

Autonomous Driving Segway Robots

by

Jiaming Liu

**A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering
(Electrical Engineering)
in the University of Michigan-Dearborn
2021**

Master's Thesis Committee:

**Professor Weidong Xiang, Chair
Associate Professor Sridhar Lakshmanan
Assistant Professor Alireza Mohammadi**

©Jiaming Liu

2021

ACKNOWLEDGEMENTS

I would like to extend my sincerest gratitude to my advisor Professor Weidong Xiang and my co-advisor Sateesh K. Addepalli, for all the guidance and supervision, for all the backups, and also for giving me the opportunity to present our works throughout this thesis. Also, I extend my warmest thanks to my parents Kezhuang Liu and Zhengchun Liu hereby for being my strongest support while reaching my degree. Special thanks to Shiyuan Wang, Xiyuan Wang and all the partners in our lab for those inspiration, collaboration, and technical supports. Thanks to my roommates Zhengru Li, Feng Han, Yuxiao Zhang, and Yuchi Guo for all the days and nights together. I would also like to say thanks to my car, for leaving me a dreamy memory, accompanying me so that I am not alone.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	ii
LIST OF TABLES	v
LIST OF FIGURES	vi
ABSTRACT.....	x
CHAPTER	
1 INTRODUCTION	1
1.1 Overview	1
1.2 Main Objectives	7
1.3 Thesis Outline	8
2 LITERATURE REVIEW	9
2.1 Autonomous Robot	9
2.2 Obstacle Avoidance.....	14
2.3 LiDAR and Point Cloud.....	16
2.4 Sensor Data Fusion.....	17
2.5 SLAM and Filtering	19
2.6 Path Planning and Reinforcement Learning.....	23
3 APPROACH	31
3.1 Hardware Schematic of the Robot	31
3.2 Kinematic and Dynamic Model of the Robot	33
3.2.1 Two-Wheel Differential Kinematic Model	33
3.2.2 Dynamic Model	35
3.3 Controlling and Sensing.....	36
3.3.1 Controlling the Robot.....	36
3.3.2 Sensing.....	38
3.4 Software System Framework	44
3.5 Function Realization	45
3.5.1 Obstacle Avoidance (2D-LiDAR Occupied Grid Mapping).....	45

3.5.2 Sensor Data Fusion.....	50
3.5.3 2D SLAM.....	54
3.5.4. Path Planning.....	61
3.5.5 Simulation.....	63
4 TESTING AND RESULTS.....	68
4.1 Test Procedure and Environment.....	68
4.2 Test Results.....	71
4.2.1 Obstacle Avoidance.....	71
4.2.2 Data Fusion.....	73
4.2.3 2D SLAM.....	77
4.2.4 Path Planning.....	80
4.2.5 Simulation Results.....	82
5 CONCLUSION AND SCOPE.....	86
5.1 Limitations.....	87
5.2 Real World vs. Simulation.....	87
5.3 Scope.....	88
REFERENCES.....	89

LIST OF TABLES

Table 2.1: Overview of common sensors for autonomous robotics. A: active; P: passive; PC: proprioceptive; EC: exteroceptive. [35].....	11
Table 2.2: Advantages and limitation of various sensors within USVS. [24].....	14
Table 2.3: Comparison between lase & vision SLAM in some aspects.	21
Table 2.4: Configuration of lase SLAM through development.	22
Table 3.1: Hardware configuration of the proposed robot.....	31
Table 3.2: Specification of DS3225 25KG digital servo.	37
Table 3.3: Specification of LS 16 Channel LiDAR.....	38
Table 3.4: Vertical angles corresponding to laser ID.	39
Table 3.5: Specification of HIKVISION DS-2CD2455FWD-IW camera.	40
Table 3.6: Fuzzy logic rule sets.	49
Table 3.7: Meanings corresponding to different colors on map for path planning.....	61
Table 4.1: Specification of different test environments.....	70
Table 4.2: Specification of 2D local grid map.....	72
Table 4.3: Measurements of objects in map and reality.	77
Table 4.4: Path scoring time for different algorithms.....	81

