

國立臺灣大學電機資訊學院電機工程學研究所



碩士論文

Graduate Institute of Electrical Engineering
College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

智慧服務機器人基於遞迴類神經網路進行

未知室內語意導航之研究

Unknown Indoor Semantic Navigation Based on
Recursive Neural Network for Intelligent Service Robotics

陳長鈞

Chang-Jiun Chen

指導教授：羅仁權 博士

Advisor: Ren.C. Luo, Ph.D.

中華民國 106 年 7 月

July 2017

誌謝



就讀碩士班的這兩年是我人生中一段美好的回憶。時光匆匆飛逝，一轉眼就到了鳳凰花開的時刻。特別感謝父母親辛苦地栽培與教誨，給予我支持與鼓勵，讓我能夠無憂無慮地專心於課業上。感謝我的指導教授羅仁權教授，提供我們豐富的資源以及在研究過程與論文的指導，同時培養我們大至國際觀、小至做人處事的道理。過程中所經歷的一切都是一輩子難得的體驗，也從其中學到非常多寶貴的知識與經驗。二年的碩士生涯中，老師充分地展現出國際級專家學者的風範與視野，不僅帶領我們探索學術知識上的深度與廣度，更給予我們充分的機會去體驗研究以外的經驗，期望我們成為均衡發展的人才。在待人處事方面，老師更是教給我們許多社會上應對進退的準則，這一切都使我獲益良多。

在國立臺灣大學智慧機器人及自動化國際研究中心 (NTU- iCeIRA) 兩年的研究生活中，我要感謝鏡文、瑋隆、東榕、繼棠、金成、昕昞、旭佳、禮聰、志遠、獻章等博班學長；他們不僅不厭其煩的給我指導，並且也教我如何自己摸索出問題的答案。還要感謝碩班學長銘駿、建安、士紘、煒森、文謙、金博、建偉、冠志、柏宏、榮育，從優秀的學長們身上我學到很多，也常把他們當做努力的榜樣，期許自己也能跟學長們一樣厲害。更不能忘記同屆一起努力奮鬥的伙伴俊豪、莉彤、李晟、晴岡、靖霖、昱佑、達方、凱鈞、仲凱、孟勳、柏凱，和你們一起窩在實驗室，不論是做研究、忙比賽還是一起玩樂，都是我碩班生涯最快樂的事情之一。謝謝積極認真又貼心的學弟妹們，培淳、石崑、武昱、嵩詠、智堅、威辰、錦賢、育榕、育澤、名彥、展嘉、何鑫、王昊、曾旻，幫忙大大小小的事務，一起打球運動。也要感謝默默在背後幫我們完成許多瑣事的雯雅 (Tracy)、煜倫 (Dornin)、姿伶 (Amy)、芳嫻 (Helen)、佩芸 (Winnie) 等助理們。

最後，感謝我的朋友們在背後給予我莫大的鼓勵，你們的存在是我前進的動力，希望我們的友誼能夠持續一輩子。能完成這篇論文，我要感謝在我生命中出現的每一個人，真的非常謝謝你們。

陳長鈞 謹誌

民國一百零六年七月

中文摘要



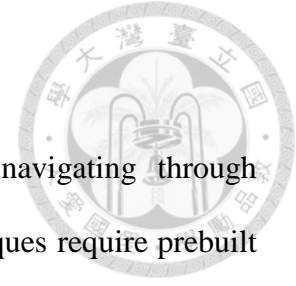
近年的研究已經讓服務型機器人具備能在複雜的室內環境中移動之功能。然而，這些技術往往需要根基於事先建立好的環境地圖，因而無法應用在未知的環境中。與此相對，人類在進入未知的環境時，常依靠問路這一方法，來得知如何抵達某一地點，並進一步移動到該處。目前的移動型機器人，尚缺乏這種依據接收到的口頭指令，在未知的環境中導航的能力。

在本研究中，我們的目標是將於未知環境中導航的功能，實作在移動型機器人上。我們以室內環境作為主體，利用遞迴類神經網路的方法，讓機器人學習人類導航的方法。我們設計了一個導航系統，並以人類的導航紀錄和相對應的導航指令來訓練此系統。我們將導航指令進行分割，並將每個切割出來的簡單指令分類到十個我們所定義的基本指令集當中；而每筆人類的導航紀錄，都是根據某一類基本指令來進行收集的。在訓練類神經網路模型的過程中，我們更提出一驗證的方法來檢驗訓練之模型的有效性。

最後，我們在模擬和實際的環境中測試此導航系統。我們在搬運機器人「企鵝」上實作我們的系統，並實驗其是否能根據不同的導航指令，移動到對應的地點。我們將機器人的移動路徑，與接受同樣指令的人類所走出來的路徑進行比較；而結果顯示，基於此一導航系統的移動型機器人，能達到接近於人類的導航表現。

關鍵字：服務型機器人，移動型機器人、語意式導航、機器人深度學習、人機互動

ABSTRACT



Recent researches have made service robots capable of navigating through complex and clustered indoor environments. However, such techniques require prebuilt maps and cannot be applied to unknown environments. By contrast, when entering an unknown environment, humans can ask someone for directions to figure out how to get to a specific location, and further navigate to the destination by following the instructions. Present mobile robots lack the ability of navigating under unknown environments according to the given verbal instructions.

In this research, we aim to implement the ability of navigating through unknown environments on mobile robots. We focus on indoor environments, using recursive neural networks to make robots learn the methods of navigating from humans. We design a navigation system, which is trained by human-controlled navigating records along with instructions. Instructions are split and then classified into ten basic classes, and each navigating record is collected according to one of these basic instruction classes. During the training process, we propose a validating method to evaluate the effectiveness of our models.

Finally, we put our system to the test under both simulation and real environments. We implement the system on a warehouse robot called 'Penguin', and test whether it can get to desired positions according to different given instructions. We compare the navigation paths of our mobile robot with those of humans following the same verbal instructions. The results show that our mobile robot can achieve similar performance to that of humans.

Keywords: service robotics, mobile robotics, semantic navigation, deep learning in robotics and automation, human-robot interaction



CONTENTS



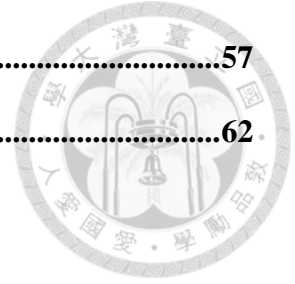
口試委員會審定書

誌謝	i
中文摘要	ii
ABSTRACT	iii
CONTENTS	v
LIST OF FIGURES	viii
LIST OF TABLES	x
Chapter 1 Introduction	1
1.1 Problem Statement.....	1
1.2 Related Works.....	4
1.3 Research Objective	6
1.4 Thesis Structure	7
Chapter 2 System Architecture	8
2.1 Hardware Specifications	8
2.1.1 Motors.....	9
2.1.2 Sensor	10
2.1.3 Central Control Computer	14
2.1.4 Power Supply System.....	16
2.2 Software Architecture	18
2.2.1 Overview	18
2.2.2 Laser Range Finder Layer	20
2.2.2.1 Interpolation	20
2.2.2.2 Preprocessing	22



2.2.3	Instruction Layer.....	23
2.2.3.1	Speech Recognition.....	24
2.2.3.2	Conversion	24
2.2.4	Neural Network Model.....	25
2.2.5	Post-processing Layer	26
2.2.5.1	Speed Adjusting Function	26
2.2.5.2	Halting Counter	27
Chapter 3	Training Data Set	29
3.1	Instruction.....	29
3.2	Basic Instruction Sets	32
3.3	Human-Controlled Navigating Records	36
3.3.1	Database.....	36
3.3.2	Teleoperation Program	37
3.4	Features and Advantages	39
Chapter 4	Training and Experiments.....	41
4.1	Training Models.....	41
4.1.1	Implementation.....	41
4.1.2	Validation.....	42
4.1.3	Monitors	43
4.2	Experiments and Results.....	45
4.2.1	Simulation.....	45
4.2.2	Real Environment.....	45
4.2.3	Interpolation	46
4.2.4	Comparisons	52
Chapter 5	Conclusions and Future Works.....	55

REFERENCE57
VITA62



LIST OF FIGURES

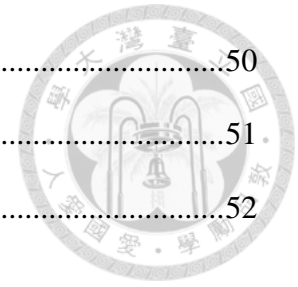


Fig. 1.1.1 Navigating under unknown indoor environment	2
Fig. 1.1.2 Example of expressing a navigation path by different instructions	3
Fig. 2.1.1 iCeiRA warehouse robot ‘Penguin’	8
Fig. 2.1.2 Servomotors and motion controllers	9
Fig. 2.1.3 Different environment structures observed from laser range finder	11
Fig. 2.1.4 Two series of Laser Range Finder being used in our research	14
Fig. 2.1.5 Control diagram of navigation system	16
Fig. 2.1.6 The overall circuit diagram implemented onboard	17
Fig. 2.1.7 DR-UPS40 by Mean Well.....	18
Fig. 2.2.1 Structure of our navigation program	19
Fig. 2.2.2 Executing process of our navigation system	20
Fig. 2.2.3 Modified saturation function	23
Fig. 2.2.4 Structure of neural network model.....	25
Fig. 3.1.1 Ten paths corresponding to ten classes of simple instructions.....	31
Fig. 3.1.2 Example of executing a complex instruction	32
Fig. 3.2.1 Process of generating a complete instruction sentence	33
Fig. 3.3.1 Example of the recording files	37
Fig. 3.3.2 Control method of <i>teleop_twist_keyboard</i>	38
Fig. 3.3.3 Control keys of our teleoperation program	39
Fig. 4.1.1 Loss monitor and validation monitor	44
Fig. 4.2.1 Floor plans of Building for Research Excellence	47
Fig. 4.2.2 Screenshot of the demonstration video	48
Fig. 4.2.3 Experiment results of Class 1 and Class 2 instructions.....	49

Fig. 4.2.4 Experiment results of Class 3 and Class 4 instructions.....50

Fig. 4.2.5 Experiment results of Class 5 and Class 6 instructions.....51

Fig. 4.2.6 Experiment results of Class 7 and Class 8 instructions.....52



LIST OF TABLES



Table 2.1-1 Technical specifications of Hokuyo UTM-30LX	12
Table 2.1-2 Technical specifications of Hokuyo URG-04LX	13
Table 3.1-1 Class of simple instructions.....	30
Table 4.2-1 Success rate of each basic instruction	48

Chapter 1 Introduction

1.1 Problem Statement



Suppose you are attending an interview in an office building. During the break you need to go to the restroom. However, since it is the first time you have been to this location, you do not know the direction. You might look for a floor plan or signs to find your way, or ask someone for directions. In the latter case, you may receive straightforward instructions, such as ‘You just go down this aisle and turn left at the second corner, and you will see it.’ Even though you had never walked along this path, nor had you seen the map, you are still able to reach the destination by following the instruction. In our daily life, we often ask for directions to find out how to get to specific locations. Humans possess the ability to navigate through unfamiliar environment according to simple instructions. By contrast, can robots achieve the same thing?

To execute a variety of tasks, service robots often need to have the ability of navigating and avoiding obstacles under different environments. Many methods have been developed to achieve this, including the effective simultaneous localization and mapping (SLAM) algorithms. Although these kinds of algorithms can make mobile robots navigate to destinations properly, they all depend on the understanding of the entire environment. In other words, robots need to construct the map of environment before starting to conduct their missions. On the other hand, exploration algorithms let robots able to navigate and construct maps under unknown environments. However, these algorithms cannot command robots to explore specific locations. Therefore, there has not been any algorithm that can have robots move to desired locations to execute tasks right after they enter a new, unknown space.

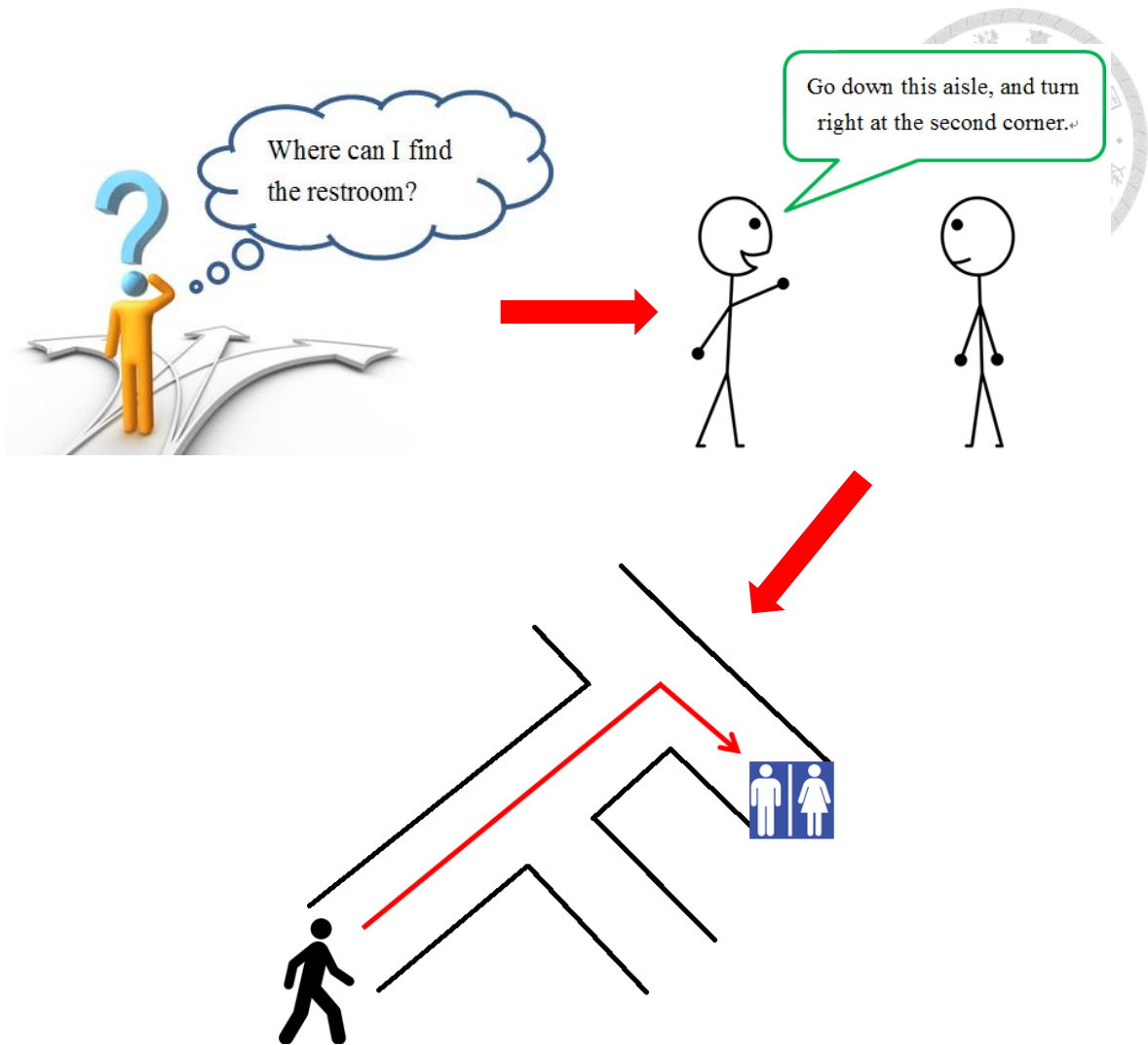


Fig. 1.1.1 Navigating under unknown indoor environment

In addition to the necessity of building maps for navigation, to let robots understand the navigating instructions from human, the efficient communication between human and robots is also a critical issue. Maps that robots use for localization and navigation, such as occupancy grid map, lack semantic information. Meanwhile, it is not easy to express natural language by mathematical models. To solve these problems, researches have been done to construct semantic maps for users. Robots classify their positions into different locations, such as corridors, halls or room 302, adding semantic information to the maps. Other researches establish probabilistic models for the instructions written in commonly verbal expressions, helping the control

system of robots handle the uncertainty and ambiguity inside human verbal instructions. Nonetheless, due to the variety and complexity of verbal instructions, it is hard to express all the possible patterns of instructions in mathematical models. For example, we can clearly see that the following two instructions refer to the same navigation path according to the map shown in Fig. 1.1.2, although they are expressed in such different ways:

- Go straight to the end, turn right, and then enter the second room on your left hand side.
- You just turn right at the end of this aisle, and then turn left at the second entry.

