of $\epsilon$ in $\epsilon$ -greedy exploration.
final exploration	0.05	final value of $\epsilon$ in $\epsilon$ -greedy exploration.

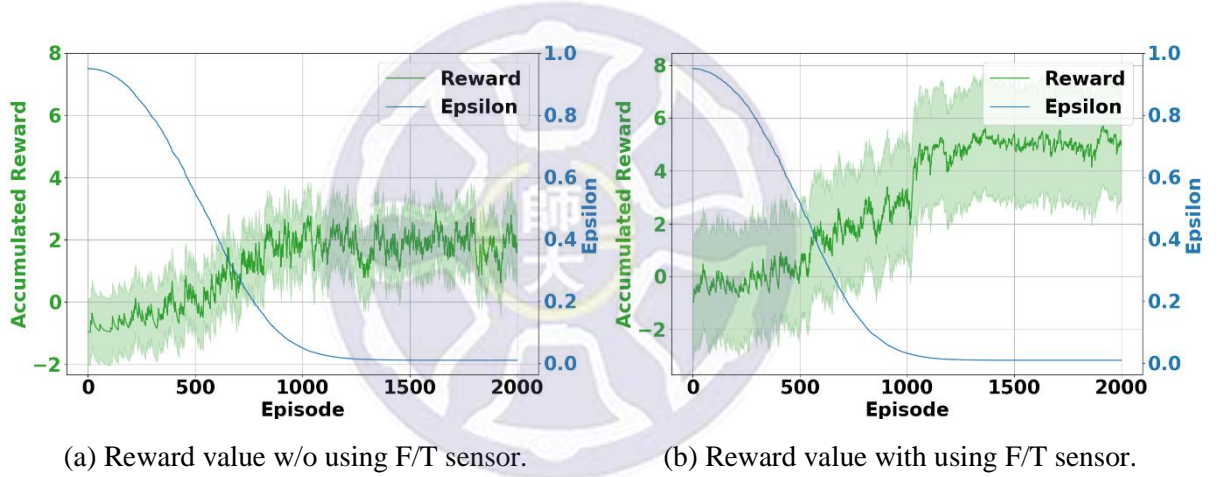


Figure 4-7 Comparison of accumulated reward during training.

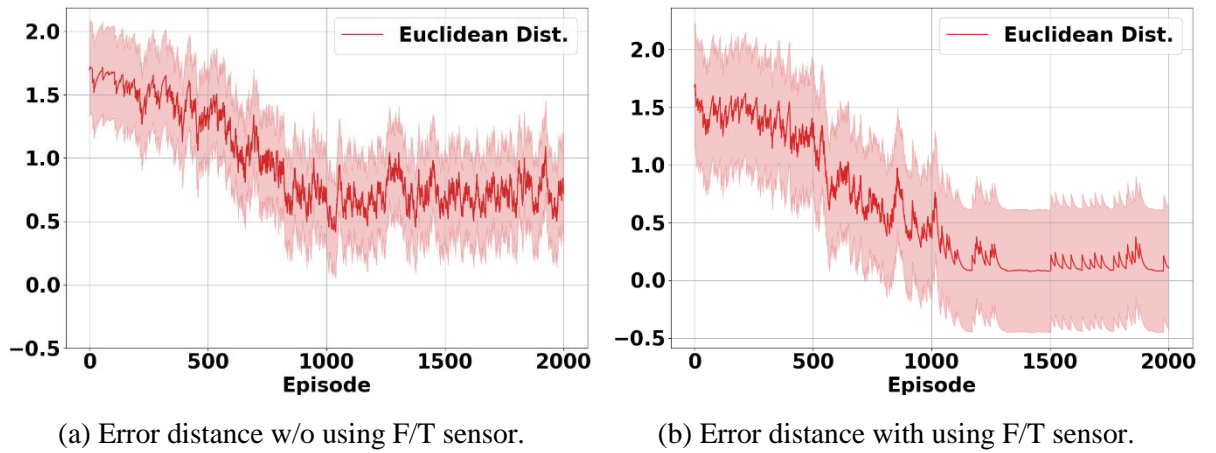


Figure 4-8 Comparison of Euclidean error during training.



Based on the proposed learning framework, two different sets of states are compared to evaluate the efficiency learning process. Those two states are based on the IMU pitch and roll axis, with and without using the F/T sensor. For this purpose, there were a total of 2000 episodes executed to learn the  $CoB_x$  walk balance parameter. Additionally, it took approximately 14 hours of training time to complete 2000 episodes in the ROS gazebo simulator.

Figure 4-7 shows the reward at each episode according to the DQN method. The plateau green curve means that the sub-optimal offset  $CoB_x$  with maximal reward was acquired in around 1000 episodes. Meanwhile, Figure 4-8 shows the Euclidean error that the robot reached the target line or not. This means the robot required to drag for a 2-meter distance to reach the finish (red line) from the start (white line). Therefore, the summary results of these comparison draw on the proposed DQL-COB with only using IMU sensors obtained maximum accumulated reward on 3.6 scalar value and reach minimum error 0.4-meter. On the other hand, our proposed DQL-COB with having F/T sensors as additional state is superior in terms of both accumulated reward and Euclidean error value. During training, we reported the highest accumulated reward of 5.8 scalar value and reached the best minimum error on 0.1 meter.