LIST OF FIGURES

Figure 1.1: Three generations of Stanford Cart. (a): First generation in 1961. (b): Third generation in 1964-71. (c): Fourth generation in 1979. Image credit: Stanford Cart via Stanford Robotics' Legacy, Stanford News.....	2
Figure 1.2: NavLab-5 (left side) and the artificial neural network for learning human driving skills (right side). Image credit: NavLab 5 via Carnegie Mellon University.....	3
Figure 1.3: Poster for First DRAPA Grand Challenge (top side), and some of the hardware structure of Stanley, the entrant vehicle developed based on a Volkswagen Touareg. Image credit: The DARPA Grand Challenge: Ten Years Later via DRAPA.....	4
Figure 1.4: Overview of the robot proposed in this thesis, taken in University of Michigan – Dearborn, Sep 17th 2020.	6
Figure 2.1: One kind of robotics classification based on control.	10
Figure 2.2: General autonomous robot scheme. [35].....	10
Figure 2.3: General wheel configuration for rolling mobile robots. [35].....	12
Figure 2.4: Examples of TurtleBot Family. Image credit: www.TurtleBot.com.....	13
Figure 2.5: One example of obstacle avoidance logic (left) and guiding 2D grid map (right). [8]	15
Figure 2.6: CURM configuration. [24].....	16
Figure 2.7: FOT configuration of LiDAR. [30].....	17
Figure 2.8: One example flow chart of fusing data from LiDAR and camera. [23].....	17
Figure 2.9: One typical application of data fusion, 3D object co-detection and segmentation The segmented parts also has semantic informations. [33].....	18
Figure 2.10: One example of fuzzy logic based data fusion. [32]	18
Figure 2.11: Hardware setup and corner extraction for ILCC. [31]	19

Figure 2.12: Different type of map. (a): 3D point cloud map. (b): 2D semantic map. (c): 2D grid map. (d): 3D semantic map. [38]	20
Figure 2.13: General classification of robot path planning.....	24
Figure 2.14: Citation of robotic path planning techniques through years. [6].....	25
Figure 2.15: One example of path planning based on 2D grid map. [28].....	26
Figure 2.16: (a): General classification of machine learning. (b): Simple representation of reinforcement learning.....	29
Figure 2.17: Pseudocode of Q-learning.	29
Figure 2.18: (a): One example of typical maze puzzle problem. (b): Local 2D grid map generated from our robot. This can be as similar as the maze puzzle problem. [37]	30
Figure 3.1: Hardware setup for the proposed robot.	32
Figure 3.2: The kinematic model of the robot. (a) shows the global and robot coordinate of the robot, and (b) shows the decomposition of the robot's motion.	33
Figure 3.3: Dynamic model of the robot.....	35
Figure 3.4: The PID controller configuration for the robot.	36
Figure 3.5: Digital servo structure and PWM working principle.	37
Figure 3.6: Wiring diagram for servos and Arduino.....	38
Figure 3.7: Vertical scan angle of LiDAR.	39
Figure 3.8: Visualization of captured point cloud data.....	40
Figure 3.9: Configuration of wheel encoder.	41
Figure 3.10: Roll, pitch, and yaw of IMU on the robot.	43
Figure 3.11: Software system framework.....	44
Figure 3.12: Flow chart for obstacle avoidance.....	46
Figure 3.13: LD, FD, RD on grid map.....	47
Figure 3.14: Fuzzy membership functions of inputs. (a): LD. (b): FD. (c): RD. (d): α . (e): v	48
Figure 3.15: Fuzzy membership functions of output. (a): θ . (b): α	49

Figure 3.16: Flowchart for data fusion process. [31].....	50
Figure 3.17: One example scene of collecting data by the robot.....	51
Figure 3.18: Camera, image, and real-world coordinates.	52
Figure 3.19: POMDP representation of 2D SLAM.	55
Figure 3.20: Flowchart of 2D SLAM process.	56
Figure 3.21: The two components of the motion model. Within the interval $L^{(i)}$ the product of both functions is dominated by the observation likelihood in case an accurate sensor is used. [12] ...	57
Figure 3.22: Flowchart of path planning process.....	61
Figure 3.23: Save zone (light blue) and dangerous zone (pink) on map for path planning.....	63
Figure 3.24: Model of robot in simulation (right side) and visualization of LiDAR in simulation (left side).	64
Figure 3.25: Indoor testing environment on simulation. The blue dot is our robot model, and the main obstacle includes wall, fire hydrant, and fast food restaurant.....	65
Figure 3.26: Outdoor testing environment on simulation. Overview (bottom side) and street scene (top side).	66
Figure 4.1: Pictures of several test environments. Upper left: university building hallway. Middle and downer left: research lab. Right: Room.	69
Figure 4.2: Testing environment for data fusion. Left: indoor scenario. Right: outdoor scenario.	70
Figure 4.3: Plain view of 2D local grid map and 2D point cloud.	72
Figure 4.4: Upper: raw image capture from the camera. Downer: undistorted image after calibration.	74
Figure 4.5: Left: mean error in pixels of 25 images. Right: position of checkerboards in a camera-centric basis.....	75
Figure 4.6: Corners of checkers on point cloud.	75
Figure 4.7: Four corners of larger checkerboard with respect to image and point cloud.	76
Figure 4.8: Colored point cloud of indoor scenario after transformation.	76