It is proper to claim that there are thousands of ways to express a same navigation path, and building models for all of them is impossible. Therefore, constructing an efficient algorithm for mobile robots to understand instructions received from human and execute navigation is never a simple work.

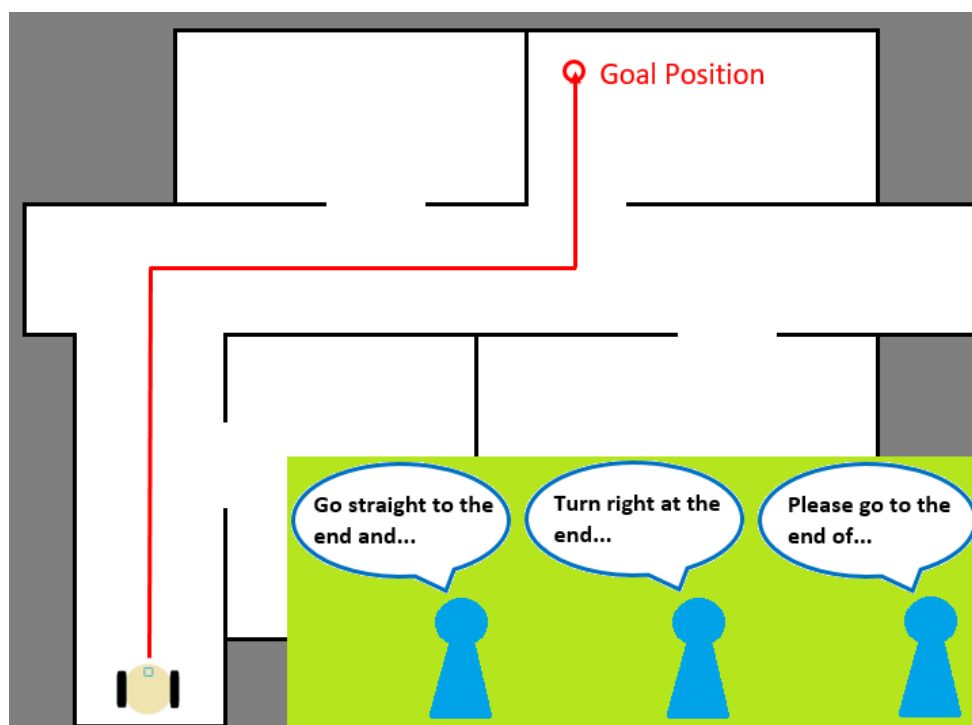
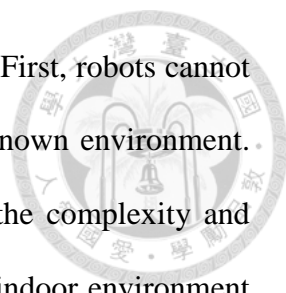


Fig. 1.1.2 Example of expressing a navigation path by different instructions



The problem we want to solve in our research has two aspects. First, robots cannot navigate to a certain location before constructing a map of the unknown environment. Second, it is hard for robots to follow human commands due to the complexity and variety of verbal instructions. Therefore, our question is: Given an indoor environment which is unknown to a mobile robot, can the robot navigate to a certain location by following human verbal instructions?

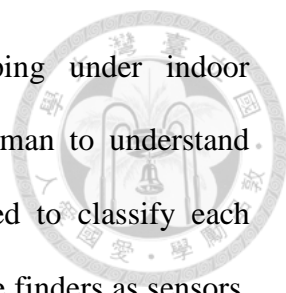
Before we describe the detailed objectives of this research and our method for the above question, we will first discuss some previous researches related to this topic in the next section.

1.2 Related Works

Moving to a specific location according to semantic instructions is a very intuitive way of navigation for mobile robots. However, this is never an easy task. To have robots efficiently use semantic information, a method using topological-semantic-metric map and Bayesian models is proposed to construct semantic maps for human-like navigation [1]. The direction of moving is computed from probabilistic models, and obstacles are avoided during semantic navigation. This research shows a great success in navigating robots to goal positions using symbolic descriptions.

An abstract map is implemented to represent unseen environments for mobile robots to navigate using only symbolic language phrases [2]. In addition, another research aims to plan semantic paths for human by integrating multiple sensors and using results from simultaneous localization and mapping (SLAM) algorithms [3]. These researches all try to bridge the gap between verbal instructions human use for instructing and mathematical models machines use for executing tasks.

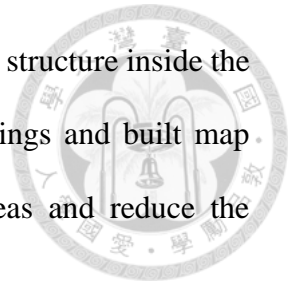
To construct efficient communication between human and robots, many researches



use the methods of machine learning to do semantic mapping under indoor environments, classifying positions into different locations for human to understand easily. A supervised learning algorithm—Adaboost, has been used to classify each position into corridor, room or doorway [4]. They use 2D laser range finders as sensors, and have the system recognize environmental shapes around the position. Some researches afterward improve the performance of it. K-means and Learning Vector Quantization methods, as well as the Markov model have been used to improve the classification rate of door [5]. A learning algorithm using the classification results from SVM and CRF is proposed, and experiments have been conducted on various environments to demonstrate its performance over real-world task [6]. In addition, different kinds of sensors are used to classify the location more precisely. A place categorization system is built upon convolutional network, and its accuracy has been evaluated using 3 different types of cameras [7]. The convolutional neural network is also applied to classify places, where LIDAR sensor is used to create occupancy grids data [8]. These efforts are all successful in adding semantic information to known or new environments.

While moving to unknown places, predicting the location we are about to enter helps us decide the next action to be taken. For example, if we want to go to room 305, and we have seen room 301, 302 sequentially, we may assume that our direction is correct, and we need to walk through the aisle and pass by two more rooms before seeing our destination. This predicting ability allows us to avoid wasting time on searching more information about the environments, such as entering room 304 to check whether it is the correct one. Researchers implement this predicting ability on mobile robots to improve the performance of mapping, localization and navigation. An algorithm called P-SLAM is introduced to look-ahead mapping [9]. This algorithm uses

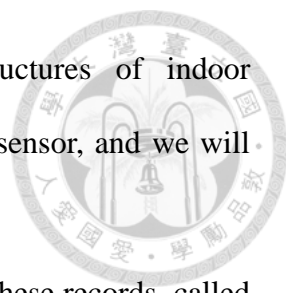
the built map of explored regions to decide whether there is a similar structure inside the unexplored regions. Besides, similarities between current surroundings and built map have been used to actively predict close loops in unexplored areas and reduce the uncertainty during exploration [10].



1.3 Research Objective

In this research, our motivation is to make the robot ‘understand’ human verbal instructions used in navigation instructions, and move to the corresponding destinations without having any pre-knowledge about the indoor environment. We now specify the detailed objectives we aim to achieve, as well as the methods we plan to use.

- We design a navigation system, which can be applied on our mobile platform. This system receives verbal instructions from humans, and controls the robot to navigate through unknown environments in both simulation and real world.
- We use the method of machine learning to construct the main body of our navigation system. The neural network model is trained to learn the way human navigate according to some instructions.
- The neural network model takes instructions and data acquired by sensor as its input, and outputs moving and rotating velocity commands to the mobile robot, instead of local or global goal positions in the environment.
- We give restrictions to the instructions applied in this research. They need to be a certain type of instructions, and they should be legal. A legal instruction is defined to be an instruction that can be successfully execute under the given environment. For example, if you cannot turn right ahead, then an instruction telling you to turn right at the front is not legal. We will explain the type of instructions we consider in Chapter 3.

- 
- Sensor we choose should be able to recognize the structures of indoor environments. We choose to use 2D laser range finder as our sensor, and we will explain our reasons in Chapter 2.
 - We use navigation records to train our neural network model. These records, called human-controlled navigating records, are generated by manually control robots to navigate by different people.
 - We train our model using data collected in simulation, and the navigation system should be able to apply in real environment.

By achieving the above objectives, we aim to give a proper solution to the stated question.

1.4 Thesis Structure

In Chapter 2, we introduce the system architecture, including the specifications of hardware components we use on our experiment platform, and the structure of presented software. Chapter 3 describes the training data set we use to train our neural network model. We will also describe the categories of instructions we consider in our research, as well as our method of collecting training data.

In Chapter 4, we introduce the process of training our neural networks, and the method we use to validate the efficiency of models. Next, we describe the details of experiments conducted under simulation and real environments. We will analyze the results and do some comparisons to examine the performance of our navigation system. Finally, Chapter 5 provides conclusions to our effort, explains the contributions of this research, and discusses the future works.



Chapter 2 System Architecture

In this chapter, we first introduce the hardware we use to test our navigation system. We will describe the specification of each component on the mobile platform, including motors, sensors, control computer and the power supply system. We will also explain the reasons for choosing these components, as well as our concerns while designing this platform. In the second part of this chapter, we describe the software architecture. Our navigation system consists of several functional layers and the recursive neural network model. We will introduce the function of each layer, and how we implement these layers. The structure of our neural network model will also be discussed.

2.1 Hardware Specifications

In this research, we implement our navigation system on the warehouse robot called ‘Penguin’, which is shown in Fig. 2.1.1. This robot is equipped with two differential wheels, one laser range finder mounted at the top, an Xtion Pro Live RGB-D camera, and a central control computer. In the following paragraphs, we will describe the specifications of these components.



Fig. 2.1.1 iCeIRA warehouse robot ‘Penguin’



2.1.1 Motors

The warehouse robot has two differential wheels and two omnidirectional wheels served as passive wheels. The former are driven by two Faulhaber 4490H048B Brushless DC-Servomotors with reduction of 66:1. Each servomotor is connected to a corresponding MCBL 3006 Motion controller. The motion controllers are connected via a RS232-USB signal cable to the CPU's USB port.

We set the maximum speed of two motors to be 10000rpm via motion controllers. The gear ratios of two differential wheels are both 3, thus the overall gear ratio is equal to 200. Radius of the differential wheel is 0.075m. Therefore, the theoretical maximum velocity of our robot is approximately 0.4m/s. However, due to robot's weight the maximum velocity will be smaller in practice. This specification is used when collecting training data and training our neural network models.

The wheel odometry is attached on the servomotor. We are not completely certain which kind of encoder has been mounted on the servomotors, but the documentation suggests the servomotor model type can co-operate with two-channel or three-channel optical or magnetic encoders.

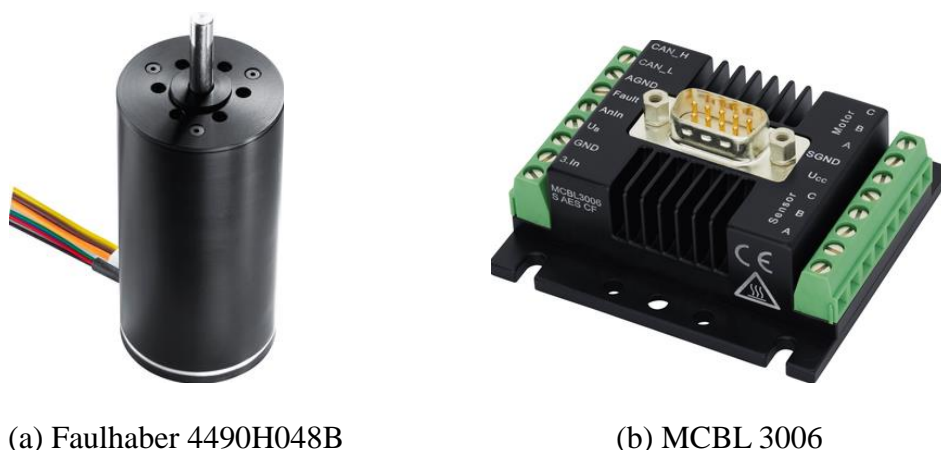


Fig. 2.1.2 Servomotors and motion controllers

2.1.2 Sensor

We only use the laser range finder to gather information from the environment. The RGB-D camera is not used in our research. We now explain why we choose to use laser range finder as the main sensor.



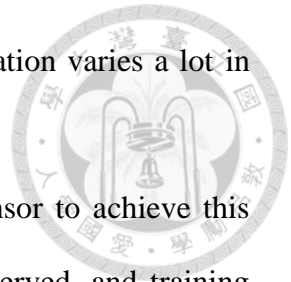
To get a well-trained neural network model, it is essential to prepare a sufficient amount of training data. However, collecting data in real environments is time-consuming and difficult, especially when we want to record the entire navigation process. Therefore, we aim to collect our training data under simulation environment. In such case, using laser range finder as sensor has many advantages.

First of all, the amount of data returned by a laser range finder is much less than that by other sensors, such as camera. As a result, speed of calculation become faster, and delay of navigation system is prevented.

Next, although the amount of data is small, it is sufficient to realize the structure of indoor environment through laser range finder. We take Fig. 2.1.3 as an example. A mobile robot is navigating through an indoor environment, and the collected sensor data is visualized in rviz, a 3D visualization tool of ROS. From Fig. 2.1.3 (a), we can tell that the robot is moving along a corridor, while Fig. 2.1.3 (b) shows that there is a crossroad in front of the robot. Thus, robots and humans can decide their actions according to the sensor measurements.

Last but not least, the effect of noise on laser range finder is small. The sensor measurements collected from simulation environments have similar performance to those from real environments. Therefore, it is appropriate to train our system using the records collected in simulation, and test it in the real world. In contrast, if we use camera as the main sensor, the training process will be completely different. We cannot

use simulation images to train out navigation model, since the situation varies a lot in real world due to several factors, such as influence of light.



To sum up, we consider laser range finder an appropriate sensor to achieve this navigation task. The structures of indoor environments can be observed, and training can be done using simulation data without considering the effect of noise.

The laser range finder we use in our research is Hokuyo UTM-30LX. Table 2.1-1 describes its technical specifications. It should be noticed that UTM-30LX has a scanning range of 270 degrees. If we place the laser range finder at the front of our warehouse robot, both sides of it will be blocked by the robot. This will result in being unable to get the full scanning range. Therefore, we mount the laser range finder on top of the mobile robot. Height of the sensor is 45cm from ground.