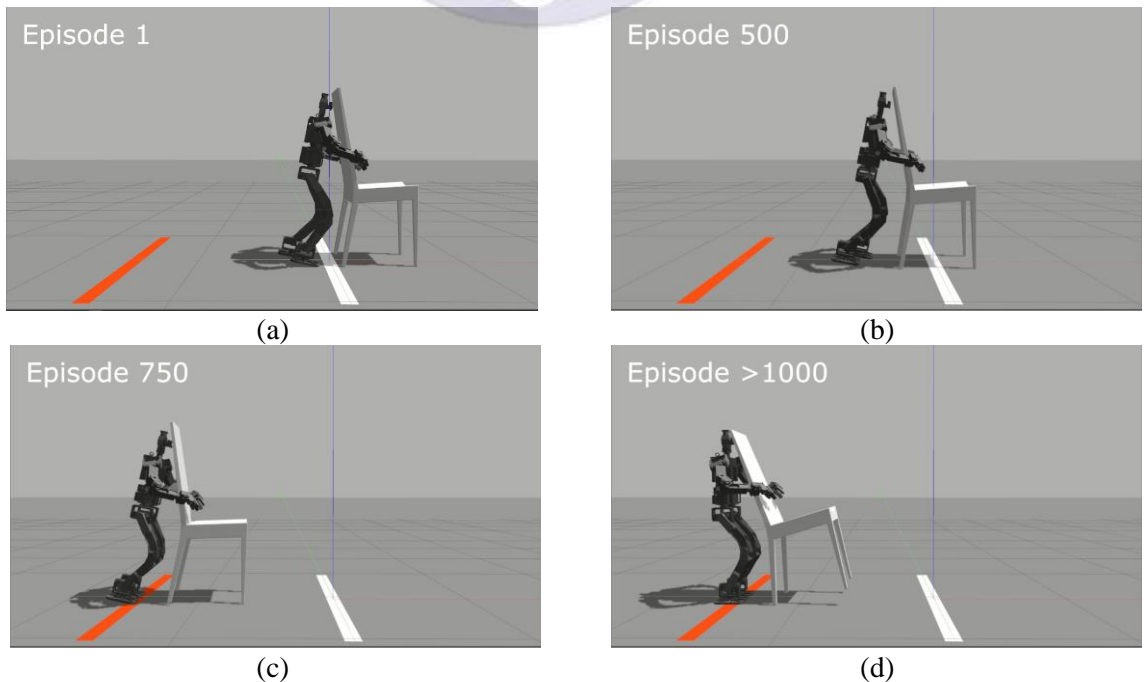


Figure 4-9 Snapshot during training in the Gazebo simulator.

Correspondingly, the graph result presented in Figure 4-7 and Figure 4-8 is re-illustrated in Figure 4-9. Whereas, the robot is having close behavior during training progress with and without using the F/T sensor. Therefore, to generalize the learning process, the phases are demonstrated with four-stage in Figure 4-9. First, in episode 1 until 500, due to the high probability of exploration, the robots keep falling forward in the starting stage. Second, during episode 500 to 750, the robot starts having prior knowledge with shows on the robot that can drag up to half of the target distance. Then, in the third stage (episode 750 till 1000), the robot is capable to drag until the target distance. However, in this phase, the performance is counted as semi-successful. Because the robot still falls in the finish line. Finally, after episode 1000 until 2000, the robot is qualified to drag the object. As a rule of exploitation in prior knowledge, the robot accomplishes dragging objects till the target line without falling.

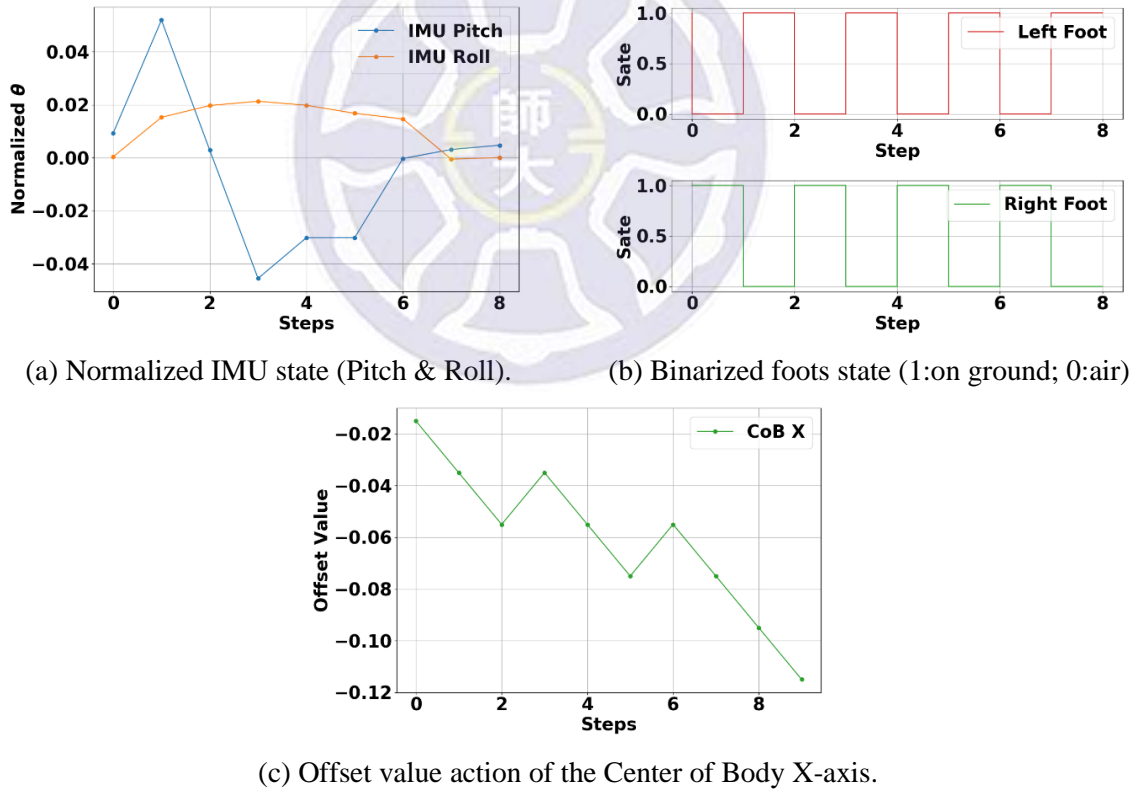


Figure 4-10 Recorded (states, actions) pair by the learned DQN during testing.