Figure 4.9: Color image, point cloud, and fusion results of indoor and outdoor scenario.	77
Figure 4.10: Map constructing process with current LiDAR hit (green color) before removing ground hit.	78
Figure 4.11: Map constructing process without current LiDAR hit after removing ground hit. ...	79
Figure 4.12: Left: result of global 2D grid map in scale of $30 \times 30 m$. Right: corresponding scheme of room.	79
Figure 4.13: 2D global grid map for path planning in scale of $600 \times 600 pixels$ (left) and in detail (right).	80
Figure 4.14: Path generated from different algorithm with the same start and destination.	81
Figure 4.15: Path planning and navigation of simulation robot. The window tagged “Image” presents the visualization of virtual camera set on the robot model.	82
Figure 4.16: 2D grid map generated from Gamapping of simulation.	83
Figure 4.17: 3D lase map constructing process of simulation. The left is the 3D lase map, and the right is the corresponding real scene in simulation environment.	83
Figure 4.18: Result of 3D lase map generated by Loam algorithm.	84
Figure 4.19: Result of 3D lase map generated by Lego-loam algorithm.	85

ABSTRACT

In this thesis, an autonomous driving robot has been proposed and built based on a two-wheel Segway self-balancing scooter. Sensors including LiDAR, camera, encoder, and IMU were implemented together with digital servos as actuators. The robot was tested simultaneously with the functionality features including obstacle avoidance based on fuzzy logic and 2D grid map, data fusion based on co-calibration, 2D simultaneously localization and mapping (SLAM) and path planning under different scenarios both indoor and outdoor. As a result, the robot initially has the ability of self-exploration with avoiding obstacles and constructing 2D grid map simultaneously. A simulation of the robot with same functionalities except data fusion has also been tested and performed based on robot operating system (ROS) and Gazebo as the simple comparison of the robot in real world.

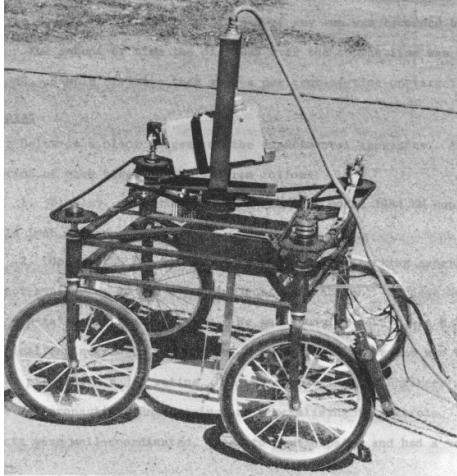
CHAPTER 1

INTRODUCTION

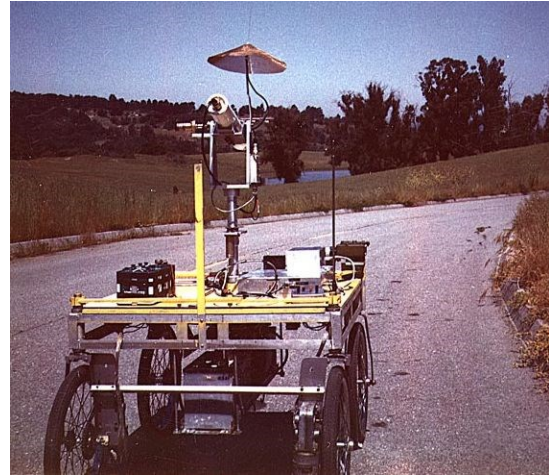
1.1 Overview

Autonomous driving, as known as self-driving, is a very popular research topic in current which integrates several automated systems including sensing and perception, movement-controlling, networking, artificial intelligent, and decision-making to achieve a safe and fully automated system with little or no human input. The 1.1 overview section will briefly introduce the development history of autonomous driving, the connection and relationships between autonomous driving and our thesis's topic, autonomous driving robot.