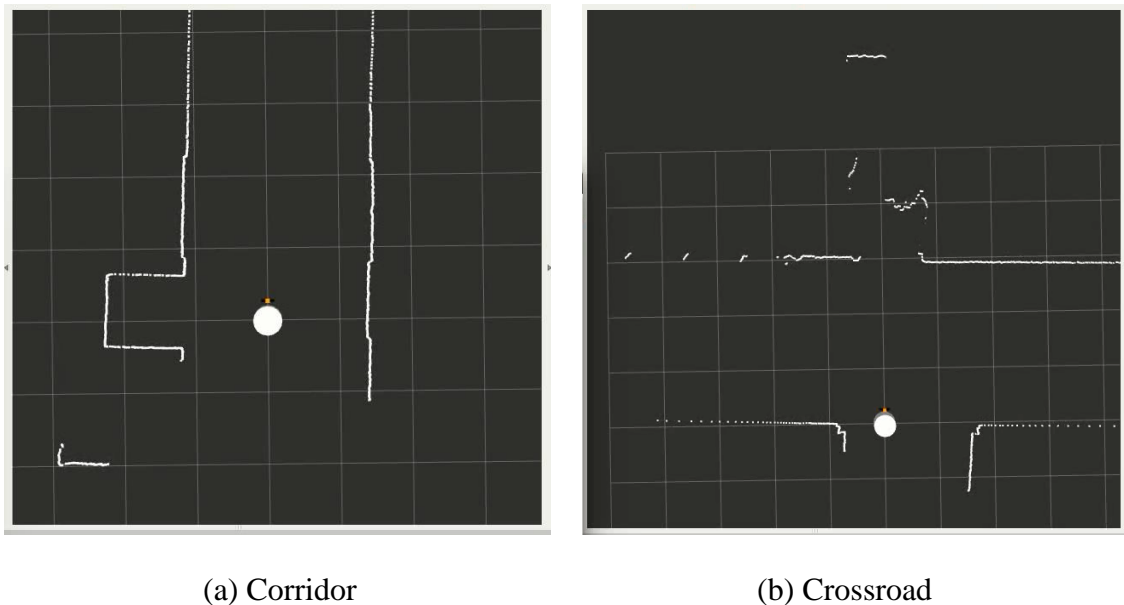



Fig. 2.1.3 Different environment structures observed from laser range finder

Table 2.1-1 Technical specifications of Hokuyo UTM-30LX



Model No.	UTM-30LX
Power Source	12VDC±10% (Current consumption: Max: 1A, Normal 0.7A)
Light Source	Semiconductor laser diode($\lambda=905\text{nm}$) Laser safety Class 1(FDA)
Detection Range	0.1 to 30m (White Square Kent Sheet 500nm or more), Max. 60m 270°C
Accuracy	0.1 to 10m:±30mm, 10 to 30m: ±50mm *1
Angular Resolution	0.25°(360°/1,440 steps)
Scan Time	25msec/scan
Sound Level	Less than 25 dB
Interface	USB2.0(Full speed)
Synchronous output	NPN open collector
Command system	Exclusively designed command SCIP Ver. 2.0
Connection	Power and Synchronous output: 2m flying lead wire USB:2m cable with type-A connector
Ambient (Temperature/Humidity)	-10 to +50 ° C, less than 85%RH(without dew and frost)
Vibration Resistance	Double amplitude 1.5mm 10 to 55Hz, 2 hours each in X,Y, and Z direction
Impact Resistance	196m/s ² , 10 times in X, Y, and Z direction
Weight	Approx. 370g (with cable attachment)

Table 2.1-2 Technical specifications of Hokuyo URG-04LX

Model No.	URG-04LX-UG01
Power Source	5VDC±5%(USB Bus Power)
Light Source	Semiconductor laser diode ($\lambda=785\text{nm}$), laser safety class 1
Measuring area	20 to 5600nm (white paper with 70nmx70nm), 240 degrees
Accuracy	60 to 1,000nm: $\pm 30\text{nm}$
	1,000 to 4,095nm: $\pm 3\%$ of measurement
Angular resolution	Step angle: approx. 0.36° ($360^\circ/1,024$ steps)
Scanning time	100 ms/scan
Noise	25 dB or less
Interface	USB2.0/1.1 [Mini B] (Full speed)
Command System	SCIP Ver. 2.0
Ambiance Illuminance *1	Halogen/Mercury lamp: 10,000Lux or less, Florescent: 6000 Lux (Max.)
Ambient temperature/Humidity	-10 to $+50^\circ\text{C}$, 85% or less (Not condensing, not icing)
Vibration resistance	10 to 55Hz, double amplitude 1.5mm each 2 hours in X, Y, and Z directions
Impact Resistance	196m/s ² , Each 10 times in X, Y, and Z directions
Weight	Approx 160g

To verify that our navigation system could be implemented on different mobile platform, we conduct experiments in simulation using mobile robot equipped with other type of laser range finder. The laser range finder we use here is URG-04LX. Table 2.1-2 describes its technical specifications. It should be noticed that the scanning range and the number of measurement steps of URG-04LX are different from those of UTM-30LX. To make our neural network model work properly, some efforts need to be done to eliminate this difference. We will describe the method we use in the software section.



(a) Hokuyo UTM-30LX



(b) Hokuyo URG-04LX

Fig. 2.1.4 Two series of Laser Range Finder being used in our research

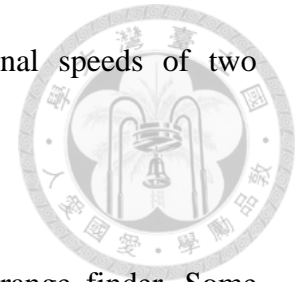
2.1.3 Central Control Computer

Our central control computer is the Jetson TX1 Developer Board. Its main function is to control the robot movement and receive sensor data. On the central control computer we run the Robot Operating System (ROS) [11]. Two major ROS nodes are used to achieve such tasks.

- *ros_control* [12]

In brief, it controls the rotation of two wheel motors. Our navigation program, as well as the teleoperation program, publishes the command velocity to this node.

And then, *ros_control* calculates the corresponding rotational speeds of two differential wheels and the driving servomotors.



- *hokuyo_node* [13]

This node handles the sensor data acquired from 2D laser range finder. Some parameters in this node can be adjusted to deal with the situation of using different laser range finders, where the detection ranges and angular resolutions are both different.

In addition to these two nodes, the ROS node *teleop_twist_keyboard* is used to remotely control the robot while our navigation program is not running [14]. It serves as a teleoperation program, however in the next chapter we will mention that we design our own teleoperation program and the reason for doing that.

To conduct our experiments more efficiently, we run our navigation program on a remote computer so that we can modify our program immediately. The central control computer and the remote computer are connected to the same local area network. Through the network ROS node *hokuyo_node* publishes readings of the laser range finder to the remote laptop, and after computation our navigation program publishes the command velocity back to *ros_control* node. Our program is packaged in a ROS node, and the exchange of messages is achieved through ROS topics.

The local area network can be provided by a cellphone. Fig. 2.1.5 shows the control diagram of our navigation system.



Fig. 2.1.5 Control diagram of navigation system

2.1.4 Power Supply System

The electric circuit used on-board consumes exclusively DC current, which means that any AC flow should be converted to DC flow previous to usage, and thus we deploy a DC Power Supply unit to import appropriate voltage and current flow. Normally the power system is supplied no more than 30V and 6A. If the supplied current exceeds that upper bound, fuse and UPS may serve as buffers and safety measure. The power system is designed under the following prerequisites:

- Safety first
- Ease of maintenance. All input/output ports are located at the rear of the robot.
- The circuit should be as precise and succinct as possible containing only necessary components.
- When connected to external power supply, the circuit works as one single close-loop. Such design ensures that the batteries are safely and efficiently charged and discharged.



- When the external power source is unplugged, there should be two independently functional close-loops circuits. One loop supplies power exclusively to the computer, while another loop to the motors and the controllers.
- Ensure that the batteries would not discharge and charge simultaneously, making sure to extend the batteries life-span.
- An emergency stop button is placed in the motor's power supply circuit to cut the power toward the motor if the robot runs into problems. Likewise, a toggle switch is placed at the computer's power circuit for ease of maintenance.
- The circuit implementation needs to follow industrial standards and convention.
- Voltmeters and Ammeters are placed to monitor on-board power supply level in real-time scale.

Following the mentioned demands, the designed circuit is shown in Fig. 2.1.6.

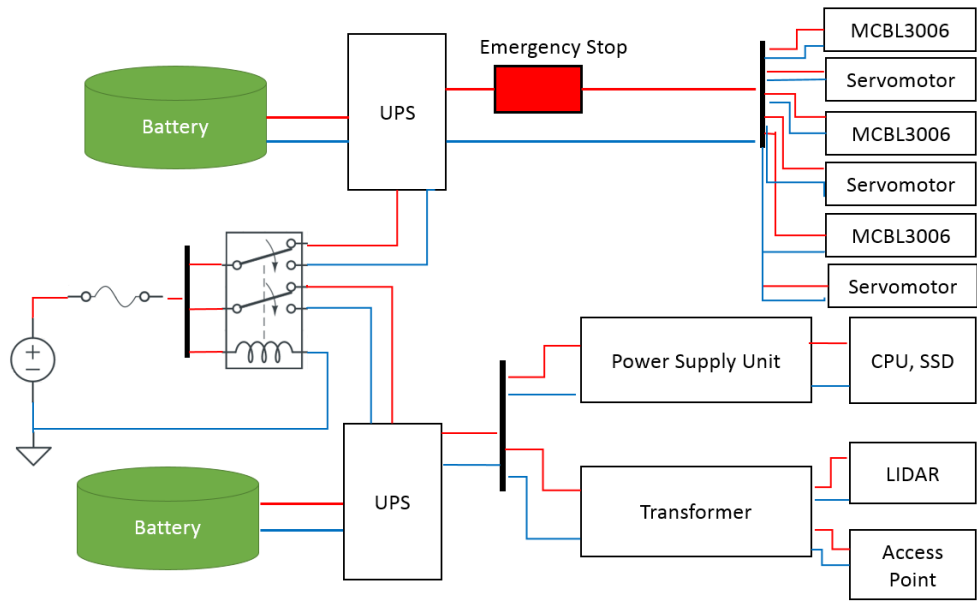


Fig. 2.1.6 The overall circuit diagram implemented onboard

We use relay as an automated and analogue switch to link and break both loops of the circuit. Once an external power source is connected, the current pass through the main switchboard, where all grounds are connected with the physical ground, while two

live wires supply the current for sub-loops with roughly the same voltage. For each loop, the first node to connect with the main power supply is located at the Uninterrupted Power Supply (UPS). By connecting in a redundant manner, the UPS not only serves as an intermediary hub for supplying the load, but also to charge and discharge the batteries, hence playing an intermediary and buffer role of power regulator between the main power source, the battery and the load. Considering the reliability, robustness, and durability of such key role, we utilized one DR-UPS40 for each sub-circuit.



Fig. 2.1.7 DR-UPS40 by Mean Well

2.2 Software Architecture

2.2.1 Overview

Fig. 2.2.1 shows the overall structure of our navigation program. It is composed of one neural network model and several processing layers. We will describe functions of these components later.

Fig. 2.2.2 shows the executing process of our navigation system. At a certain time step, the navigation system takes the current laser range finder readings, as well as the instruction given by user as its input. After computing it outputs the desired moving and rotating velocity of mobile robot. The robot will navigate in the given speed for a time period, entering a different position. This leads to different readings of the laser range

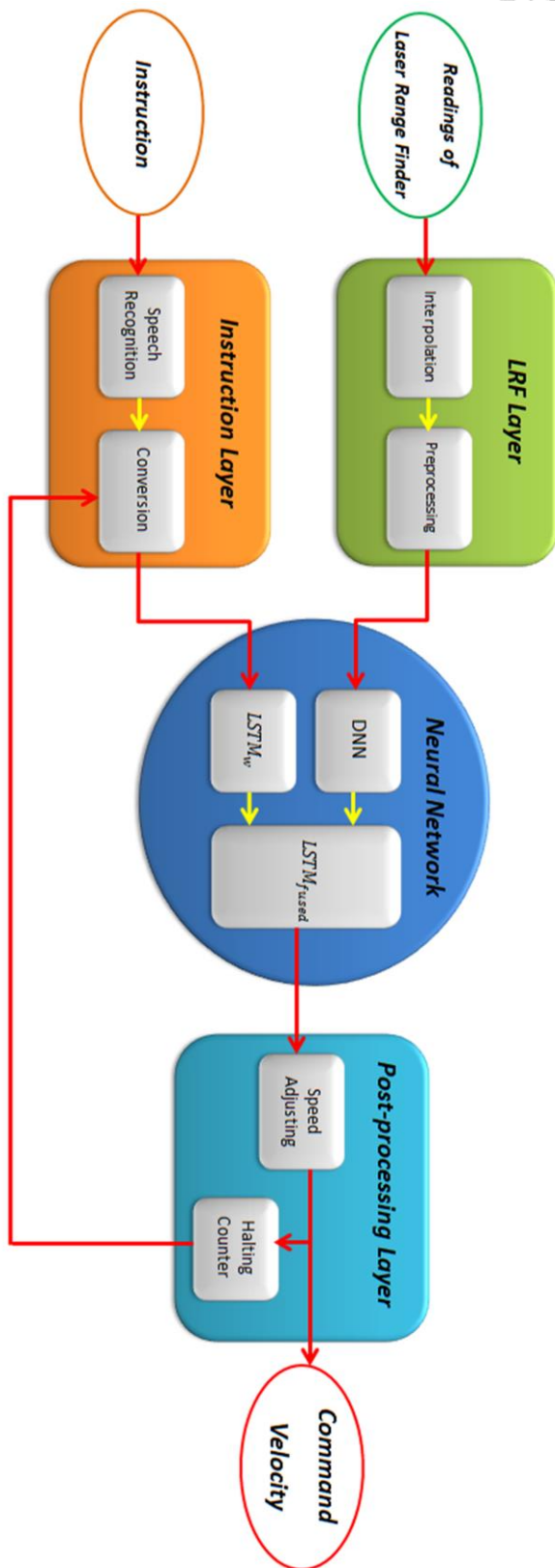


Fig. 2.2.1 Structure of our navigation program

finder at the next time step. Thus, the model will take the new readings and the same instruction as its new input, and decide velocity of the next time step.

In the following sections, we will introduce the function of each layer as well as the structure of our neural network.

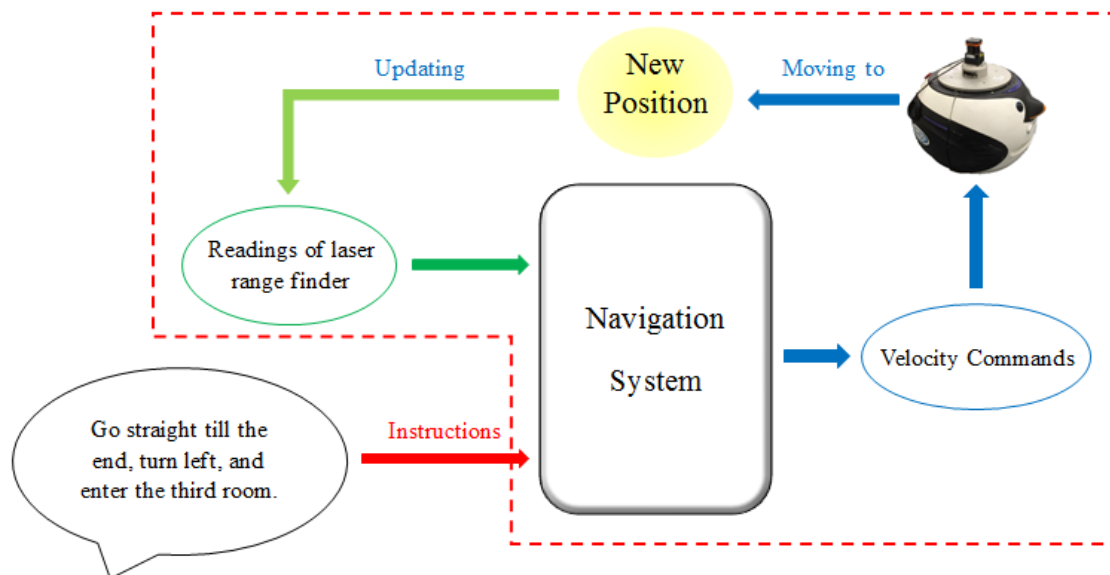


Fig. 2.2.2 Executing process of our navigation system

2.2.2 Laser Range Finder Layer

This layer consists of two stages: the interpolation stage, and the preprocessing stage.

2.2.2.1 Interpolation

We use Hokuyo UTM-30LX as our main sensor in this research. However, in order to expand the usage of our system to different mobile robots, we need to consider the cases when different laser range finders are used.

Different laser range finders have different detection ranges and angular resolutions. If our system use the entire scanning range of UTM-30LX, changing the sensor to a laser range finder with smaller scanning range, such as Hokuyo URG-04LX, will be difficult. The reason is that even if we disregard the difference between their

maximum measuring distances, getting measurements from angles that are beyond the scanning range of URG-04LX is impossible. Therefore, we only use 220 degrees of UTM-30LX's scanning range. Since this value is larger than 180, objects behind the left part and right part of the robot can be detected. It is thus sufficient for our system to recognize the environment structures using such scanning range.

However, due to the difference between angular resolutions, sometimes we cannot get measurement from a particular angle using other sensors. Interpolation stage is thus designed to solve this problem. As its name implies, in this stage we use the method of linear interpolation to calculate the desired measurements. Details of our method are described as follows:

Consider two different laser range finders A and B. Assume B is the original sensor we use on the mobile robot, and A is the substitute. Now, we want to get the measurements of sensor B using sensor A. Since some angle measurements of B cannot be obtained by A, we apply the following interpolation method:

We define A_j and B_k to be the measurements of A and B. Here

$$0 \leq j < \text{number of measurement steps of A} \quad (2.2-1)$$

$$0 \leq k < \text{number of measurement steps of B} \quad (2.2-2)$$

For example, the number of measurement steps of UTM-30LX is 1440, while that of URG-04LX is 683. We define A_{min} and B_{min} as the minimal measuring angles of A and B. For instance, the minimal measuring angle of Hokuyo UTM-30LX is -135° . Next, we define A_{diff} and B_{diff} to be angular resolutions of A and B. B_{steps} is the number of measurements we want to obtain, while B_{start} is the starting step. For example, if we take Hokuyo URG-04LX as sensor B, and we want to discard the leftmost and rightmost 30 steps of the measurements, then we will let $B_{start} = 30$ and

$B_{steps} = 623$.