Based on results shown in Figure 4-7 and Figure 4-8, the proposed learning framework outperforms having an F/T sensor along with the IMU sensor on the pitch and roll axis on the environment state. Then, the learned DQL-COB model was tested

again in the Gazebo simulator. Therefore, we recorded the state and action  $(s, a)$  pair is illustrated in Figure 4-10. The learned robot show tends to decrease the offset value  $CoB_x$  to maintain the IMU pitch on the center range. In other words, maintaining the balanced posture of the robot also means that the algorithm successfully keeps track of the CoM of the robot. As a result, the robot capable to drag heavy and large objects without falling and it shown in the IMU pitch graph [see Figure 4-10(a)]. This behavior shows the robot act identically as a normal human. Whereas, a person tends to moves the torso backward during dragging for acquires large force.

### 4.3.2. DQL-COB Empirical Evaluation Result

This section implements the novel DQL-COB algorithm for the first time that uses training results in simulation and applied to the real adult-sized THORMANG-wolf robot for solving the MLHO problems. To evaluate the robustness of the learned DQL-COB algorithm, those mentioned objects and surfaces shown in Table 4-1 and Table 4-2 were illustrated again in Figure 4-11 until Figure 4-13.

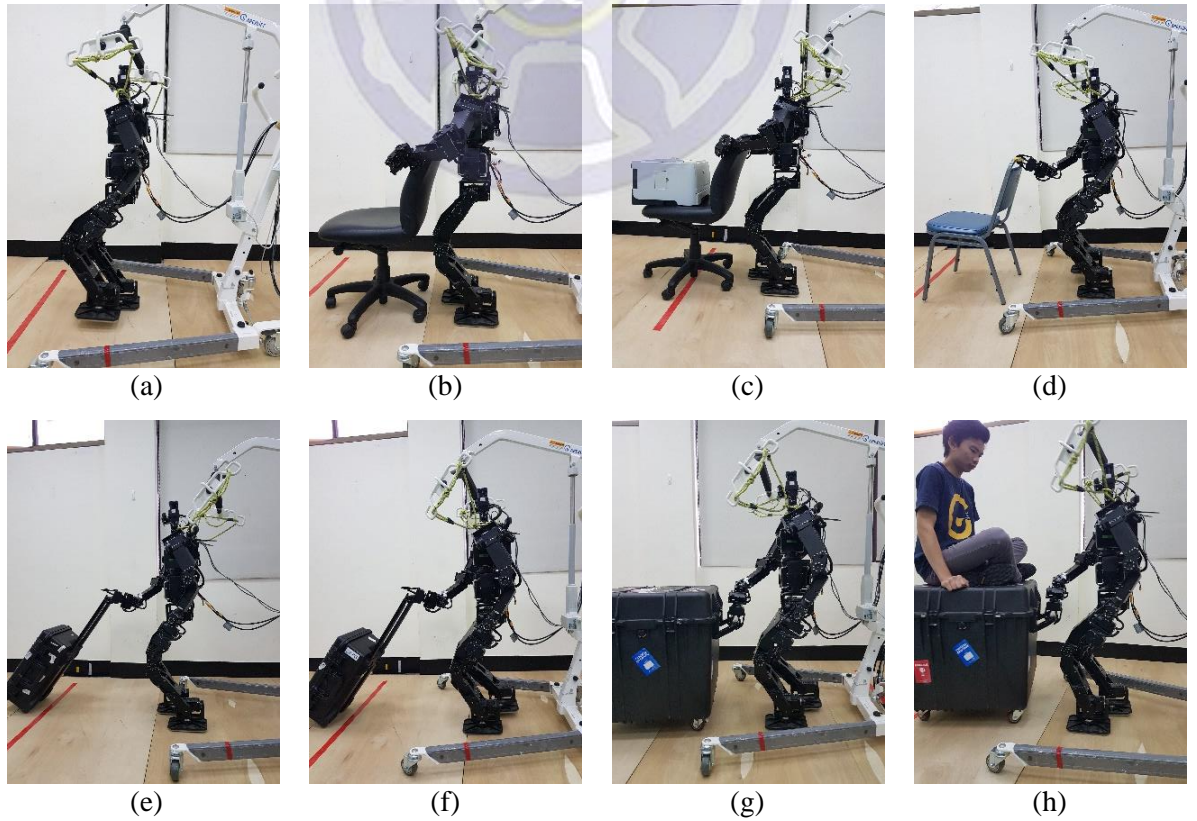


Figure 4-11 Snapshots testing on plywood surfaces.





(a)



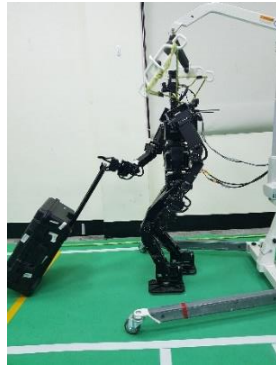
(b)



(c)



(d)



(e)



(f)



(g)



(h)

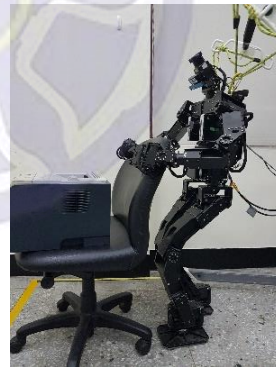
Figure 4-12 Snapshots testing on green carpet surfaces.



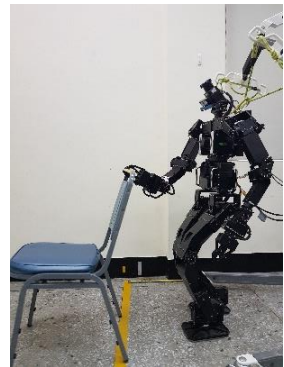
(a)



(b)



(c)



(d)



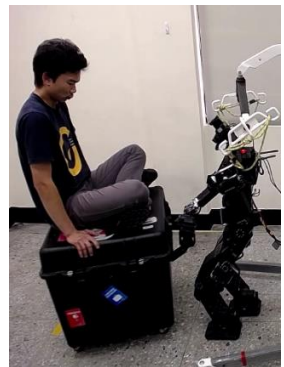
(e)



(f)



(g)



(h)

Figure 4-13 Snapshots testing on tile surfaces.