It has been nearly half century for people on the journey of chasing the autonomous driving dream. The word “driving” in autonomous driving reminds that there should be something to be driven. It could be a car, a drone, or a robot. As the most essential transportation in our society, cars were the first to be associated with autonomous driving. The first “autonomous driving car” generally accepted is the Stanford Cart, as shown in Figure 1.1. It was a long-term project early from 1961 to 1980. The first Stanford Cart was built in 1961 by mechanical engineering graduate student James L. Adams, based on a cart with four bicycle wheels and motors, a single black and white camera, connected through a very long cable to a console and TV display. The original objective of the Stanford Cart is to research the feasibility of remotely controlling a vehicle through vision and radio. The control efficiency of the Cart was very limited at the beginning, but it provided a platform of researching and developing the autonomous driving technologies. In 1971-80, the Stanford Cart was added on a “slider” to obtain multiple visions, a KL10 processor and some early AI systems to use binocular vision to navigate slowly while avoiding obstacles. In 1979, the cart successfully crossed a chair-filled room without human intervention in about five hours. Although the efficiency was still very limited, it laid the basic research direction of



(a)



(b)



(c)

Figure 1.1: Three generations of Stanford Cart. (a): First generation in 1961. (b): Third generation in 1964-71. (c): Fourth generation in 1979. Image credit: Stanford Cart via Stanford Robotics' Legacy, Stanford News.

modern autonomous driving car, controlling, sensing from the environment, and making decisions.

From the 1980s – 2000s, with the breakthroughs in computer, robot control and sensing technologies, the autonomous driving has entered a stage of rapid development. The military, universities and companies have expanded extensive and close cooperation, which catalyzed the landing of many autonomous driving cars. The Defense Advanced Research Projects Agency (DARPA) has developed the Strategic Computer Program (SCP), which it hopes will benefit from

rapid advances in Computer architecture, software, and chip design, and push AI technology to new heights. As one of the sponsored research institutions, Carnegie Mellon University formed NAVLAB in 1984, and launched the first on-road autonomous driving car NavLab-5 in 1995 (as shown in Figure 1.2). NavLab-5 was built based on a 1990 Pontiac Trans Sport, with a sensing system including a Sony RGB camera, a laser rangefinder, GPS, fiber optic damping gyro, optic encoder, and a computing system including a SPARCLX portable workstation and an HC11 microcontroller. With an algorithm based on the early neural network, NavLab-5 had the ability of analyzing the road condition to steer autonomously from learning human driving behaviors. This entrusted the intelligence to the car. In this period, the foundation of sensors selection, algorithms and research direction of modern autonomous driving car has been set.

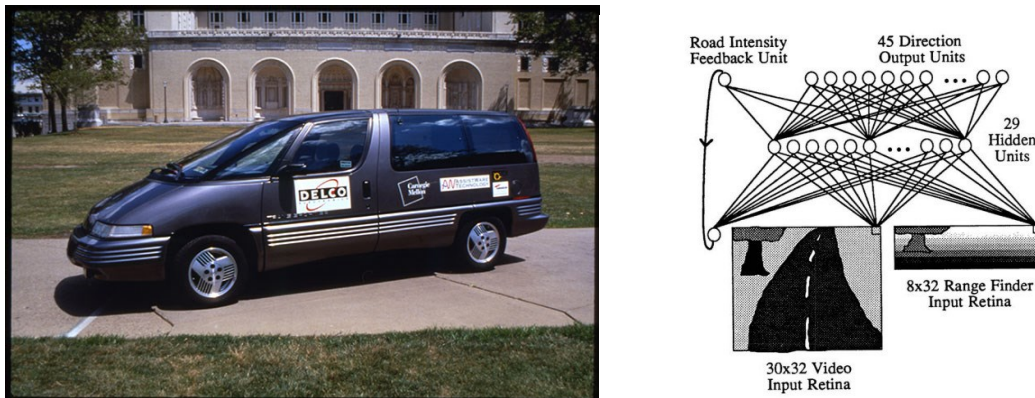


Figure 1.2: NavLab-5 (left side) and the artificial neural network for learning human driving skills (right side). Image credit: NavLab 5 via Carnegie Mellon University.