Now, for all the steps we consider, we first calculate their angles:

$$angle = B_{min} + (B_{start} + i) * B_{diff}, 0 \leq i < B_{steps} \quad (2.2-3)$$

Here every angle is guaranteed to lie in the scanning range of sensor A. Next, the following two variables are calculated for each of the steps:

$$SAMP_i = \left\lfloor \frac{angle - A_{min}}{A_{diff}} \right\rfloor, 0 \leq i < B_{steps} \quad (2.2-4)$$

$$INTER_i = (angle - A_{min}) \bmod A_{diff}, 0 \leq i < B_{steps} \quad (2.2-5)$$

Finally, given all the measurements A_j obtained from sensor A, we calculate the measurement for each step we consider by the following equation:

$$B_i = A_{SAMP_i} + (A_{SAMP_{i+1}} - A_{SAMP_i}) * \frac{INTER_i}{A_{diff}}, 0 \leq i < B_{steps} \quad (2.2-6)$$

By applying the linear interpolation method, we obtain the measurements B_i of sensor B from measurements A_j of sensor A. We do some experiments to verify the effectiveness of our interpolation stage. The results will be discussed in Chapter 4.

2.2.2.2 Preprocessing

Before applying input data to the neural network, we preprocess data obtained by laser range finder to improve the performance of our neural network.

First, to deal with the difference in maximum measuring distances between different laser range finders, we set a threshold value to the received measurements. Readings exceeding 5 meters will be modified to 5. That is, objects 5 meters away from the sensor will not be detected due to this saturation function. However, to create greater difference between empty spaces and barriers, we change the modified value from 5 to 10 meters.



To sum up, we modify the input data as follows:

$$f(x_i) = \begin{cases} x_i, & x_i < 5 \\ 10, & x_i \geq 5 \end{cases} \quad (2.2-7)$$

Here x_i represents the sensor measurements.

$$0 \leq i < \text{number of measurement steps} \quad (2.2-8)$$

Fig. 2.2.3 explains our saturation function more clearly.

After applying the modified saturation function, we normalize the input data using the following equation:

$$\hat{x}_i = \frac{f(x_i)}{\sqrt{\sum_{k=1}^N f(x_k)^2}} \quad (2.2-9)$$

We use the normalization function provided by Scikit-learn [15]. Finally, the preprocessed data is fed into our neural network.

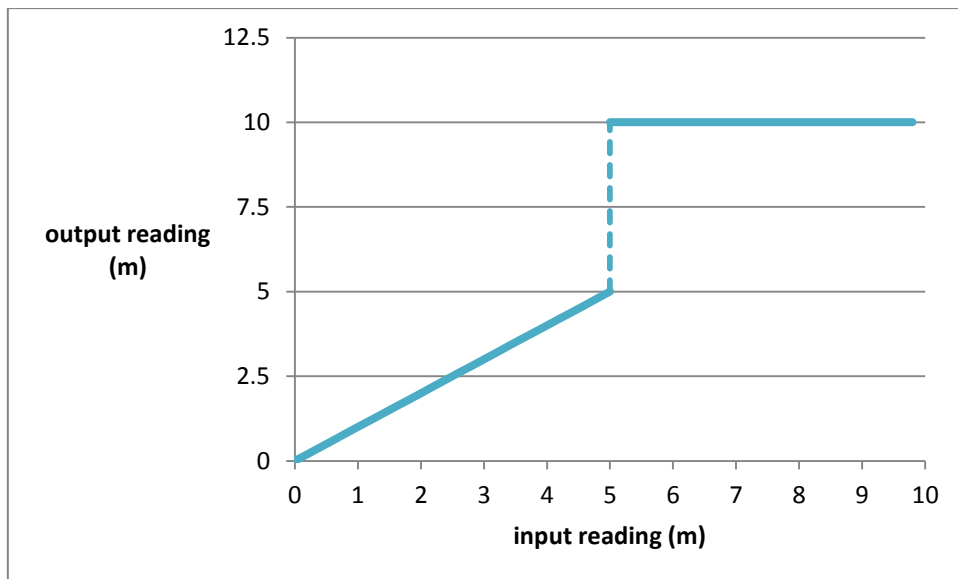


Fig. 2.2.3 Modified saturation function

2.2.3 Instruction Layer

This layer also consists of two stages: the speech recognition stage, and the conversion stage.



2.2.3.1 Speech Recognition

We implement the speech recognition stage to make our robot receive human instructions directly. Users can command the robot verbally instead of typing instructions into the remote computer.

In this research, we use the python package, SpeechRecognition, to implement our speech recognition function [16]. This package supports seven speech recognition engines and APIs, namely CMU Sphinx, Google Speech Recognition, Google Cloud Speech API, Wit.ai, Microsoft Bing Voice Recognition, Houndify API, and IBM Speech to Text. We choose to use Google Speech Recognition due to its ease of implementation and high recognition accuracy.

Verbal instruction inputted will be converted into text in this stage.

2.2.3.2 Conversion

The conversion stage first splits the input instruction into several shorter ones to handle the situation where the given instruction is too long and complex. The split instructions are called **simple instructions**, and they are inputted into the neural network sequentially. We will explain the method of splitting in the next chapter.

Before inputting a simple instruction into the neural network, the conversion stage converts each word in the instruction into vector. A pre-trained Global Vectors for Word Representation (GloVe) model is used here, and we decide to use a 100-dimension vector to represent one word [17]. We use the concept of word vectors to have our program consider the semantic information contained in user instructions.

For each simple instruction, we sequentially fed its word vectors into our neural network. The set of all word vectors of a simple instruction is called the **sentence vector** of that instruction.



2.2.4 Neural Network Model

Fig. 2.2.4 shows the structure of our neural network model. It consists of two stages. The red box in Fig. 2.2.4 indicates the first one.

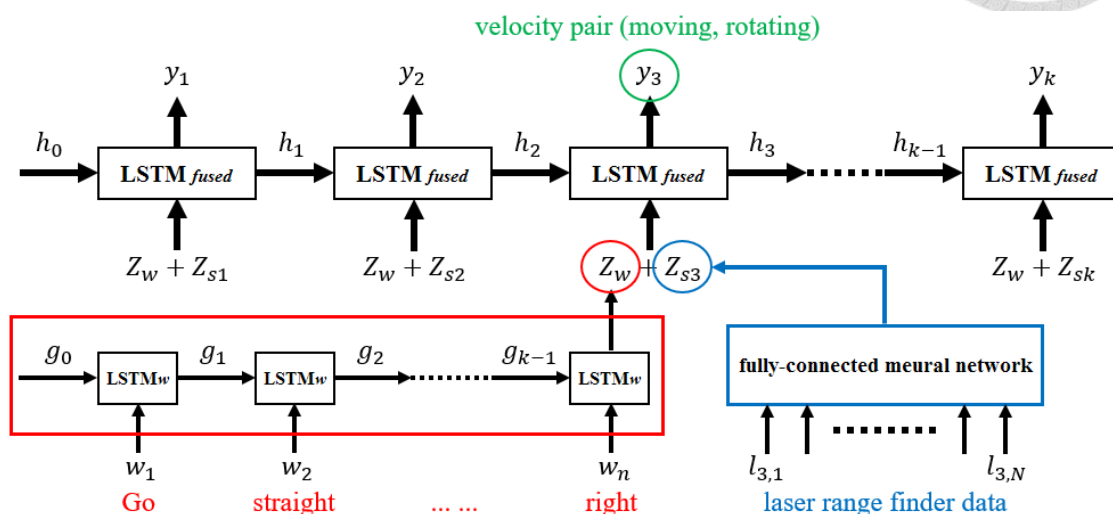


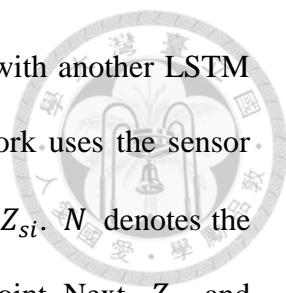
Fig. 2.2.4 Structure of neural network model

The first stage is a Long short-term memory (LSTM) layer $LSTM_w$. It takes the sequence of word vectors (w_1, w_2, \dots, w_n) , which is generated from a simple instruction, as input, and outputs the instruction vector Z_w to the second stage. Here n is variant since the length of each simple instruction is not fixed.

$LSTM_w$ serves as an instruction classifier. We aim to use vector Z_w to represent the class of the given simple instruction. We define ten **basic instruction classes** in our research. Each simple instruction will be classified into one of these classes, and the navigation system will decide the actions to take according to the obtained class. We will give a definition of basic instruction class, and explain how we classify instructions into ten classes in Chapter 3.

Since we aim to classify simple instructions into different classes, Softmax activation function is used in $LSTM_w$:

$$\sigma(\mathbf{Z})_i = \frac{e^{z_i}}{\sum_{k=1}^N e^{z_k}} \quad (2.2-11)$$



The second stage combines a fully-connected neural network with another LSTM layer $LSTM_{fused}$. At time step i , the fully-connected neural network uses the sensor data $(l_{i1}, l_{i2}, \dots, l_{iN})$ from laser range finder to calculate its output Z_{si} . N denotes the number of measurement steps, and l_{ik} represents the k^{th} sampling point. Next, Z_w and Z_{si} are concatenated and fed into $LSTM_{fused}$. The second LSTM layer will then calculate the velocity pair (*moving, rotating*) for controlling the mobile robot.

As for the activation functions, we choose to use ReLU in the fully-connected neural network, since all the input data from laser range finder are positive:

$$\sigma(\mathbf{Z})_i = \max(0, z_i) \quad (2.2-11)$$

In $LSTM_{fused}$, we simply choose linear activation function since the outputs represent speeds and can be either positive or negative.

It should be noticed that the output of our neural network model will affect the input of it. That is, different velocity commands will lead the robot to different positions, and thus readings of the laser range finder will be different.

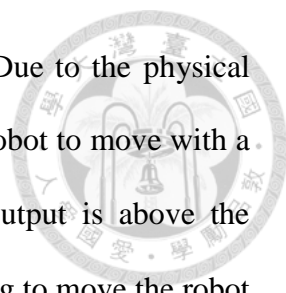
2.2.5 Post-processing Layer

The post-processing layer is composed of two functions: the speed adjusting function, and the halting counter.

2.2.5.1 Speed Adjusting Function

The speed adjusting function consists of two sub-functions. It is unlikely that the neural network will output pure zero. Therefore, when the output is below a certain value, we assume that the neural network is intending to stop the robot. We define the minimum value v_{min} to be 10^{-3} . The first sub-function changes either the moving velocity or rotating velocity to 0 when its absolute value is smaller than v_{min} .

The second sub-function prevents the navigation program from outputting



velocities which are too slow for the mobile robot to move with. Due to the physical constraints, such as friction, in real world, it is not possible for the robot to move with a speed which is below some certain value. However, when the output is above the minimum value v_{min} , we assume that the neural network is intending to move the robot. Therefore, we need to increase the output velocity when the output lies in this range in order to make the robot move. We define the minimum moving velocity v_{move} to be $2 * 10^{-2}$. When the absolute value of output velocity is above v_{min} and below v_{move} , the second sub-function will pull it up to v_{move} . This prevents the robot from getting stuck at the same position while the network is telling the robot to move.

To sum up, the speed adjusting function can be expressed by the following equation:

$$f(v) = \begin{cases} 0, & |v| \leq v_{min} \\ \text{sgn}(v) * v_{move}, & v_{min} < |v| \leq v_{move} \\ v, & \text{otherwise} \end{cases} \quad (2.2-12)$$

Here $v_{min} = 10^{-3}$ and $v_{move} = 2 * 10^{-2}$, and v represents the moving or rotating velocity outputted by the neural network model.

2.2.5.2 Halting Counter

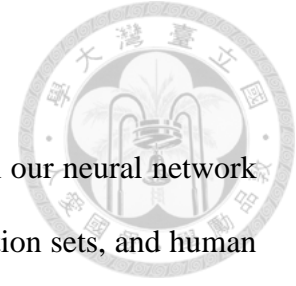
This counter decides when the program stops. When completing the navigation process, the mobile robot should stop at its final position. Therefore, if the neural network model is trained well, our program should constantly output $(moving, rotating) = (0, 0)$ after the robot reach its destination. We can assume that the robot has completed its navigation if the program continuously outputs $(0, 0)$ for a number of time steps.

The halting counter counts the number of continuous zero outputs. If this number is above a certain value, the counter sends a stopping signal back to the neural network

layer and the conversion stage inside the instruction layer. The former will clear its memory, preparing to execute a new navigation task. The latter will load the next instruction into the network model. After that, the counter is set to zero, and the program continues the navigation. If there is no instruction left, the entire program stops.

In this research, we set the number of time steps to stop the process to be 50, which is approximately 5 seconds.

Chapter 3 Training Data Set



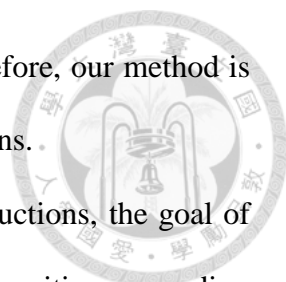
In this chapter, we will describe the training data set we use to train our neural network model. The training data set is composed of two parts: basic instruction sets, and human controlled navigating records. We will first describe the type of instructions we consider in our research. Next, we explain our method of generating the instruction training data set and that of collecting the navigation records. We will also introduce the teleoperation program we design for this research, and discuss the advantages of using such data structure for our training database.

3.1 Instruction

In the previous chapter, we have mentioned that we use laser range finder as the main sensor of our system. Despite the advantages we discussed, some information cannot be acquired by the laser range finder. For example, it is not easy to achieve object recognition using such sensor. Room number cannot be detected, either. Thus, we have to specify the instructions we consider in our research more clearly.

In short, we only consider instructions describing the structure of indoor environments. That is, aisles, corners and rooms are to be discussed. Higher level semantic instructions are not considered, such as room or aisle number, which needs more sensors to get enough information.

In our application, most of the given instructions are short and simple, like ‘Turn right, and then go straight to the end of the aisle.’ However, for cases with complex indoor environments, the longer the path that instructors command the robot to move, the more complex their instructions become. We consider it difficult for the robot to complete such complicated instructions, since it is not possible for the neural network to



‘remember’ so much information contained in the instruction. Therefore, our method is to split the original instruction into several shorter, simpler instructions.

After we divide the original instruction into many simple instructions, the goal of our navigation system is to make the mobile robot move to specific positions according to these simple instructions in sequence. To decide what actions to take for each simple instruction, we classify instructions into several classes. A **basic instruction class** is defined to be a set of simple instructions that have similar meanings. Each class has a basic instruction that represents the class. Fig. 3.1.1 and Table 3.1-1 show the ten basic instruction classes we define in this research, as well as their representative basic instructions.

Path / Class Number	Representative Basic Instructions
1	Go straight to the end
2	Turn right
3	Turn left
4	Go back (turn around)
5	Turn right at the second corner
6	Turn left at the second corner
7	Turn right at the third corner
8	Turn left at the third corner
9	Go straight to the end and turn right
10	Go straight to the end and turn left

Table 3.1-1 Class of simple instructions

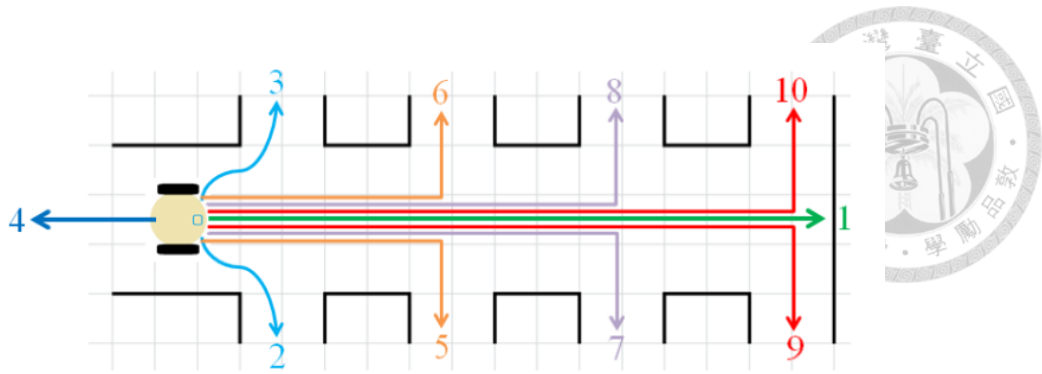


Fig. 3.1.1 Ten paths corresponding to ten classes of simple instructions

It should be noticed that the second basic instruction contains two different meanings: ‘Turn right immediately’, and ‘Turn right at the first corner’. So is the third instruction.