Based on the simulation, the DQL-COB algorithm model was trained on two different sets of states. Therefore, in the evaluation test to a real robot, the tests were conducted with 10 trial experiments on each object and each surface with and without using the F/T sensor. As a result, the total number of tests that have done was 480 tests, which are: 3 surfaces x 8 different object x 10 trial x 2 with and without using the F/T sensor. Table 4-7 shows the numbers of success rate in all experimental results and the video is available at ([https://youtu.be/pGy5bJm3J\\_E](https://youtu.be/pGy5bJm3J_E)). The details of each success rate in Table 4-7 means that the robot accomplished the dragging with Euclidean distance more than equal target distance without falling.

Table 4-7 Summary and comparison of the success rate result for all experiments.

**Note:** the term “w/o” indicating abbreviation of without.

Object	Plywood		Carpet		Tile	
	with F/T Sensor	w/o F/T Sensor	with F/T Sensor	w/o F/T Sensor	with F/T Sensor	w/o F/T Sensor
Default (No Object)	100%	100%	100%	100%	100%	100%
Office Chair	100%	100%	90%	100%	100%	90%
Office Chair with Load	100%	100%	100%	100%	100%	70%
Foot Chair	100%	100%	100%	100%	100%	100%
Small Suitcase	100%	100%	100%	100%	100%	100%
Small Suitcase with Load	100%	100%	100%	90%	90%	60%
Big Suitcase	100%	100%	100%	100%	100%	100%
Big Suitcase with Human	90%	0%	0%	0%	60%	0%

As shown in Figure 4-14, the success rate of dragging all objects for the entire experiment was having a slight difference in the performances. Therefore, the total percentage of success rate using the learned robot with having the F/T sensor is 92.91%. Comparatively, the robot successful rate without using the F/T sensor is 83.75%. Therefore, to analyze the difference between the learned robot on two different sets of states, the recorded  $(s, a)$  pairs are compared.

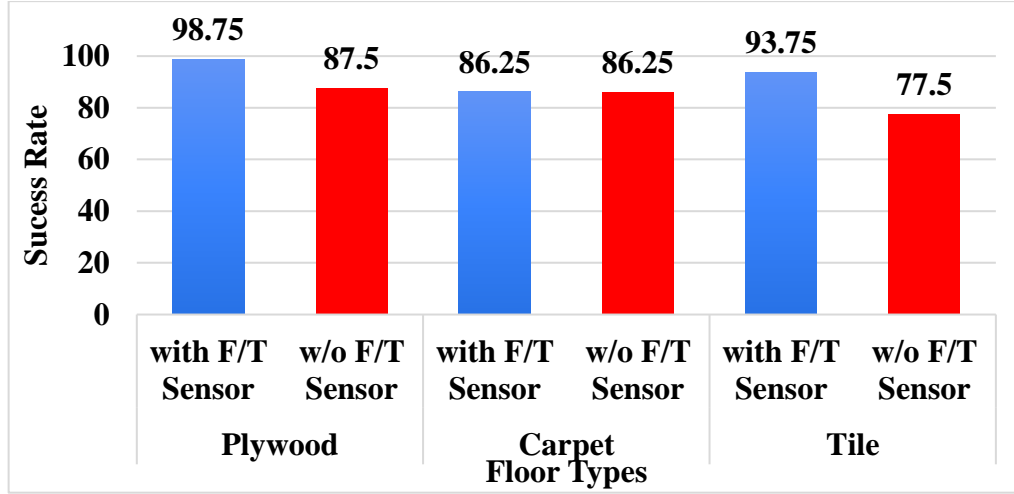
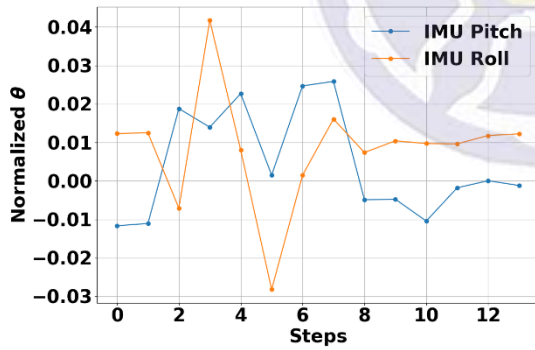
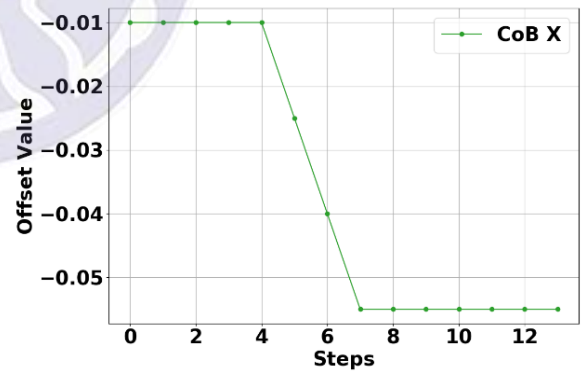


Figure 4-14 The success rate of dragging all objects per each surface.

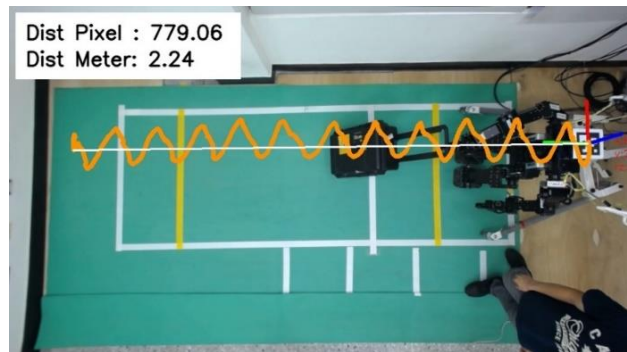
Additionally, the evaluation number of footsteps are kept in the same with the simulation. This means that the target distance for straight backward walking is still same 2 meters. Although with the absence of the F/T sensor, the robot still could drag most of the experimental objects. This shows in Figure 4-15, the captured  $(s, a)$  pairs during dragging a small suitcase with the load as the evaluation sample.