When it came to 21st century, an autonomous driving challenge named “DARPA Grand Challenge” (as shown in Figure 1.3) by DARPA has attracted ICT companies and Silicon Valley such as Google from all over the world to join in the research and development of autonomous driving cars, which also causes the "intelligent" transformation of the traditional automobile industry and gives birth to a trillion-dollar industry. DARPA challenge has been held for three times, and the “Stanley” built based on a Volkswagen Touareg from Stanford racing team won the second challenge in 2005 among all the 43 teams (the hardware structure of Stanley has been shown in Figure 1.3).



Figure 1.3: Poster for First DRAPA Grand Challenge (top side), and some of the hardware structure of Stanley, the entrant vehicle developed based on a Volkswagen Touareg. Image credit: The DARPA Grand Challenge: Ten Years Later via DRAPA.

By this challenge, DARPA has successfully explored the potential of autonomous driving cars, is also the basis route of hatching it. The hardware system should be composed basically by camera, lidar, millimeter wave radar, wire control system and the computing cell, while the sensing and fusion, object detection and positioning, path and action planning algorithm should compose the software system. The modern autonomous driving system is the combination of the hardware and software systems together. What we are researching and developing currently is about carrying out more in-depth and refined technological iterations on this basic route.

The series of challenges initiated by DARPA has promoted the birth of an autonomous driving ecosystem composed of inventors, engineers, programmers, and developers. It also contributed to the rise of autonomous vehicle technology entrepreneurship and investment. Google, Tesla, Uber, Baidu, etc. have successively announced plans to develop autonomous vehicles, making no secret

of their ambitions in this emerging industry. At the same time, many autonomous driving startups such as Velodyne and Aurora have sprung up. In the scorching outlook of Internet companies, even the conservative traditional automakers and the supply chain behind them are "forced" to join the "autonomous vehicle arms race" because this is a matter of life and death. Till now, the concept of autonomous driving has been divided to 5 levels from pure human driving to fully automated. Lots of R&D projects have landed several autonomous driving cars of different levels, and some companies even announced that their AV of L5 ADAS will be on the road in 2021. No matter how, autonomous driving technologies is evolving and will continue to evolve, and it will gain much more diversity as more and more companies join in.

What is autonomy? Autonomy is the ability to make your own decisions. In humans, autonomy allows us to do the most meaningful, not to mention meaningless, tasks. This includes things like walking, talking, waving, opening doors, pushing buttons, and changing light bulbs. In robots, autonomy is really no different. Autonomous robots, just like humans, also have the ability to make their own decisions and then perform an action accordingly. A truly autonomous robot is one that can perceive its environment, make decisions based on what it perceives and/or has been programmed to recognize conditions and then actuate a movement or manipulation within that environment. With respect to robot mobility, for example, these decision-based actions include but are not limited to some basic tasks like starting, stopping, and maneuvering around obstacles that are in their way.

Autonomous robot can be useful in many application scenarios. In here, we list some of the application of autonomous robots, and we define the robot proposed in this thesis as a research and education robot.

- Delivery Robot
- Construction Robot
- Research and Education Robot

At the current time, autonomous driving is not only concentrated on cars, but it can also be implemented on drones or robots as well. The core concept of typical autonomous driving technology is about sensing from the environment, controlling, and making decisions. This shows a huge similarity with the concept of robots, especially intelligent robots. Therefore, some

technologies or algorithms developed from robots can be shared and implemented with vehicles, while developing and testing directly on a robot is more convenient and safer than straightly put a test car on the road. It can be a platform for developing advanced autonomous technologies and algorithms. Also, autonomous driving robots have several unique features and applicable scenarios. Based on the thoughts and relationship of autonomous driving and robots, an autonomous driving robot was built and landed (as shown in Figure 1.4)



Figure 1.4: Overview of the robot proposed in this thesis, taken in University of Michigan – Dearborn, Sep 17th 2020.