Since our application is under indoor environments, and we use laser range finder as the sensor, we assume that these ten basic instructions suffice to describe the sequential steps of conducting a complete navigation. That is, we can use these basic instructions to construct all the legal instructions we consider in this research. Therefore, all the simple instructions split out from the original instruction can be classified into one of these basic instruction classes.

Now we explain our method of splitting user commands into simple instructions. Since each basic instruction contains enough information, it is not possible that a person may say such a long sentence containing more information than the basic instruction without a pause. Therefore, we simply use punctuation marks to split sentences into simple instruction.

The idea of splitting instruction into simple ones, converting each word into word vector to construct the sentence vectors for each simple instruction, and executing them sequentially can be described by Fig. 3.1.2. After completing the execution of some simple instruction, the robot will enter a different position from its previous point. And then, it will execute the next simple instruction to reach its next local destination. By

executing these instructions sequentially, the robot will finally get to the location indicated by the original instruction.

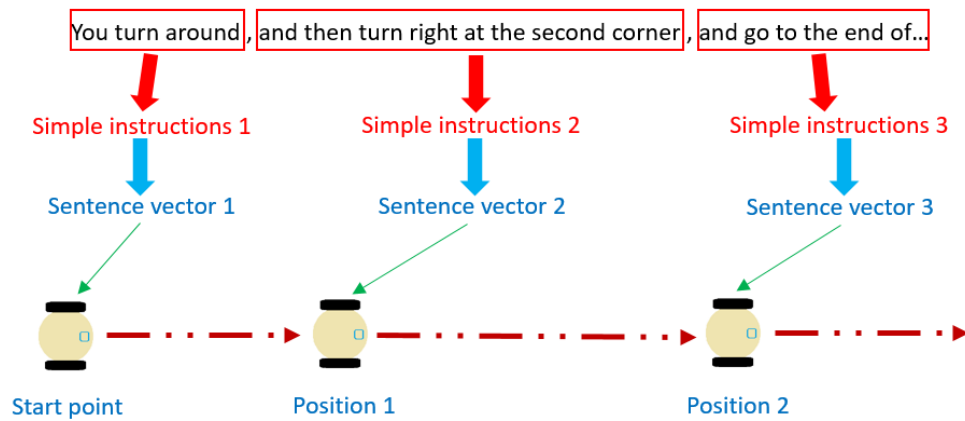


Fig. 3.1.2 Example of executing a complex instruction

We collect ten sets of simple instructions corresponding to the above ten classes to train our model. These ten sets are called **basic instruction sets**, and each of them should be a subset of their corresponding basic instruction class. In the next section, we describe how we construct these ten sets.

3.2 Basic Instruction Sets

In Chapter 2, we mention that we design a neural network model that classifies verbal instructions into ten basic instruction classes. To construct a database for training such a classification model, a straightforward approach is to collect thousands of instructions and label them manually. However, there is no online instruction database that can be directly used for training our model. Most of the commonly used databases contain high-level semantic information, such as room numbers or features that can only be captured by camera. These instructions cannot be classified into any of our instruction classes. Besides, typing numerous different instructions will be a tedious work. Therefore, we design an algorithm that can generate instructions using small collections of phrases.

To construct a sufficient training database, we first manually create sets of phrases. Each of these sets plays a different role in constructing a complete instruction sentence. While generating instructions, our algorithm chooses some of these sets, randomly picks one phrase from every chosen set, combines these phrases to form an instruction sentence, and labels the sentence according to the sets it chose. The process of generating a complete instruction sentence is described by Fig. 3.2.1. Each of the boxes represents a collection of phrases.

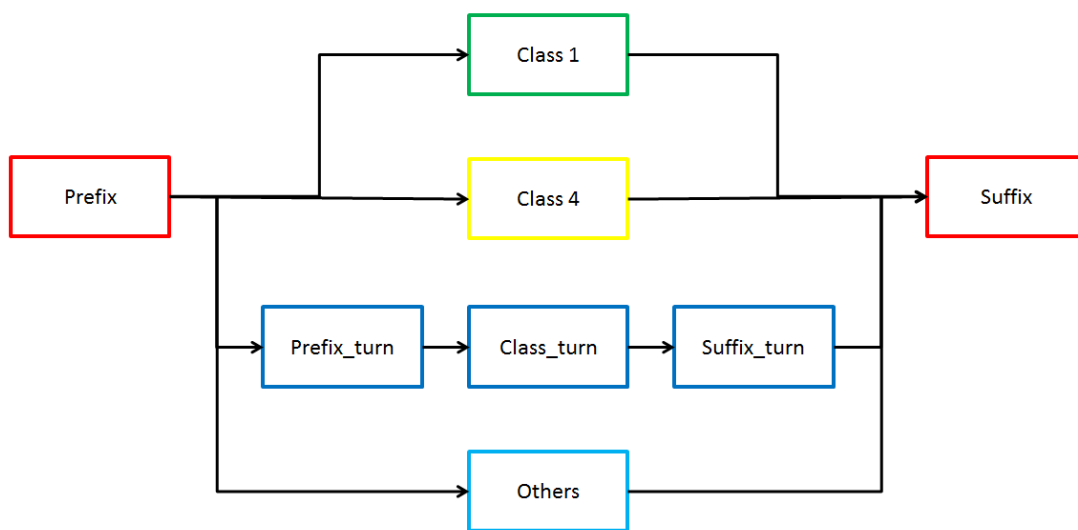


Fig. 3.2.1 Process of generating a complete instruction sentence

The sets of phrases are as follows:

1. **Prefix:** Phrases or words that can be attached to the beginning of instruction sentences, without changing the meaning of instructions.

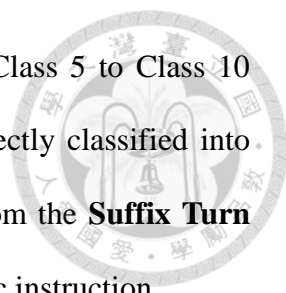
For example: ‘Please ...’, ‘Would you ...’, ‘I need you to ...’

2. **Suffix:** Phrases or words that can be attached to the end of instruction sentences, without changing the meaning of instructions.

For example: ‘... please.’, ‘... and stop.’, ‘... thank you.’

3. **Class 1:** Phrases used to construct Class 1 instruction sentences.

For example: ‘go straight to the end’, ‘walk down this aisle’

- 
4. **Class Turn:** Phrases used to construct Class 2, Class 3, and Class 5 to Class 10 instruction sentences. Phrases belonging to this set can be directly classified into Class 2 or Class 3 basic instruction, and appending phrases from the **Suffix Turn** set to their end may make them become Class 5 to Class 10 basic instruction.

For example: ‘turn right’, ‘make a left’

5. **Prefix Turn:** Phrases that can be attached to the front of phrases from **Class Turn** without changing the meaning of constructed instruction sentences.

For example: ‘go straight and’, ‘walk down this way and’

6. **Suffix Turn:**

There are two kinds of phrases that belong to this set:

- A. Phrases that can be attached to the end of phrases from **Class Turn** without changing their classification. For instance, ‘turn right’ and ‘turn right at the corner’ both belong to Class 2 basic instruction.

For example: ‘at the corner’, ‘at the crossroad’

- B. Phrases that can be attached to the end of phrases from **Class Turn** to change their classification, making them become Class 5 to Class 10 basic instruction. For instance, ‘turn right’ belongs to Class 2 basic instruction, but ‘turn right at the second corner’ belongs to Class 5 basic instruction.

For example: ‘at the second corner’, ‘at the end of this aisle’

7. **Class 4:** Phrases used to construct Class 4 instruction sentences.

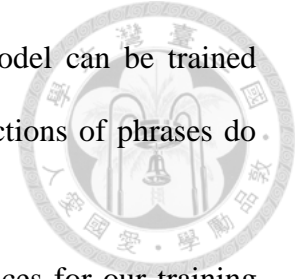
For example: ‘go back’, ‘turn around’

8. **Others:** Phrases that cannot be classified into any of the above sets.

For example: ‘go straight to the end and turn right’

It should be noticed that we convert each word in the instruction into word vector by GloVe before importing into our neural network. Since word vector represents the

meaning of that word to a certain extent, we assume that our model can be trained successfully without having seen every word. Therefore, the collections of phrases do not need to contain every possible phrase.



Now, we give some examples of generating instruction sentences for our training database, i.e., the ten basic instruction sets.

- **Prefix + Class 1 + Suffix** → Class 1 basic instruction

‘Could you’ + ‘go straight till the end’ + ‘please’ = ‘Go straight to the end’

- **Prefix Turn + Class Turn + Suffix** → Class 2 basic instruction

‘Go straight and’ + ‘make a right turn’ + ‘please’ = ‘turn right’

- **Prefix + Class Turn + Suffix Turn** → Class 5 basic instruction

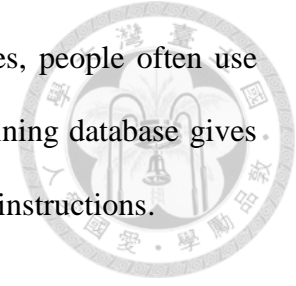
‘Please’ + ‘make a right turn’ + ‘at the second intersection’ = ‘Turn right at the second corner’

For each basic instruction set, we consider all possible combinations of sets of phrases that generate sentences for it. This method greatly increases the amount of training data for training our neural network model. Experiments show that the classification accuracy of our model, as well as the ability to handle new, unfamiliar instructions is improved using such expanded database.

It should be noticed that some of the generated sentences are not grammatically correct. For example, the sentence ‘Please turn right please’ is not proper since the word ‘please’ is used twice. However, we consider that including such sentences in our training database is legal.

Our neural network model does not need to know what kinds of instructions are grammatically correct. Its function is to classify instructions, extracting semantic information from user commands. As long as the given sentence refers to the correct

class of basic instruction, we consider it useful in training. Besides, people often use incorrect grammar when speaking. Using such sentences in our training database gives our model the ability to handle the variety and complexity of verbal instructions.



3.3 Human-Controlled Navigating Records

3.3.1 Database

A human-controlled navigating record is a set of recording sequences. Each recording sequence contains hundreds of sampling data, where every sampling data consists of the velocity of the robot and readings of its laser range finder at that sampling moment. One recording sequence is considered as one training data sequence. While collecting training data, we let a person remote robot to the destination according to a specific basic instruction, and during the navigation we collect the (*velocity, measurement*) pairs. By doing so we obtain a sequence that describe the expected behavior of a robot after receiving an instruction.

Fig. 3.3.1 shows an example of our recording files. An indicator is added to the beginning of every recording sequence, while another one is used at the end of each sequence. In Fig. 3.3.1 we use red box and blue box to annotate the starting and ending indicator. The starting indicator also shows the class of the recording sequence.

The yellow underline in Fig. 3.3.1 shows an example of a sampling data. The former part is the (*moving, rotating*) velocity pair, while the latter part is the measurements from laser range finder. After preprocessing the latter part is used as part of the input of our neural network, while the former part serves as the referenced label during training process.

```

0.240982;-0.000172308;0.0827409;2.70470;2.69911;2.6936;2.6882;2.68293;2.67778;2.67275;2.66784;2.66305;2.65837;2.65381;2.64937;2.64504;2.
0.205536;-0.00150286;0.0827481;2.70479;2.69914;2.69363;2.68823;2.68296;2.67781;2.67278;2.66787;2.66308;2.6584;2.65384;2.6494;2.64507;2.6
0.205582;-0.00113922;0.0827325;2.70496;2.69931;2.69379;2.68839;2.68312;2.67796;2.67293;2.66802;2.66322;2.65854;2.65398;2.64954;2.6452;2.6
0.000110576;-0.000190968;2.71076;2.70499;2.69934;2.69382;2.68842;2.68315;2.67799;2.67296;2.66805;2.66325;2.65857;2.65401;2.64956;2.64523
0.000102767;-0.000195656;2.71096;2.70518;2.69953;2.69401;2.68861;2.68333;2.67817;2.67313;2.66822;2.66342;2.65874;2.65417;2.64972;2.64538
0.00124415;-0.000958927;2.71138;2.70559;2.69993;2.69439;2.68898;2.68369;2.67853;2.67348;2.66855;2.66375;2.65905;2.65448;2.65002;2.64568;
5.78742e-05;0.000124655;2.71129;2.7055;2.69984;2.69431;2.6889;2.68362;2.67845;2.67341;2.66848;2.66368;2.65899;2.65441;2.64996;2.64562;2.
3.36813e-05;-7.47657e-05;2.7113;2.70551;2.69985;2.69432;2.68891;2.68362;2.67846;2.67342;2.66849;2.66368;2.65899;2.65442;2.64996;2.64562;
3.25649e-05;-8.62397e-05;2.71132;2.70553;2.69987;2.69434;2.68893;2.68364;2.67848;2.67343;2.66851;2.6637;2.65901;2.65444;2.64998;2.64564;
3.24978e-05;-8.68992e-05;2.71133;2.70555;2.69989;2.69436;2.68895;2.68366;2.67849;2.67345;2.66852;2.66371;2.65902;2.65445;2.64999;2.64565;
3.24888e-05;-8.69685e-05;2.71135;2.70557;2.69991;2.69437;2.68896;2.68367;2.67851;2.67346;2.66854;2.66373;2.65904;2.65446;2.65001;2.64566;
3.24813e-05;-8.70056e-05;2.71137;2.70558;2.69992;2.69439;2.68897;2.68369;2.67852;2.67347;2.66855;2.66374;2.65905;2.65447;2.65001;2.64567;
3.24748e-05;-8.70411e-05;2.71139;2.70561;2.69994;2.69441;2.689;2.68371;2.67854;2.67349;2.66857;2.66376;2.65907;2.65449;2.65003;2.64569;2.
3.24748e-05;-8.70411e-05;2.7114;2.70562;2.69996;2.69442;2.68901;2.68372;2.67855;2.67351;2.66858;2.66377;2.65908;2.6545;2.65004;2.6457;2.
3.24617e-05;-8.71121e-05;2.71142;2.70564;2.69998;2.69444;2.68903;2.68374;2.67857;2.67352;2.66859;2.66378;2.65909;2.65452;2.65005;2.64571;
3.24552e-05;-8.71477e-05;2.71144;2.70566;2.69999;2.69446;2.68904;2.68375;2.67858;2.67354;2.66861;2.6638;2.65911;2.65453;2.65007;2.64572;
STOP
R;3;
3.12608e-05;-9.45645e-05;3.57306;3.56535;3.5578;3.55042;3.5432;3.53615;3.52926;3.52252;3.51595;3.50953;3.50327;3.49716;3.4912;3.48539;3.
3.12538e-05;-9.46142e-05;3.57309;3.56537;3.55783;3.55045;3.54323;3.53617;3.52928;3.52255;3.51597;3.50955;3.50329;3.49718;3.49122;3.48541;3.
3.12467e-05;-9.4664e-05;3.57311;3.56539;3.55784;3.55046;3.54325;3.53619;3.5293;3.52256;3.51599;3.50957;3.5033;3.49719;3.49123;3.48543;3.
3.12397e-05;-9.47139e-05;3.57314;3.56543;3.55788;3.5505;3.54328;3.53622;3.52933;3.52259;3.51602;3.5096;3.50333;3.49722;3.49126;3.48545;3.
3.12327e-05;-9.47638e-05;3.57316;3.56545;3.5579;3.55052;3.5433;3.53624;3.52935;3.52261;3.51603;3.50961;3.50334;3.49723;3.49127;3.48547;3.
3.12256e-05;-9.48139e-05;3.57319;3.56547;3.55792;3.55054;3.54332;3.53627;3.52937;3.52263;3.51605;3.50963;3.50337;3.49725;3.49129;3.48549;
3.12186e-05;-9.4864e-05;3.57323;3.56551;3.55796;3.55058;3.54336;3.5363;3.5294;3.52266;3.51608;3.50966;3.50339;3.49728;3.49132;3.48551;3.
3.12116e-05;-9.49142e-05;3.57325;3.56553;3.55798;3.55059;3.54337;3.53631;3.52942;3.52268;3.5161;3.50968;3.50341;3.4973;3.49133;3.48552;3.
3.12045e-05;-9.49644e-05;3.57332;3.5656;3.55805;3.55066;3.54344;3.53638;3.52948;3.52274;3.51616;3.50973;3.50346;3.49735;3.49139;3.48558;
0.0499561;-0.001026;3.5738;3.56607;3.5585;3.5511;3.54387;3.5368;3.52989;3.52314;3.51655;3.51011;3.50383;3.4977;3.49173;3.48591;3.48024;3.
0.0500487;8.60791e-05;3.57387;3.56614;3.55857;3.55117;3.54393;3.53686;3.52995;3.52319;3.5166;3.51016;3.50388;3.49775;3.49178;3.48595;3.4
0.0500523;0.000126472;3.57386;3.56612;3.55856;3.55116;3.54392;3.53685;3.52993;3.52318;3.51659;3.51015;3.50387;3.49774;3.49176;3.48594;3.
0.0500497;9.57135e-05;3.57384;3.56611;3.55854;3.55114;3.5439;3.53683;3.52992;3.52317;3.51657;3.51013;3.50385;3.49772;3.49175;3.48592;3.4
0.0500472;6.64798e-05;3.57383;3.56609;3.55853;3.55113;3.54389;3.53682;3.52991;3.52315;3.51656;3.51012;3.50384;3.49771;3.49173;3.48591;3.
0.150263;-0.00167264;0.0829142;3.56646;3.55888;3.55147;3.54423;3.53714;3.53022;3.52346;3.51686;3.51041;3.50412;3.49798;3.492;3.48616;3.4
0.348235;0.0034201;3.57454;3.56678;3.5592;3.55178;3.54452;3.53743;3.53049;3.52372;3.51711;3.51065;3.50435;3.4982;3.49221;3.48636;3.48067;
0.586551;0.05568e-05;0.0828441;3.55752;3.55052;3.54322;3.53612;3.52922;3.52252;3.51592;3.50952;3.50322;3.49712;3.49122;3.48532;

```

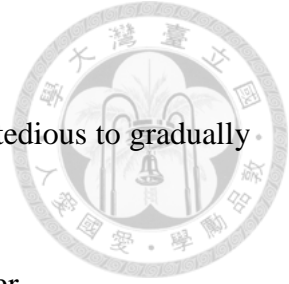
Fig. 3.3.1 Example of the recording files

3.3.2 Teleoperation Program

To collect human-controlled navigating records, we give users some basic instructions and record the (*velocity, measurement*) pairs during their navigation. To fulfill such task, a teleoperation program is required to let the user control and navigate the robot. Since we develop our navigation system on ROS, a straightforward method is to use the ROS node *teleop_twist_keyboard* [14]. However, it is hard to perform natural navigation using such program due to its maneuverability. To let the user navigate the robot smoothly, we design our unique teleoperation program.