(a) IMU States



(b) Offset  $COB_x$



(c) Trajectory and Euclidean distance during dragging

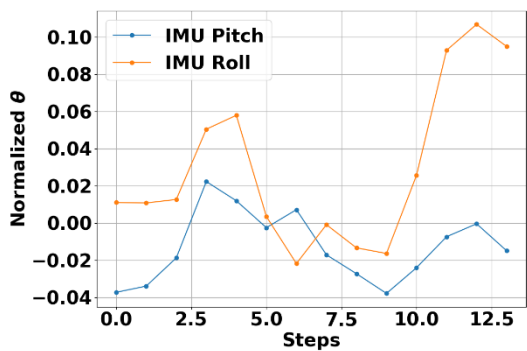
Figure 4-15 Recorded  $(s, a)$  pair without using the F/T sensor.



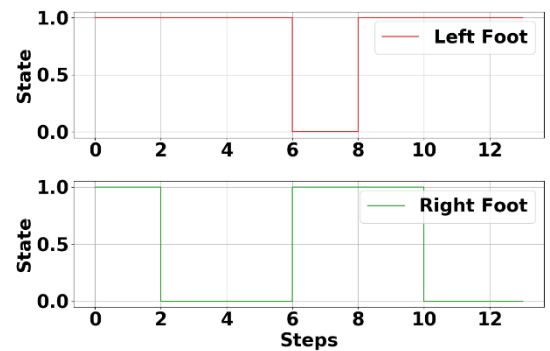
Generally, an example result of the captured behaviors of  $(s,a)$  pairs on the proposed DQL-COB algorithm without using the F/T sensor in Figure 4-15 shows a general trend for dragging other types of objects. The actions taken by the robot appear to be sluggish or insensitive toward the dynamic of states. For this reason, the shortcoming was on dragging the massive objects (big suitcase with a human), that robot fails to drag it.

On the contrary, the performance of the learned agent with having the F/T sensor is more robust. This proves on the robot that capable to drag a big suitcase (18.6kg) with an adult human (66kg) seated on top of it. The proposed learning framework shows the capability of a THORMANG-Wolf robot to drag an object with a load 84.6kg (double of its weight). As shown in Figure 4-16, the recorded  $(s,a)$  pairs of the robot using the F/T sensors during dragging a massive object. This is showing the benefit of using F/T sensors as the additional features to the DQL-COB algorithm, that the actions taken by the robot are more stable.

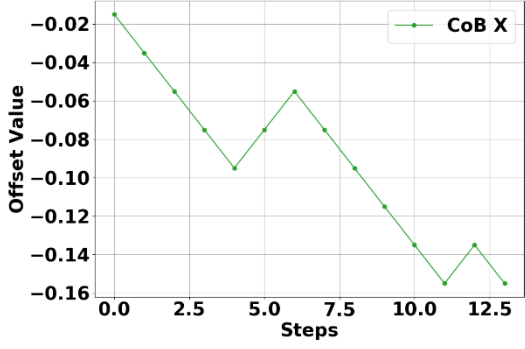
As a result, the experiments result clearly show that the proposed hierarchical method for dragging objects which implemented based on deep neural networks can recognize and perform superior results with a very high success rate. Although the state of sensor values of real robots is slightly different from simulation, it allows having better generalization for unknown states. For this reason, the proposed DQL-COB algorithm can generalize more features based on experience environment and action.



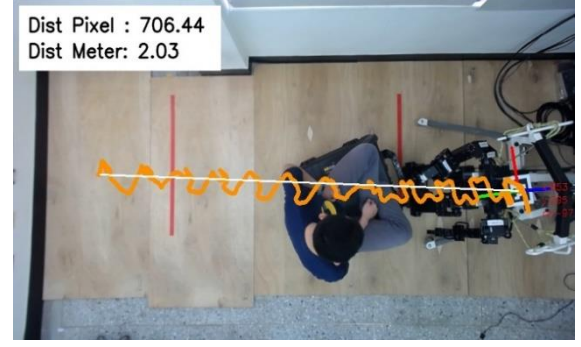
(a) IMU States



(b) Foot States



(c) Offset  $CoB_X$



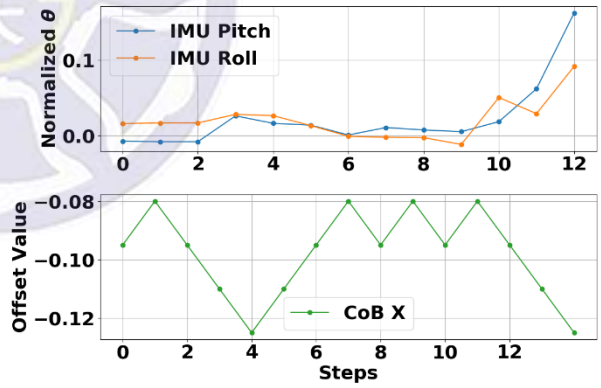
(d) Trajectory and Euclidean distance during dragging

Figure 4-16 Recorded (states, actions) pair using the F/T sensor.

To answer, the failure condition in Table 4-7, the analysis of sample conditions is illustrated in Figure 4-17 and Figure 4-18. Therefore, as illustrated in Figure 4-17, the robot was failed to drag an empty small suitcase at the finish line. The main reason is the behavior of the fluctuating offset  $CoB_X$ . However, this problem only happened on the learned agent without using the F/T sensors.



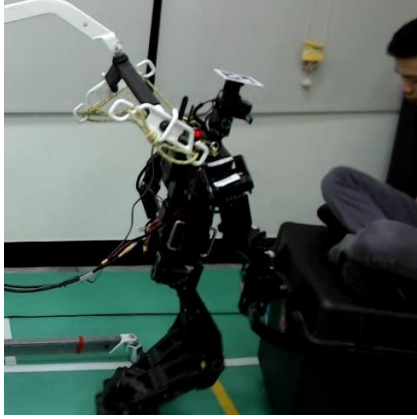
(a)



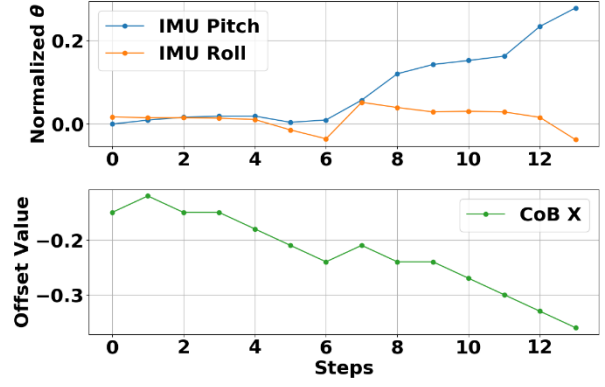
(b)

Figure 4-17 Example of a failure condition in dragging an empty small suitcase.