1.2 Main Objectives

The main objectives of this thesis include the following five parts as proposed:

- a). Build an autonomous driving robot based on a Segway self-balancing robot with sensors including a 16-channel solid LiDAR, a RGB camera, wheel encoders, a gyroscope, and actuator including servos and microcontrollers.
- b). Based on this robot, realize several basic functions including movement control, sensing, and wheel odometer.
- c). Based on these basic functions, develop and realize several autonomous driving functions including obstacles avoidance, 2D simultaneously localization and mapping (2D SLAM), camera & LiDAR data fusion and path planning.
- d). Test the performance of the robot and integrate all the autonomous driving functions mentioned to the robot simultaneously without decrease the response speed.
- e). Establish a simulation environment and model based on ROS and Gazebo with all of the autonomous driving functions mentioned in 1.2.c and implement some other algorithms to the simulation to validate and explore the potential and capability of some possible future research work underneath this robot.

Besides, the following several technologies and algorithms will be discussed and presented mainly in this thesis including:

- a). Two-wheel differential kinematics model.
- b). Microcontroller and servo.
- c). Obstacle Avoidance.
- d). LiDAR and point cloud.
- e). Sensor data fusion.
- f). SLAM and filtering.
- g). Path planning.

Some other technologies and principle used in the simulation part will not be within the scope of our discussion.

1.3 Thesis Outline

In CHAPTER 1 the history of autonomous driving and the relationship between autonomous driving and robots have been introduced briefly. In CHAPTER 2 some related works about the technologies and algorithms presented in this thesis mainly including 1.2 c, d, e, and f will be introduced.

In CHAPTER 3 the approaches used through all the process of developing the autonomous driving robot will be presented.

In CHAPTER 4 the overall testing process and results of the robot will be presented and analyzed.

In CHAPTER 5 some summary and conclusion about the performance and practical application value of this thesis will be presented. In addition, we will discuss the prospects for future research works inspired by this thesis.

CHAPTER 2

LITERATURE REVIEW

2.1 Autonomous Robot

According to JIRA (Japanese Industrial Robot Association), the common and general robotics can be classified into 6 classes:

- Class 1: Manual Handling Device
- Class 2: Fixed-Sequence Robot
- Class 3: Variable Sequence Robot
- Class 4: Playback Robot
- Class 5: Numerical Control Robot
- Class 6: Intelligent Robot

Specifically, the robot proposed in this thesis can be classified into Class 6, intelligent robot for the reason that the robot has a certain degree of learning ability. There is actually existing another kind of classification of robotics based on control, as shown in Figure 2.1. Autonomous robot has been divided into two types including pre-programmed and self-learning robot. For the robot proposed in this thesis, there is no clear line of demarcation to define which type the robot belongs. At the most of time, the robot was guided by pre-programmed functions to take action. However, the robot also has the ability of environment perception and take action by itself. Thus, in this thesis, we define this robot as an autonomous driving robot based on the ability and functionality the robot can perform.

Classification of Robots (Based on Control)

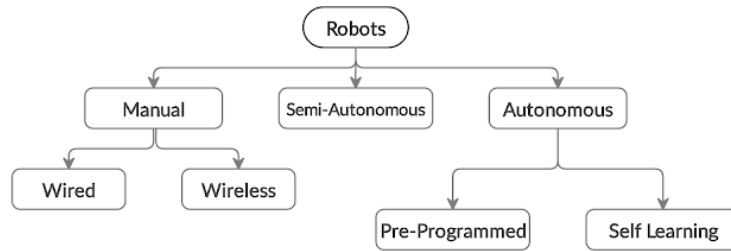


Figure 2.1: One kind of robotics classification based on control.

As mentioned before, the three most essential features of an autonomous robot include sensing, making decisions, and taking actions, as shown in the scheme in Figure 2.2.[35] To realize and fulfill the functionality requirements of autonomous robot, people usually implement several types of sensors, microcontrollers, and actuators. Till now, general sensors that can be implemented to robots, especially autonomous robots can be listed in Table 2.1.