We first describe the disadvantages of using *teleop_twist_keyboard* as the teleoperation program. Fig. 3.3.2 shows the control method of *teleop_twist_keyboard*. Eight keys are used to control the direction of moving, while six keys are used for adjusting the speed. Some problems of using this program are as follows:

- If you want the robot to move continuously, you need to hold the keys. Pressing the key once will make the robot move for a small distance and stop.
- It is hard to adjust the rotating velocity, i.e., the angular speed. That is, when the



robot is moving straight, controlling it to rotate is difficult.

- Extra keys are needed to adjust the speed of moving. It will be tedious to gradually slow down or speed up the robot.
- The control is not intuitive. Using arrow keys will be much better.

```
Reading from the keyboard and Publishing to Twist!
-----
Moving around:
  u   i   o
  j   k   l
  m   ,   .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
anything else : stop

CTRL-C to quit
```

Fig. 3.3.2 Control method of *teleop_twist_keyboard*

To solve the above questions, we develop our teleoperation program. Its control method is shown in Fig. 3.3.3.

It should be noticed that the robot rotates only when the user is holding the Left or Right key. For example, although the rotating velocity gradually increases when the user holds the Left key, it will be set to 0 when user releases the button. Therefore, the robot will not keep rotating once you press the arrow key.

The advantages of our program are as follows:

- Only five keys are used. Arrow keys are used to control the direction and speed of the robot, and user can directly adjust the moving and rotating velocity. Therefore, the control is more intuitive.
- When the user wants to accelerate the robot, he can hold the Up key to gradually speed up. User can also slightly adjust robot's moving speed.
- Pressing Left or Right key can slightly adjust the robot's direction, while holding it

can make the robot rotate and turn left or right. While rotating the robot, releasing the Left or Right key will stop its rotation.

- You do not need to hold the keys to make the robot move continuously. After pressing the Up key, the robot will continue moving until moving velocity is decreased.

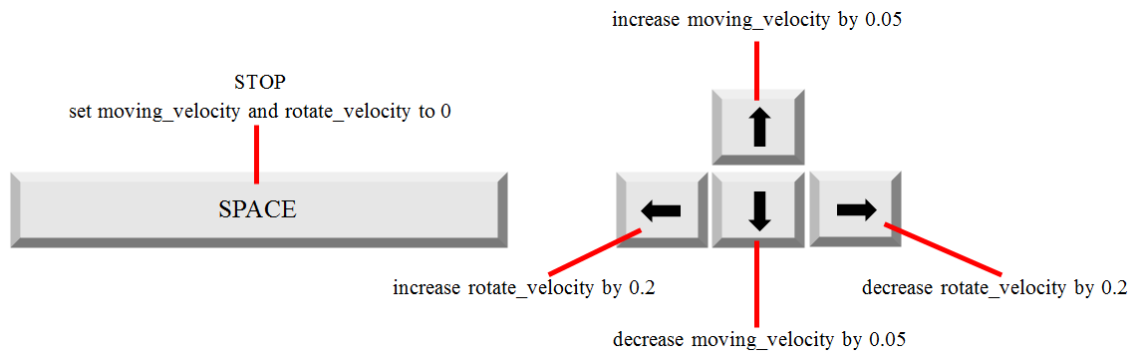


Fig. 3.3.3 Control keys of our teleoperation program

By using such teleoperation program, it will be much easier to control the robot and collect the navigation records for training database. Natural navigation can also be achieved.

3.4 Features and Advantages

As mentioned before, our training data set consists of the human-controlled navigating records and ten basic instructions sets. All instructions in each basic instruction set have similar meanings, and can be represented by the basic instruction of that set. Each recording sequence contains a sequence of $(velocity, measurement)$ pairs, and is collected according to some basic instruction.

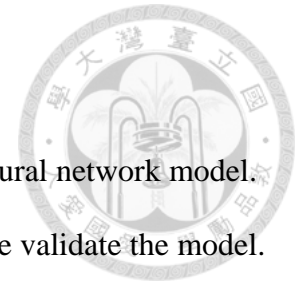
During training process, the neural network model uses measurement as part of the input—the other part comes from basic instructions sets—and the velocity as referenced labels. This model tries to make its outputs correspond to the reference velocities given by human. Thus, by using this model, robots would adjust its velocity according to the

readings of its sensor at that moment and the given initial instruction. It can be expected that the robot will move smoothly as if there is a person controlling it.

While training neural network model, for instance if we want to train the ‘turn right’ instruction, our algorithm will randomly pick one instruction from the ‘turn right’ basic instruction set, convert it into sentence vector, and use the vector as part of the input. For one training data sequence, this sentence vector will be fixed; however, for one recording sequence, we may pick different instructions from the same set, and thus would obtain different training input. By doing so, we make great use of the navigating records. For example, supposing we records 1,000 sequences for the ‘turn left’ instructions, and we construct its instruction set to have 100 instructions. By combining these two we obtain 100,000 different training sequences, and thus we could gather huge amount of training data using fewer recording sequences.

Another benefit of using such data structure for our training database is that we can easily add more instructions into the instruction sets to extend them, without having to collect the training data sequences again. Also, when recording new training sequences, there is no need to give a precise instruction. We will only need to specify the class of basic instructions being executed. Therefore, it would be much easier to expand the training data set. This method also gives good extensibility to the model. In this paper we classify simple instructions into ten basic instruction classes. If we want to add more classes of basic instructions into the original model, we only need to construct a new set of instructions, and records new sequences to train the model.

Chapter 4 Training and Experiments



In this chapter, we first describe how we implement and train our neural network model. We will also discuss how we choose which model to use and how we validate the model. Next, we describe the experiments conducted in simulation and real environment. We analyze the effectiveness of our navigation system by observing the recorded navigation paths and comparing the results with those of humans.

4.1 Training Models

4.1.1 Implementation

We use Keras to implement our neural network [18]. The network structure is as described in Chapter 2. Since our application can be considered a regression problem, we use mean squared error (MSE) function as the loss function. Our training program consists of the following steps:

1. Load in the GloVe model for converting words into vectors.
2. Load in the training data set. For each recording sequence, a corresponding simple instruction sentence is assigned.
3. Build the neural network model, or load in an old, pre-trained model for further training.
4. Start training. The batch size is set to be the maximum value that the computer can afford in memory
5. At the end of every epoch, the model is validated. Models with the lowest loss value or the best validation result will be saved. We will describe this part in the next section.
6. The training process stops only when user interrupts it. If the user interrupts the program, current model is saved.

Model with the lowest loss value is usually the best one. However, to prevent overfitting, we need to validate the model using testing data.



4.1.2 Validation

We evaluate the effectiveness of our neural network during the training process from two aspects. One is the value returned by loss function. Since we use moving and rotating speeds as training labels, the overall problem should be considered as a regression problem instead of classification. Thus, the loss value could present the effectiveness of our models to a certain extent. However, to prevent overfitting, we develop a validation function to evaluate our models over testing data.

The validation function aims to classify the velocity output of each time step into 'correct' or 'wrong' output. In the function, error between each label and the network output is calculated. If the value is below a certain threshold, we consider the output a correct one. Besides, if all outputs of an input sequence are correct, we assume that the destination could be reached. During the training process, we count the number of correct outputs in each sequence, and the number of destinations that could be reached. These two values are used to evaluate the effectiveness of models.

However, it should be noticed that the outputs of our navigation system will affect the input of it. That is, moving at different speeds may result in getting different sensor data. If the output varies a lot from the label, input sequence after this output should be considered invalid. Accumulation of small errors will also produce the same effect. Thus, the validation function should stop counting the number of correct outputs after the accumulation of errors exceeds a certain threshold. Whether the destination would be reached should not be considered either.

To sum up, we define the error of moving and rotating speed at time step i as follows:

$$d_i = v_{i,ref} - v_{i,out} \quad (4.2-1)$$

$$e_i = \omega_{i,ref} - \omega_{i,out} \quad (4.2-2)$$

Here d_i and e_i refer to the error of moving and rotating speed respectively. In addition, v and ω refer to moving velocity and angular velocity, while *ref* and *out* denotes the velocity of labels and network outputs respectively.

Next, we define the accumulation of errors after i time steps as follows:

$$D_i = \sum_{k=1}^i \frac{d_k}{f} \quad (4.2-3)$$

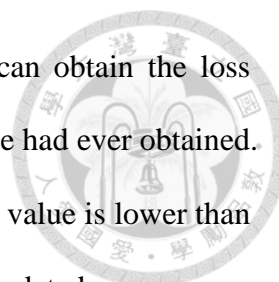
$$E_i = \sum_{k=1}^i \frac{e_k}{f} \quad (4.2-4)$$

Here f denotes the sampling frequency of the laser range finder, while D_i and E_i represents the errors in distance and angles after i time steps respectively. At last, we define four threshold values. They give restrictions to the maximum tolerable errors at every time step, and the tolerable accumulated errors. For each testing sequence, the validation function calculates d_i , e_i , D_i and E_i over each time step. If both d_i and e_i are below their threshold values, the output is considered correct. Once D_i or E_i exceeds the threshold value, the function stops testing on that sequence of data and continue to the next one. Finally, we use the number of correct outputs and reached destinations to evaluate our network model.

Each validated model has two validation values [*correct*, *destination*]. We assume that models with higher *destination* values are better than those with lower values. If two models have the same *destination* values, one with higher *correct* value is considered better.

4.1.3 Monitors

To decide which model to use at last, we design two monitors to keep track on the loss value and validation values.



The first monitor is called loss monitor. While training, we can obtain the loss value of current model. Loss monitor records the lowest loss value we had ever obtained. It helps decide whether to save the current model. If the model's loss value is lower than that recorded in loss monitor, the model is saved and the monitor is updated.

The second monitor is called the validation monitor. Similarly, it records the best validation values, and helps save the model with the best validation result.

Fig. 4.1.1 shows how these monitors work during training process. It can be seen that both monitors are updated once, and thus two new models are saved.

```
Epoch 189/10000
344/344 [=====] - 3s - loss: 0.0016
loss: 0.00163046495047 , valid: [514, 5]
loss monitor: 0.00160666390696 , valid monitor: [617, 11]
Epoch 190/10000
344/344 [=====] - 4s - loss: 0.0016
loss: 0.00157700751253 , valid: [495, 7]
loss monitor: 0.00157700751253 , valid monitor: [617, 11]
Epoch 191/10000
344/344 [=====] - 5s - loss: 0.0015
loss: 0.0015369077094 , valid: [629, 12]
loss monitor: 0.0015369077094 , valid monitor: [629, 12]
Epoch 192/10000
344/344 [=====] - 4s - loss: 0.0015
loss: 0.00153449492605 , valid: [631, 10]
loss monitor: 0.00153449492605 , valid monitor: [629, 12]
Epoch 193/10000
344/344 [=====] - 3s - loss: 0.0016
loss: 0.001557773591 , valid: [535, 7]
loss monitor: 0.00153449492605 , valid monitor: [629, 12]
Epoch 194/10000
344/344 [=====] - 3s - loss: 0.0016
loss: 0.00162494035714 , valid: [601, 8]
```

Fig. 4.1.1 Loss monitor and validation monitor

To sum up, we keep 3 models during our training process. One is the current model, which will be saved when the program stops. Another one is the model with the lowest loss value. Its loss value is recorded by the loss monitor. The last one is the model with the best validation result. Its validation values [*correct, destination*] are recorded by the validation monitor.

We found that models with lower loss values often have better validation results. That is, two monitors show consistency in our research.



4.2 Experiments and Results

4.2.1 Simulation

We simulate our system under the **Gazebo** simulating environments [19]. We choose twenty different starting positions among all the maps, and at each position we let the robot execute 100 legal simple instructions. 80 of them are instructions recorded in the basic instruction sets, while the rest are new instructions with similar meanings to some basic instructions. If the robot stops its navigation at a proper position without bumping into any obstacle, we consider it a successful navigation. The results show that for each basic instruction, the robot can navigate to the desired positions.

To analyze the results and compare our navigation paths with human controlled navigation path, we use **rviz** to record the paths and structures of environments observed by laser range finder during navigation [20]. Fig. 4.2.3 to Fig. 4.2.6 shows some examples of the success navigation paths and the indoor environment structures. In these figures, the red paths represent results from our system, while green paths indicate the paths produced by humans. The circles indicate the starting positions, while triangles specify stopping positions. However, for the results of Class 4 instructions ‘Turn around’, we use red arrows to indicate the starting positions and directions of the robot, and green arrows to represent the stopping ones.

We will discuss the experiment results and comparisons in the Comparison section.

4.2.2 Real Environment

We test our navigation system on the first and third floors of National Taiwan University Building for Research Excellence. We choose several different starting positions, giving different simple instructions to the robot and observe its behaviors. It turns out that the robot could end up at the proper positions as well. A demonstration

video is provided to show our results.

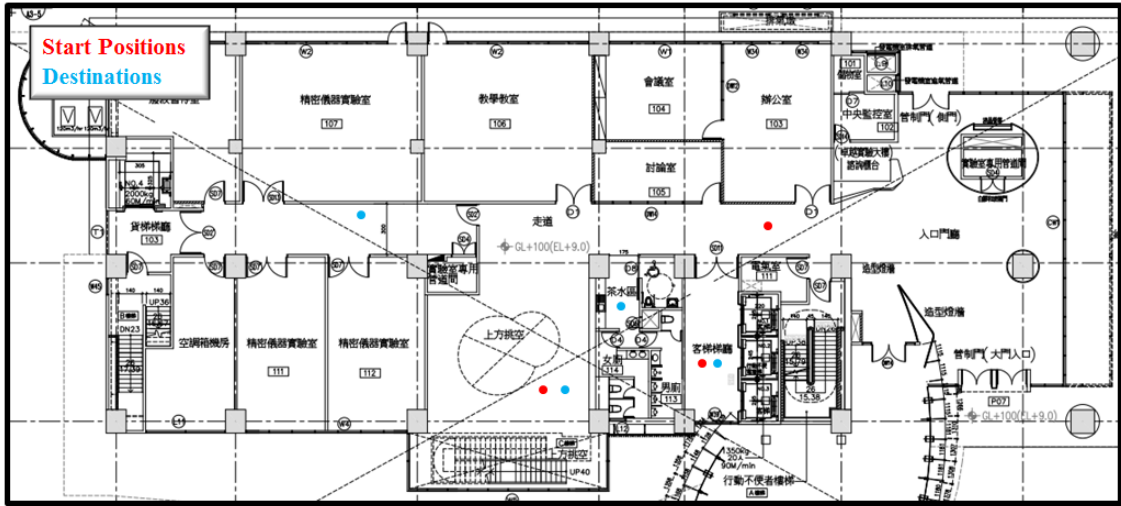
Fig. 4.2.1 shows the environment in which we test our system. The red dots indicate the starting positions, while the blue ones specify the destinations. Some instructions are designed to make the robot navigate from these starting positions to these particular destinations. We aim to have the robot move to these locations using such instructions, instead of giving it a legal simple instruction and see whether it can stop at a proper position. The results show that as long as the instruction is correct, our robot is able to navigate to the desired location.