One other thing of the highest failure rate is shows on dragging a big suitcase with a human on a green carpet surface. Wherefore, the initial value of  $CoB_X$  parameter of the robot is -0.015. This means the robot has a straight torso posture before dragging the object [see Figure 3-20(b)]. Also, the green carpet surface has a smaller coefficient of friction. As a result, the robot has a high probability to fall forward in beginning for dragging a high load object.



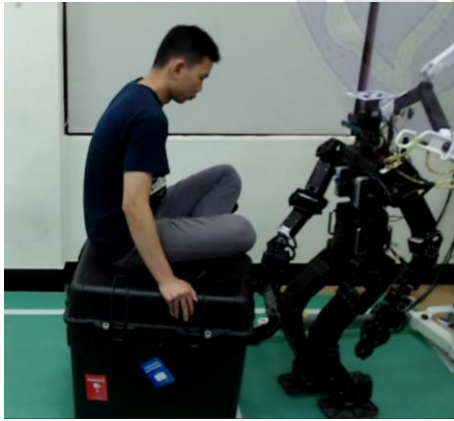
(a)



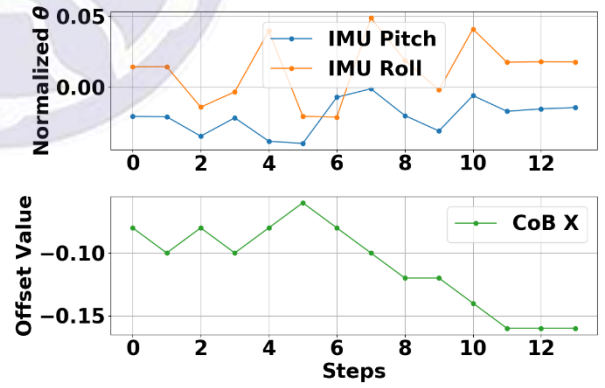
(b)

Figure 4-18 Example of a failure condition in dragging a big suitcase with a human.

For all these reasons, we conduct another experiment with a pre-defined initial offset of  $-0.08 \text{ CoB}_x$ . Wherefore, the designated initial offset implies the robot postures with an initial pose that slightly backward. Then, we applied the proposed DQL-COB by using the F/T sensor. As a result, the robot could successfully drag an object and tackle the failure condition described in Figure 4-18. Moreover, the recorded  $(s, a)$  pairs of this experiment illustrated in Figure 4-19.



(a)



(b)

Figure 4-19 The pre-defined CoB-X to dragging a big suitcase with a human.

## Chapter 5: Closing

### 5.1. Conclusion

In this thesis, presented a hierarchical deep learning method in which we used adult-sized biped humanoid robot THORMANG-Wolf to drag large size and heavy object. The main objective of the proposed method was to drag different objects on various flat surfaces through the learning-based method. The validation of this experiment was tested to drags several common objects and surfaces available in human daily life. To sum up, the learning-based approached in this MLHO problem, it can be divided into three different learning system.

First, for object detection, the approach was done using a deep learning method on 3D object classification. The 3D data were acquired using a point cloud from the robot's LiDAR scanner. A pre-process Euclidean distance-based filter and voxelization algorithm to down-sample the point cloud data into a fixed size  $30 \times 30 \times 30$  volumetric occupancy grid. In brief, the proposed TCVN model that based VoxNet [52] and V-CNN1 [53] achieved a 90% accuracy in real-time.

Second, for the floor detection, the approach was done using a deep learning method on real-time instance segmentation. We proposed a model based on the original YOLACT model [46] that is modified on the backbone network using the ResNet-18 and called this customized model as Tiny-YOLACT. For the training process, a custom dataset (floor images) was acquired from the robot's webcam with a COCO format. This custom model achieved a 34.16 mAP on validation data with an average of 29.56 fps on standard NVIDIA GTX-1060 GPU.

Third, for the deep reinforcement learning method, it was done using the Deep Q-Learning method to produce the offset parameter  $CoB_x$  during dragging. We proposed a DQL-COB algorithm with the environment and the training process was done in the ROS-Gazebo simulator. It took approximately 20 hours of training time on a single deep-learning computer (see Table 4-4) to complete 2000 episodes for the robot

to learned and solved the MLHO problem. In the experiments, the learned agent from the simulator was directly tested to a real robot with a success rate percentage of 92.91% using the F/T sensor and 83.75% without using F/T sensors.

Overall, the sequence of those multi learning-based approached was done sequentially. Briefly, a robot scan objects with LiDAR and utilized the TCVN model to access the pre-recorded motion on grasping an object. Afterward, the floor detection result from Tiny-YOLACT was used to select a coefficient offset value on offsetting  $CoB_x$ . Whereas, the DQL-COB algorithm acted as  $CoB_x$  an offset manager based on the IMU and F/T sensors. As a result, the offset  $CoB_x$  is implemented to keep tracking with the robot's center of mass, that robot can keep balance with maintaining the ZMP in support polygon.

## 5.2. Future Work

In future work, the plan is removing instance segmentation of floor detection part with extending to a deep reinforcement learning algorithm that adding a raw image from the robot camera as an additional state. Therefore, the agent can differentiate the offset value  $CoB_x$  based on types of surfaces directly and act more robustly. Also, implementing the dynamic inverse kinematic grasping point technique based on LiDAR point cloud data should be further studied. On the whole, broaden the dragging large and heavy object into pulling and pushing large and heavy objects as well.