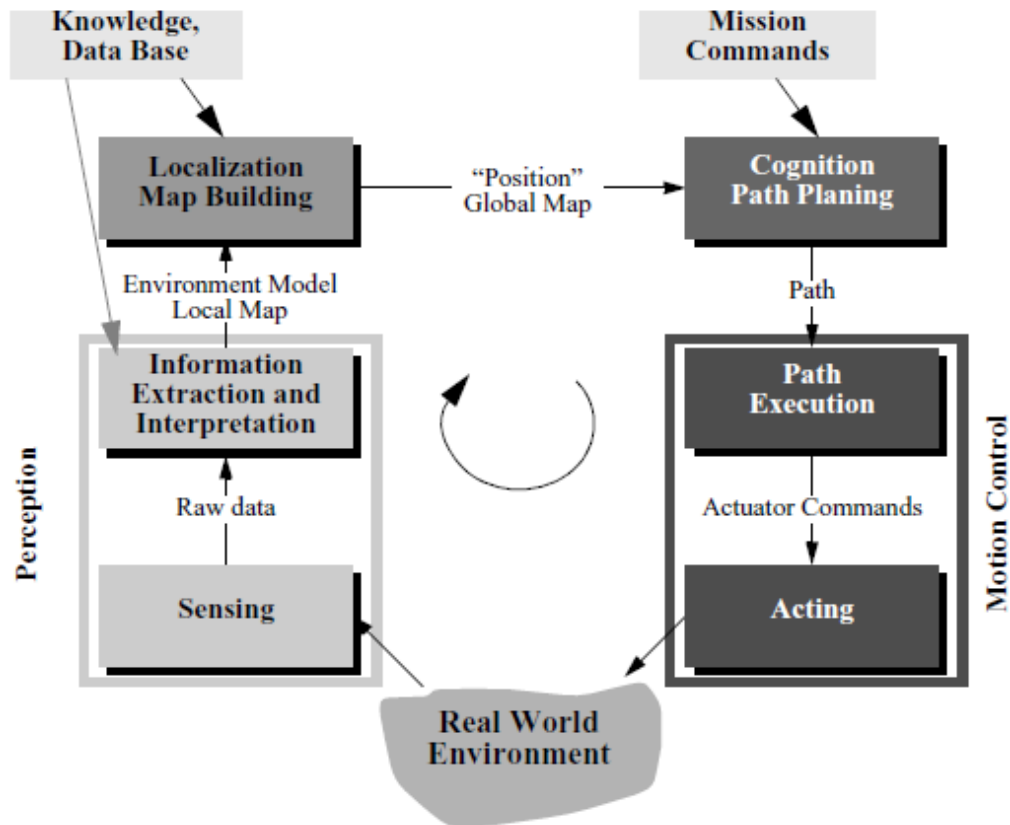


Figure 2.2: General autonomous robot scheme. [35]

Table 2.1: Overview of common sensors for autonomous robotics. A: active; P: passive; PC: proprioceptive; EC: exteroceptive. [35]

General Classification	Sensor / Sensor System	PC or EC	A or P
Wheel/motor sensors (Speed and position)	Brush encodes	PC	P
	Potentiometers	PC	P
	Optical encoders	PC	A
	Magnetic encoders	PC	A
	Inductive encoders	PC	A
	Capacitive encoders	PC	A
Heading Sensors (Orientation of the robot in relation to a fixed reference frame)	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A/P
Ground-based Beacons	GPS	EC	A
	Active optical or RF beacons	EC	A
	Active ultrasonic beacons	EC	A
	Reflective beacons	EC	A
Active Ranging (Reflectivity, time-of- flight, and geometric triangulation)	Reflectivity sensors	EC	A
	Ultrasonic sensors	EC	A
	Laser rangefinder (laser scanner)	EC	A
	Optical triangulation (1D)	EC	A
	Structured light (2D)	EC	A
Motion/speed sensors (Speed relative to fixed or moving objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision-based sensors (visual ranging, whole- image analysis, segmentation, object recognition)	CCD/CMOS cameras Visual ranging packages Object tracking packages	EC	P

Also, people use several types of microcontrollers and actuators for autonomous robotics. In here we will not list them as comprehensive as for sensors, we only list some of the most common ones.