Fig. 4.2.2 shows a screenshot of our demonstration video. The sub-screen shows the measurements obtained by the laser range finder, which are displayed in **rviz**. We can observe the structure of environment from the sub-screen. Therefore, we can judge whether our robot is taking the right action at that moment.

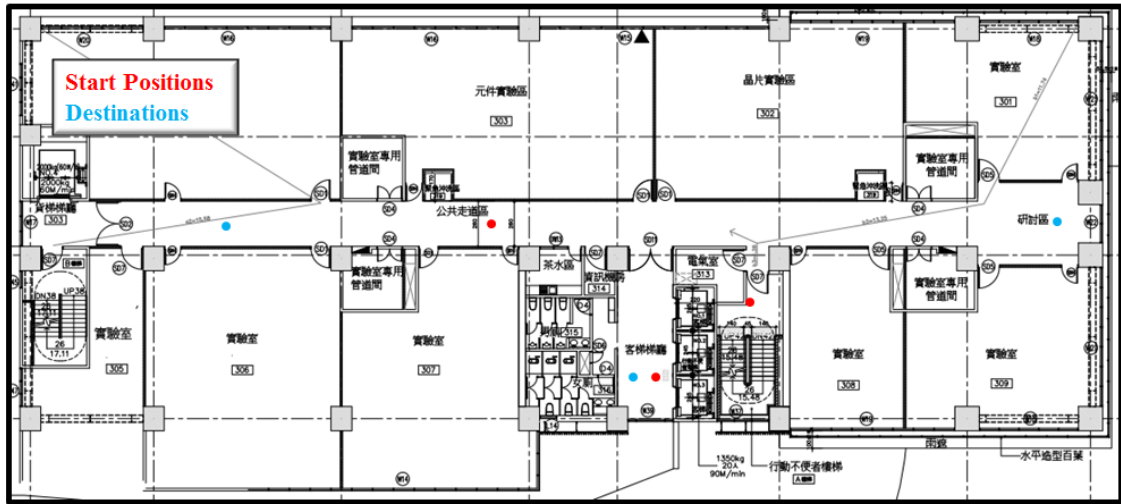
4.2.3 Interpolation

We use Hokuyo UTM-30LX on our warehouse robot ‘Penguin’. To verify the effectiveness of our interpolation stage, we change the laser range finder to URG-04LX in simulation. Their detection ranges, angular resolutions and maximum measuring distance are all different. However, the results show that robot using URG-04LX as its sensor has similar performance to that using the original sensor.

In each experiment, the starting position and the given instruction remain the same. We only change the type of laser range finder to observe the changes in behavior during navigation. In short, we find little difference between navigation paths generated by UTM-30LX and those by URG-04LX. It is proper to say that our interpolation function works well.



(a) Floor plan for 1st floor



(b) Floor plan for 3rd floor

Fig. 4.2.1 Floor plans of Building for Research Excellence



Fig. 4.2.2 Screenshot of the demonstration video

Instruction Number	Testing Times	Success Times	Success Rate
1	263	242	0.9202
2	257	185	0.7198
3	264	177	0.6705
4	241	209	0.8672
5	178	74	0.4157
6	185	83	0.4486
7	76	17	0.2237
8	62	14	0.2258
9	238	133	0.5588
10	236	140	0.5932
Total	2000	1274	0.6370