## Bibliographies

- [1] K. Tanie, "Humanoid robot and its application possibility," in *Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI2003.*, 2003, pp. 213-214.
- [2] A. Choudhury, H. Li, C. Greene, and S. Perumalla, "Humanoid Robot-Application and Influence," *arXiv preprint arXiv:1812.06090*, 2018.
- [3] S. Kajita, H. Hirukawa, K. Harada, and K. Yokoi, *Introduction to humanoid robotics*. Springer, 2014.
- [4] T. Takubo, K. Inoue, and T. Arai, "Pushing an Object Considering the Hand Reflect Forces by Humanoid Robot in Dynamic Walking," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, Spain, 2005, pp. 1706-1711.
- [5] N. Motoi, M. Ikebe, and K. Ohnishi, "Real-Time Gait Planning for Pushing Motion of Humanoid Robot," *IEEE Transactions on Industrial Informatics*, vol. 3, no. 2, pp. 154-163, 2007.
- [6] D. Omrčen, C. Böge, T. Asfour, A. Ude, and R. Dillmann, "Autonomous acquisition of pushing actions to support object grasping with a humanoid robot," in *2009 9th IEEE-RAS International Conference on Humanoid Robots*, 2009, pp. 277-283.
- [7] S. Nozawa, Y. Kakiuchi, K. Okada, and M. Inaba, "Controlling the planar motion of a heavy object by pushing with a humanoid robot using dual-arm force control," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 1428-1435.
- [8] M. Murooka, S. Nozawa, Y. Kakiuchi, K. Okada, and M. Inaba, "Whole-body pushing manipulation with contact posture planning of large and heavy object for humanoid robot," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5682-5689.
- [9] K. Harada *et al.*, "A Humanoid Robot Carrying a Heavy Object," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, Spains, 2005, pp. 1712-1717.
- [10] J. C. Vaz, H. Lee, Y. Jun, and P. Oh, "Towards tasking humanoids for lift-and-carry non-rigid material," in *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2017, pp. 316-321.
- [11] Y. Ohmura and Y. Kuniyoshi, "Humanoid robot which can lift a 30kg box by whole body contact and tactile feedback," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, USA, 2007, pp. 1136-1141.
- [12] A. Laurenzi, D. Kanoulas, E. M. Hoffman, L. Muratore, and N. G. Tsagarakis, "Whole-Body Stabilization for Visual-Based Box Lifting with the COMAN+ Robot," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, Naples, Italy, Italy, 2019, pp. 445-446.
- [13] M. De Looze, K. Van Greuningen, J. Rebel, I. Kingma, and P. Kuijer, "Force direction and physical load in dynamic pushing and pulling," *Ergonomics*, vol. 43, no. 3, pp. 377-390, 2000.



- [14] A. Argubi-Wollesen, B. Wollesen, M. Leitner, and K. Mattes, "Human body mechanics of pushing and pulling: analyzing the factors of task-related strain on the musculoskeletal system," *Safety and health at work*, vol. 8, no. 1, pp. 11-18, 2017.
- [15] D. Torricelli *et al.*, "Human-like compliant locomotion: state of the art of robotic implementations," *Bioinspiration & biomimetics*, vol. 11, no. 5, p. 051002, 2016.
- [16] K. Harada, S. Kajita, K. Kaneko, and H. Hirukawa, "Dynamics and balance of a humanoid robot during manipulation tasks," *IEEE Transactions on Robotics*, vol. 22, no. 3, pp. 568-575, 2006.
- [17] J. Yang, S. Ogawa, T. Tsujita, S. Komizunai, and A. Konno, "Massive object transportation by a humanoid robot," *IFAC-PapersOnLine*, vol. 51, no. 22, pp. 250-255, 2018.
- [18] F. Abi-Farraj, B. Henze, C. Ott, P. R. Giordano, and M. A. Roa, "Torque-Based Balancing for a Humanoid Robot Performing High-Force Interaction Tasks," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2023-2030, 2019.
- [19] E. Yoshida, P. Blazevic, V. Hugel, K. Yokoi, and K. Harada, "Pivoting a large object: whole-body manipulation by a humanoid robot," *Applied Bionics and Biomechanics*, vol. 3, no. 3, pp. 227-235, 2006.
- [20] E. Yoshida *et al.*, "Motion planning for whole body tasks by humanoid robot," in *IEEE International Conference Mechatronics and Automation, 2005*, Niagara Falls, Ont., Canada, 2005, vol. 4, pp. 1784-1789: IEEE.
- [21] M. Stilman, K. Nishiwaki, and S. Kagami, "Humanoid teleoperation for whole body manipulation," in *2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA, 2008, pp. 3175-3180: IEEE.
- [22] M. Stilman, K. Nishiwaki, and S. Kagami, "Learning object models for whole body manipulation," in *2007 7th IEEE-RAS International Conference on Humanoid Robots*, 2007, pp. 174-179.
- [23] E. Berger, "Friction modeling for dynamic system simulation," *Applied Mechanics Reviews*, vol. 55, no. 6, pp. 535-577, 2002.
- [24] J. Woodhouse, T. Putelat, and A. McKay, "Are there reliable constitutive laws for dynamic friction?," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 373, no. 2051, p. 20140401, 2015.
- [25] S. Kim, S. Hong, and D. Kim, "A walking motion imitation framework of a humanoid robot by human walking recognition from IMU motion data," in *2009 9th IEEE-RAS International Conference on Humanoid Robots*, 2009, pp. 343-348.
- [26] B. Hengst, M. Lange, and B. White, "Learning ankle-tilt and foot-placement control for flat-footed bipedal balancing and walking," in *2011 11th IEEE-RAS International Conference on Humanoid Robots*, 2011, pp. 288-293.
- [27] J. Lin, K. Hwang, W. Jiang, and Y. Chen, "Gait Balance and Acceleration of a Biped Robot Based on Q-Learning," *IEEE Access*, vol. 4, pp. 2439-2449, 2016.
- [28] W. Wu and L. Gao, "Posture self-stabilizer of a biped robot based on training platform and reinforcement learning," *Robotics and Autonomous Systems*, vol. 98, pp. 42-55, 2017.