Table 4.2-1 Success rate of each basic instruction

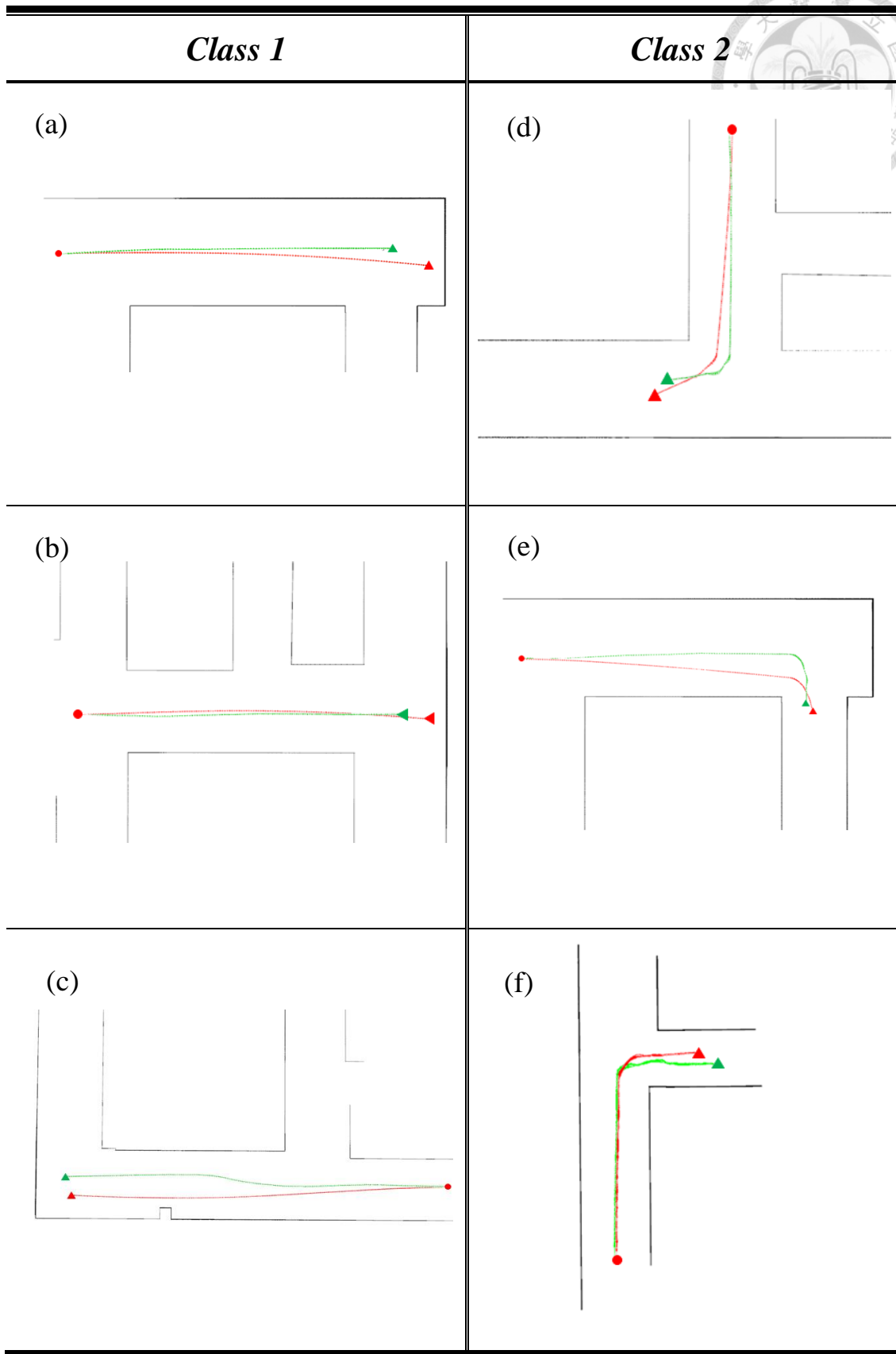


Fig. 4.2.3 Experiment results of Class 1 and Class 2 instructions

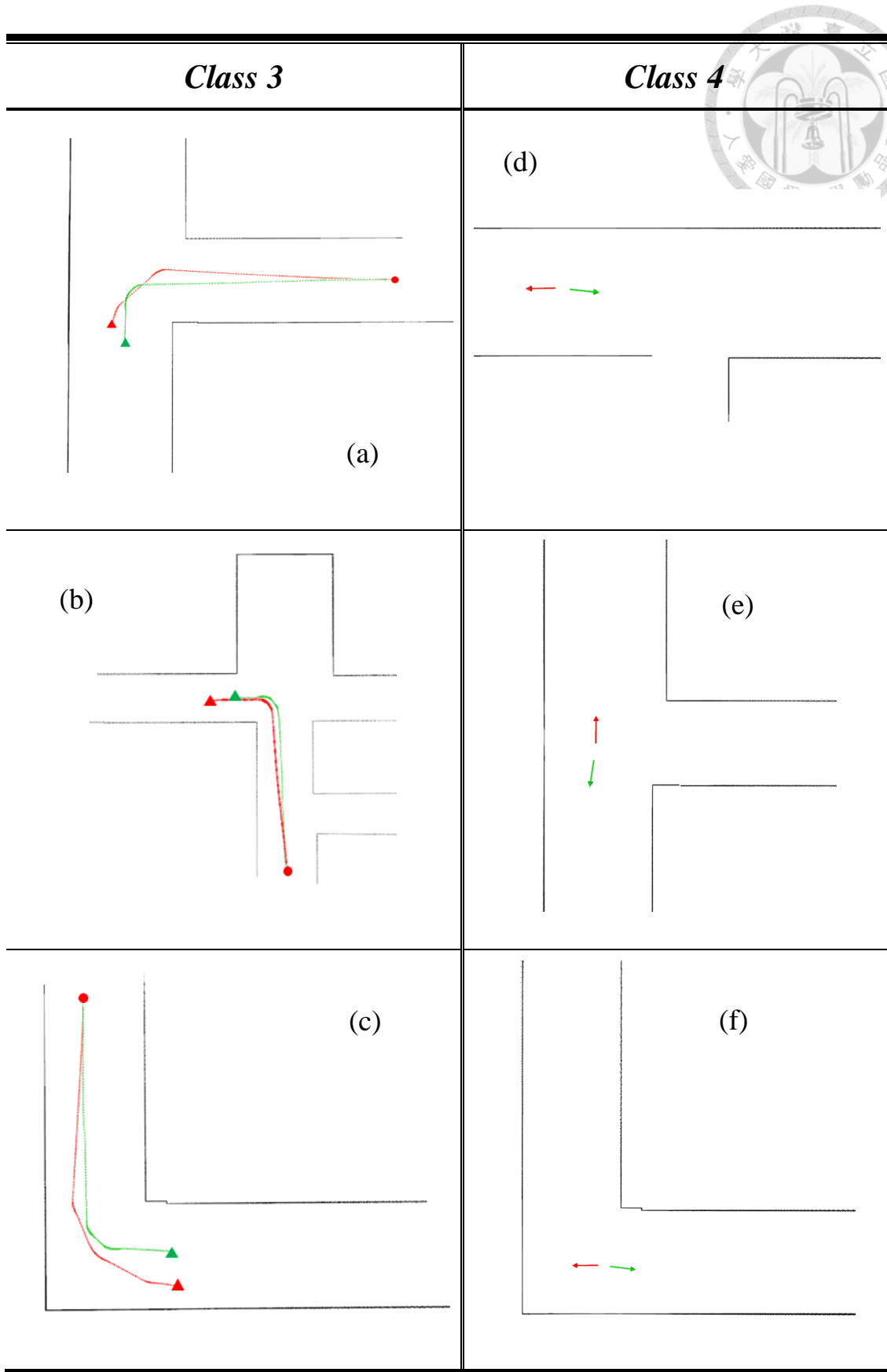


Fig. 4.2.4 Experiment results of Class 3 and Class 4 instructions

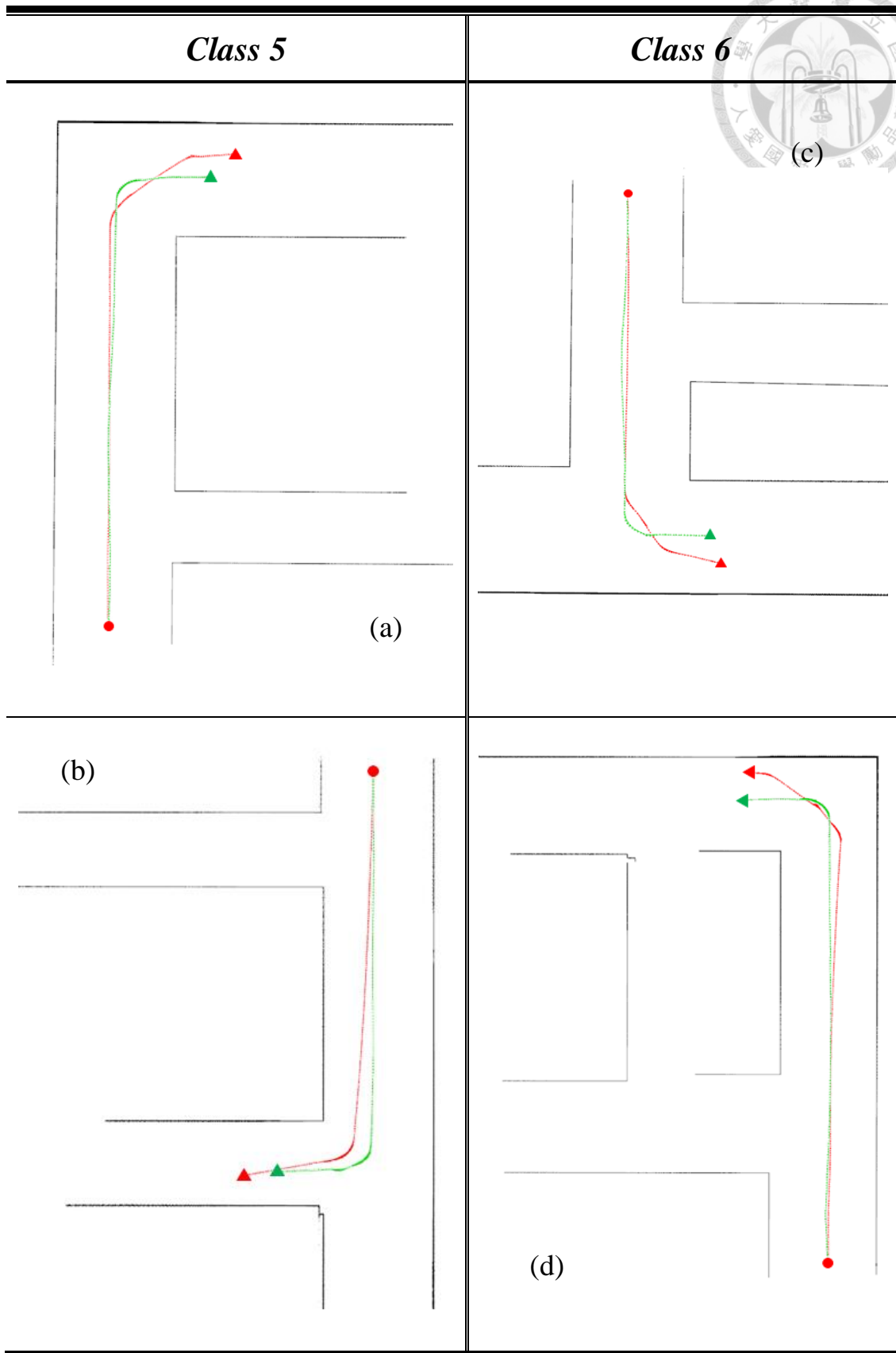


Fig. 4.2.5 Experiment results of Class 5 and Class 6 instructions

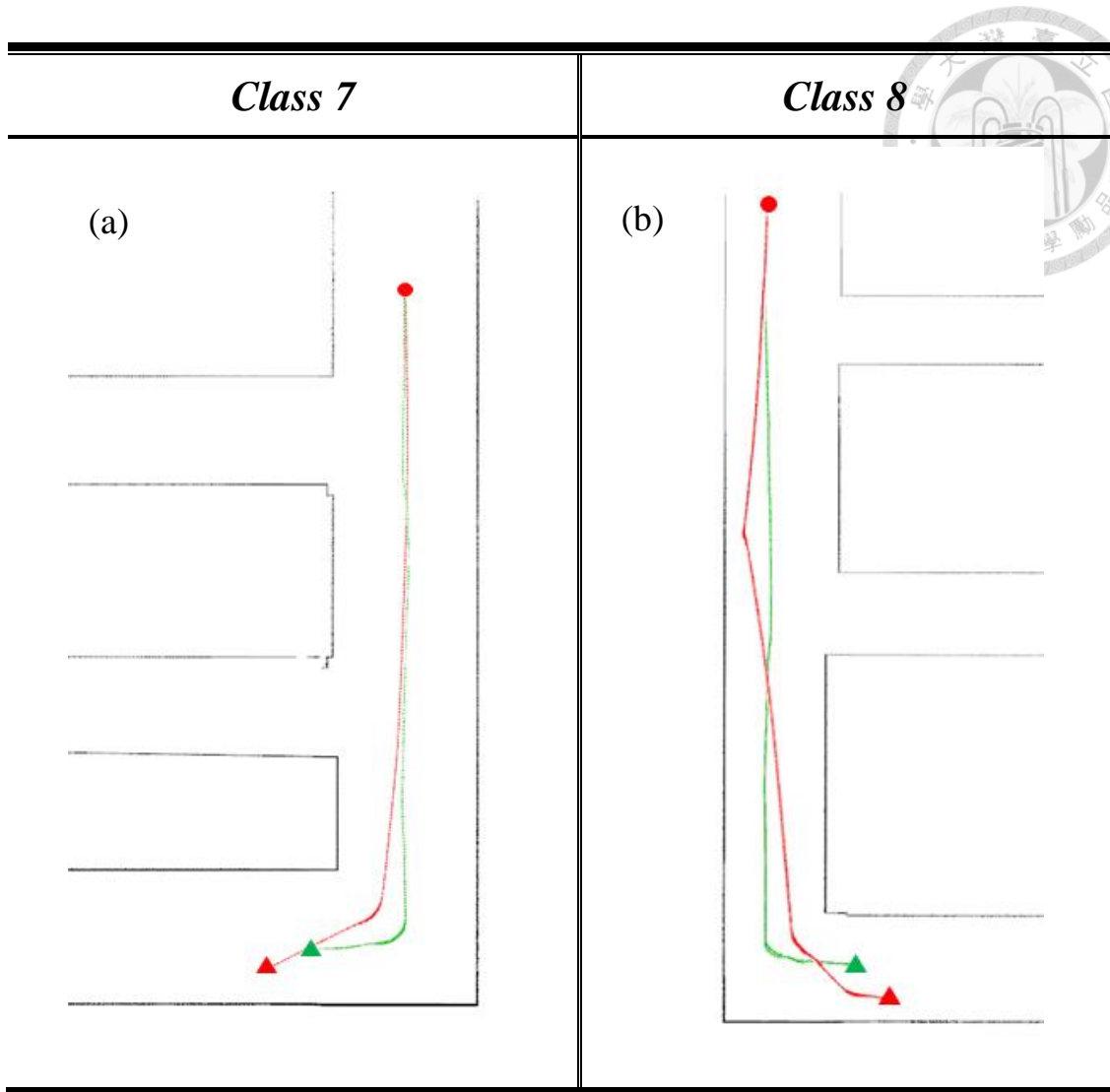
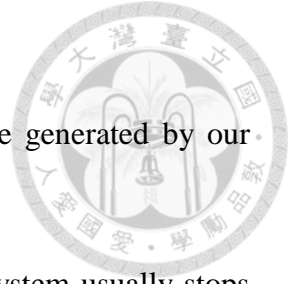


Fig. 4.2.6 Experiment results of Class 7 and Class 8 instructions

4.2.4 Comparisons

We can compare the results of our system with those produced by humans in the above figures. Table 4.2-1 shows the success rate of each basic instruction. Some features and problems can be observed:

- In most cases, our system can navigate the robot smoothly. The generated paths are straightforward and intuitive, and are very close to the shortest paths to complete these instructions. Examples can be seen in Fig. 4.2.4 (b), Fig. 4.2.5 (b) and Fig. 4.2.6 (a). In such case, there is little difference between the path of our system and



that of humans.

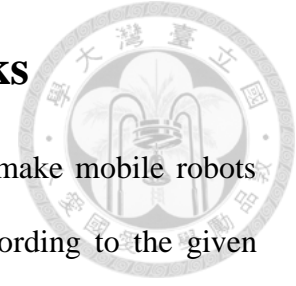
- The paths generated by humans have sharper turns than those generated by our navigation system.
- While executing the ‘Go straight to the end’ instruction, our system usually stops the robot at a closer distance than human does. This can be seen in Fig. 4.2.3 (a) and Fig. 4.2.3 (b), where the red triangles are closer to the walls than the green ones.
- Compared with the robot controlled by humans, robot controlled by our system often stops at a position closer to the side wall after performing a right or left turn. This can be observed in Fig. 4.2.4 (c) and Fig. 4.2.5 (c).
- While turning right or left, the robot often performs a two-stage turn in our system. Fig. 4.2.4 (c) and Fig. 4.2.5 (c) shows some examples. This results in a longer navigation path.
- It is hard for our system to control the robot to stay in the middle of the road. The robot sometimes gradually moves close to the side walls, and ends up bumping into them. This phenomenon can be observed in Fig. 4.2.4 (c) and Fig. 4.2.6 (b). Therefore, when the distance of path become longer, the success rate drops.
- Although it cannot be seen from the figure, the cost of time for these navigations are all very low; sometimes the robot even moves with a faster speed than human does, and the destination can still be reached successfully.
- The system spends most of its time turning right and left. When performing a two-stage turn, the robot often temporarily stops in the middle of the road. However, the robot can still get to the correct position eventually.
- The mobile robot may execute wrong instructions during navigation. For example, while executing the ‘go straight to the end’ instruction, seldom it will choose to

turn right when it observes that there is a way on its right hand side. We observe that adding more training data to the training set would improve this situation to some extent.

- From Table 4.2-1, we can observe that the success rate drops when the distance of path become longer. We assume that when robots need to travel long distances, the chance of making wrong decisions increases, and the accumulation of small errors may result in failure. This is because the output of our program will affect its input, and thus any small error in the output velocity has a great effect on the final result of navigation.

Some of the above problems will affect the execution of the next instruction, such as stopping at a position close to the wall, or executing wrong instructions. Therefore, to make our robot execute more complex navigation instructions, we aim to find solutions for these problems in the future.

Chapter 5 Conclusions and Future Works

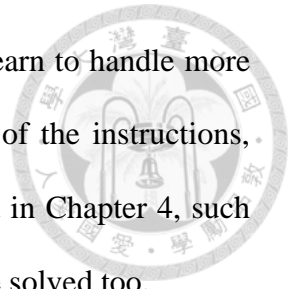


In this paper we present a neural network based model that make mobile robots capable of navigating through unknown indoor environments according to the given verbal instructions. The navigating system splits instructions expressed in natural language into several simple instructions, and compute the sentence vector for each of them. The sentence vector along with the readings of laser range finder mounted on the mobile robots is given to the neural network as input data. The model will calculate the moving and rotating speed of robot, leading it to new positions. New data acquired from sensor will then be used to calculate the next movement of the robot. The process will continue until the robot stops at one position for a time period—that is, the robot arrives at the destination. And then, the next sentence vector will be executed.

The difference between our methods and others is that we do not decide local or global goal positions for robots to navigate to, since the application is under unknown environment. Instead we decide the velocity at the next time step. Moreover, we do not aim to construct probabilistic models for keywords used in instructions. We use the concept of word vector and machine learning to handle the complexity and variety of instructions. We provide a unique solution to the problem of auto navigation under unknown indoor environments, and experiments show that our system could make mobile robots navigate to the correct positions under both simulation and real-world environments.

For future work, we first plan to add more sensors to our system. Using visual sensors will allow robots to handle higher level instructions. In addition, extra sensors could also be used to detect dynamic objects. We plan to add different types of training data into the training set gradually, including the records of avoiding obstacles and

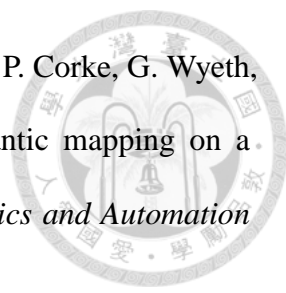
searching for the right directions. We expect that the model could learn to handle more issues during navigation. Besides, we plan to increase complexity of the instructions, adding and modifying the basic instruction sets. Problems proposed in Chapter 4, such as decreasing of success rate under long traveling distances, are to be solved too.



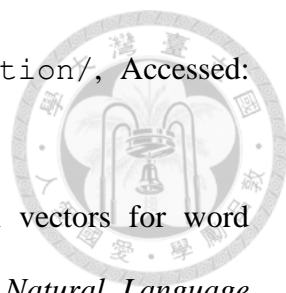
REFERENCE



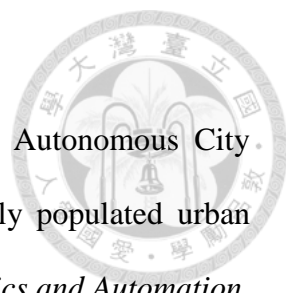
- [1] D. W. Ko, C. Yi and I. H. Suh, "Semantic mapping and navigation: A Bayesian approach," *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, 2013, pp. 2630-2636.
- [2] B. Talbot, O. Lam, R. Schulz, F. Dayoub, B. Upcroft and G. Wyeth, "Find my office: Navigating real space from semantic descriptions," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, 2016, pp. 5782-5787.
- [3] X. Zhang, B. Li, S. L. Joseph, J. Xiao, Y. Sun, Y. Tian, J. P. Muñoz and C. Yi, "A SLAM Based Semantic Indoor Navigation System for Visually Impaired Users," *2015 IEEE International Conference on Systems, Man, and Cybernetics*, Kowloon, 2015, pp. 1458-1463.
- [4] O. M. Mozos, C. Stachniss and W. Burgard, "Supervised Learning of Places from Range Data using AdaBoost," *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 1730-1735.
- [5] B. Kaleci, Ç. M. Şenler, H. Dutağacı and O. Parlaktuna, "A probabilistic approach for semantic classification using laser range data in indoor environments," *2015 International Conference on Advanced Robotics (ICAR)*, Istanbul, 2015, pp. 375-381.
- [6] L. Shi, R. Khushaba, S. Kodagoda and G. Dissanayake, "Application of CRF and SVM based semi-supervised learning for semantic labeling of environments," *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*, Guangzhou, 2012, pp. 835-840.

- 
- [7] N. Sünderhauf, F. Dayoub, S. McMahon, B. Talbot, R. Schulz, P. Corke, G. Wyeth, B. Upcroft and M. Milford, "Place categorization and semantic mapping on a mobile robot," *2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm*, 2016, pp. 5729-5736.
- [8] R. Goeddel and E. Olson, "Learning semantic place labels from occupancy grids using CNNs," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon*, 2016, pp. 3999-4004.
- [9] H. J. Chang, C. S. G. Lee, Y. H. Lu and Y. C. Hu, "P-SLAM: Simultaneous Localization and Mapping With Environmental-Structure Prediction," in *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 281-293, April 2007.
- [10] D. P. Ström, F. Nenci and C. Stachniss, "Predictive exploration considering previously mapped environments," *2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA*, 2015, pp. 2761-2766.
- [11] "ROS/Introduction - ROS Wiki", <http://wiki.ros.org/ROS/Introduction>, Accessed: 2017-07-12.
- [12] "ros_control - ROS Wiki", http://wiki.ros.org/ros_control, Accessed: 2017-07-10.
- [13] "hokuyo_node - ROS Wiki", http://wiki.ros.org/hokuyo_node, Accessed: 2017-07-10.
- [14] "teleop_twist_keyboard - ROS Wiki", http://wiki.ros.org/teleop_twist_keyboard, Accessed: 2017-07-10.
- [15] "sklearn.preprocessing.normalize — scikit-learn 0.18.2 documentation", <http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html>, Accessed: 2017-07-12.
- [16] "SpeechRecognition 3.7.1: Python Package Index - PyPI",

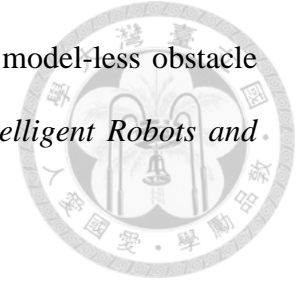
<https://pypi.python.org/pypi/SpeechRecognition/>, Accessed: 2017-07-13.

- 
- [17] J. Pennington, R. Socher, C. D. Manning, "Glove: Global vectors for word representation", *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014)*, vol. 12, pp. 1532-1543, 2014.
- [18] "Keras Documentation", <https://keras.io/>, Accessed: 2017-07-13.
- [19] "Gazebo", <http://gazebo.org/>, Accessed: 2017-07-14.
- [20] "rviz - ROS Wiki", <http://wiki.ros.org/rviz>, Accessed: 2017-07-14.
- [21] Charly Huang, "採樣式路徑規劃與立體即時建圖及定位於具社交感知之服務型機器人應用", 臺灣大學電機工程學研究所學位論文, pp. 1 - 159, 2016
- [22] H. M. Gross, H. J. Boehme, C. Schroeter, S. Mueller, A. Koenig, Ch. Martin, M. Merten and A. Bley, "ShopBot: Progress in developing an interactive mobile shopping assistant for everyday use," *2008 IEEE International Conference on Systems, Man and Cybernetics, Singapore*, 2008, pp. 3471-3478.
- [23] V. Kulyukin, C. Gharpure and J. Nicholson, "RoboCart: toward robot-assisted navigation of grocery stores by the visually impaired," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 2845-2850.
- [24] W. Hess, D. Kohler, H. Rapp and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, 2016, pp. 1271-1278.
- [25] K. Sasaki, H. Tjandra, K. Noda, K. Takahashi and T. Ogata, "Neural network based model for visual-motor integration learning of robot's drawing behavior: Association of a drawing motion from a drawn image," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg,

2015, pp. 2736-2741.

- 
- [26] G. Lidoris, F. Rohrmuller, D. Wollherr and M. Buss, "The Autonomous City Explorer (ACE) project — mobile robot navigation in highly populated urban environments," *2009 IEEE International Conference on Robotics and Automation*, Kobe, 2009, pp. 1416-1422.
- [27] Z. Zhao and X. Chen, "Semantic mapping for object category and structural class," *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, 2014, pp. 724-729.
- [28] S. Hemachandra, T. Kollar, N. Roy and S. Teller, "Following and interpreting narrated guided tours," *2011 IEEE International Conference on Robotics and Automation*, Shanghai, 2011, pp. 2574-2579.
- [29] Jingchen Tong, Dong Chen, Yan Zhuang and Wei Wang, "Mobile robot indoor semantic mapping using 3D laser scanning and monocular vision," *2010 8th World Congress on Intelligent Control and Automation*, Jinan, 2010, pp. 1212-1217.
- [30] W. Mei, W. Pan and L. Xie, "Semantic-understand-based landmark navigation method of robots," *2012 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, Zhangjiajie, 2012, pp. 760-764.
- [31] E. A. Antonelo and B. Schrauwen, "On Learning Navigation Behaviors for Small Mobile Robots With Reservoir Computing Architectures," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 4, pp. 763-780, April 2015.
- [32] J. A. Caley, N. R. J. Lawrance and G. A. Hollinger, "Deep learning of structured environments for robot search," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, 2016, pp. 3987-3992.

- [33] L. Tai, S. Li and M. Liu, "A deep-network solution towards model-less obstacle avoidance," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, 2016, pp. 2759-2764.



VITA



姓名：陳長鈞

性別：男

生日：民國 81 年 1 月 26 日 (1992/01/26)

籍貫：台北市

學歷：

1. 民國 106 年 國立台灣大學電機工程學研究所畢業
2. 民國 103 年 國立台灣大學電機工程學系畢業
3. 民國 99 年 台北市立建國高級中學畢業

發表著作：

Ren C. Luo and Chang-Jiun Chen, "Recursive Neural Network Based Semantic Navigation of an Autonomous Mobile Robot through Understanding Human Verbal Instructions", accepted by 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017), Vancouver, Canada, September 24-28, 2017 (EI).

榮譽事蹟：

民國 105 年 參加「長庚醫療財團法人 2016 醫療機器人比賽」榮獲 **冠軍**

民國 105 年 參加「2016 年全國機器人創意競賽 工業機器人智慧應用創意組」榮獲 **冠軍**