- [29] K. Hwang, W. Jiang, Y. Chen, and H. Shi, "Motion Segmentation and Balancing for a Biped Robot's Imitation Learning," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1099-1108, 2017.
- [30] X. Wu, S. Liu, T. Zhang, L. Yang, Y. Li, and T. Wang, "Motion Control for Biped Robot via DDPG-based Deep Reinforcement Learning," in *2018 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, 2018, pp. 40-45.
- [31] C.-C. Wong, C.-C. Liu, S.-R. Xiao, H.-Y. Yang, and M.-C. Lau, "Q-Learning of Straightforward Gait Pattern for Humanoid Robot Based on Automatic Training Platform," *Electronics*, vol. 8, no. 6, p. 615, 2019.
- [32] M. Nakada, B. Allen, S. Morishima, and D. Terzopoulos, "Learning Arm Motion Strategies for Balance Recovery of Humanoid Robots," in *2010 International Conference on Emerging Security Technologies*, 2010, pp. 165-170.
- [33] S. Yi, B. Zhang, D. Hong, and D. D. Lee, "Online learning of a full body push recovery controller for omnidirectional walking," in *2011 11th IEEE-RAS International Conference on Humanoid Robots*, 2011, pp. 1-6.
- [34] D. Luo, X. Han, Y. Ding, Y. Ma, Z. Liu, and X. Wu, "Learning push recovery for a bipedal humanoid robot with Dynamical Movement Primitives," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 1013-1019.
- [35] P. Mendez-Monroy, "Walking motion generation and neuro-fuzzy control with push recovery for humanoid robot," *Int. J. Comput. Commun.*, vol. 12, no. 3, pp. 330-346, 2017.
- [36] H. Kim, D. Seo, and D. Kim, "Push Recovery Control for Humanoid Robot Using Reinforcement Learning," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 488-492.
- [37] P. Beeson and B. Ames, "TRAC-IK: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 928-935.
- [38] J.-Y. Kim, I.-W. Park, and J.-H. Oh, "Walking Control Algorithm of Biped Humanoid Robot on Uneven and Inclined Floor," *Journal of Intelligent and Robotic Systems*, vol. 48, no. 4, pp. 457-484, 2007/04/01 2007.
- [39] M. Vukobratović and B. Borovac, "Zero-moment point—thirty five years of its life," *International journal of humanoid robotics*, vol. 1, no. 01, pp. 157-173, 2004.
- [40] J. Darrell, J. Long, and E. Shelhamer, "Fully Convolutional Networks for Semantic Segmentation," *IEEE T PATTERN ANAL*, vol. 39, no. 4, 2014.
- [41] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91-99.
- [42] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *European conference on computer vision*, 2016, pp. 21-37: Springer.
- [43] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
- [44] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980-2988.

- [45] Y. Li, H. Qi, J. Dai, X. Ji, and Y. Wei, "Fully Convolutional Instance-Aware Semantic Segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4438-4446.
- [46] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: Real-Time Instance Segmentation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 9156-9165.
- [47] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652-660.
- [48] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279-292, 1992/05/01 1992.
- [49] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
- [50] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, 2016, pp. 50-56.
- [51] ROBOTIS Inc. (December 19). *THORMANG3 Full size open platform humanoid*. Available: [http://en.robotis.com/model/page.php?co\\_id=prd\\_thormang3#](http://en.robotis.com/model/page.php?co_id=prd_thormang3#)
- [52] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany, 2015, pp. 922-928: IEEE.
- [53] V. Hegde and R. Zadeh, "Fusionnet: 3d object classification using multiple data representations," *arXiv preprint arXiv:1607.05695*, 2016.
- [54] X.-F. Han, J. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao, "A review of algorithms for filtering the 3D point cloud," *Signal Processing: Image Communication*, vol. 57, 05/01 2017.
- [55] B. Chakraborty, B. Shaw, J. Aich, U. Bhattacharya, and S. K. Parui, "Does Deeper Network Lead to Better Accuracy: A Case Study on Handwritten Devanagari Characters," in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, 2018, pp. 411-416.
- [56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition. CoRR abs/1512.03385 (2015)," ed, 2015.
- [57] M. Vukobratovic and B. Borovac, "Zero-Moment Point - Thirty Five Years of its Life," *I. J. Humanoid Robotics*, vol. 1, pp. 157-173, 03/01 2004.
- [58] S. Kajita *et al.*, *Biped walking pattern generation by using preview control of zero-moment point*. 2003, pp. 1620-1626 vol.2.

## **Autobiography**

Hanjaya Mandala received his B. App. Sc degree in Mechatronic Engineering Study Program of Electrical Engineering from Polytechnic State Batam, Indonesia in August 2017. After graduation, he has worked for one year at Epson company as an electrical designer in the factory automation department. He is in charge of designing electrical and programming automation machines. Then in August 2018, he continues his Master's Degree in the Electrical Engineering Department at the National Taiwan Normal University and joined the Educational Robotics Centre laboratory which focused on humanoid robots. He has been worked on the humanoid kid-size robots since 2014 and has obtained 3rd places at RoboCup 2018 kid-size humanoid robot competitions. Also, three-rows of 1st place at Indonesia National Humanoid Robot Competitions. His research interest includes robotics, computer vision, and artificial intelligence.



## Academic Achievement

1. H. Mandala, S. Saeedvand, and J. Baltes, "Synchronous Dual-Arm Manipulation by Adult-Sized Humanoid Robot," in 2020 International Conference on Advanced Robotics and Intelligent Systems (ARIS), (accepted)
2. IEEE/RSJ IROS 2019 (Macau) - 1<sup>st</sup> Place Humanoid Robot Application Challenge.
3. Iran FIRA RoboWorldCup Open 2019 (Iran) - 1<sup>st</sup> Place all-round HuroCup Kid-size Humanoid.
4. International Intelligent RoboSports Competition 2020 (Taiwan) - 1<sup>st</sup> Place HuroCup Kid-size Humanoid Sprint & Marathon.