Microcontrollers:

- Single chip microcontroller: STM32, Arduino, ARM, 8051, Atmel AVR
- Raspberry Pi
- Nvidia Jetson

Actuators:

- Servos
- Motors
- Artificial Muscle
- Pumps
- Fans


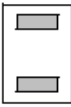
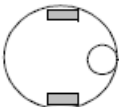
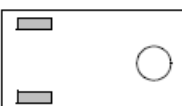

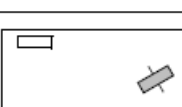
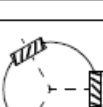
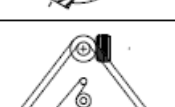
# of wheels	Arrangement	Description	Typical examples
2		One steering wheel in the front, one traction wheel in the rear	Bicycle, motorcycle
		Two-wheel differential drive with the center of mass (COM) below the axle	Cye personal robot
3		Two-wheel centered differential drive with a third point of contact	Nomad Scout, smartRob EPFL
		Two independently driven wheels in the rear/front, 1 unpowered omnidirectional wheel in the front/rear	Many indoor robots, including the EPFL robots Pygmalion and Alice
		Two connected traction wheels (differential) in rear, 1 steered free wheel in front	Piaggio minitrucks
		Two free wheels in rear, 1 steered traction wheel in front	Neptune (Carnegie Mellon University), Hero-1
		Three motorized Swedish or spherical wheels arranged in a triangle; omnidirectional movement is possible	Stanford wheel Tribolo EPFL, Palm Pilot Robot Kit (CMU)
		Three synchronously motorized and steered wheels; the orientation is not controllable	"Synchro drive" Denning MRV-2, Georgia Institute of Technology, I-Robot B24, Nomad 200

Figure 2.3: General wheel configuration for rolling mobile robots. [35]

With the implementation of sensors, microcontrollers and actuators, the final step of building an autonomous robot is a carrier to hold all the hardware. There are several types of robotics differed in types of locomotion including crawl, sliding, walking, wheeled and so on. In this thesis, we chose wheeled chassis to carry the hardware. More specifically, we chose two-wheel differential drive mode, the arrangement has been shown in Figure 2.3.

One of the most famous autonomous for research and education purpose is the TurtleBot. TurtleBot is a low-cost, personal robot kit with open-source software especially robot operating system (ROS). TurtleBot was created at Willow Garage by Melonee Wise and Tully Foote in November 2010.[44] With TurtleBot, you will be able to build a robot that can drive around your house, see in 3D, and have enough horsepower to create exciting applications. TurtleBot has evolved to the third generation till now and the features of TurtleBot is pretty much similar to our robot:

- Two-wheel differential drive
- Equipped with multiple sensors
- Modular sensors and functions
- Programmable

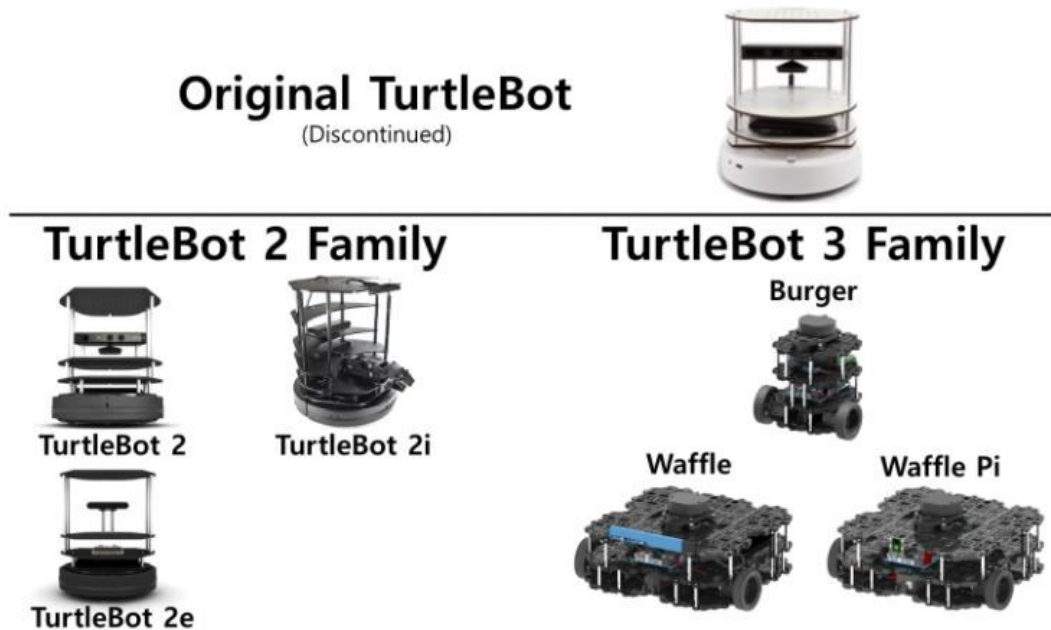


Figure 2.4: Examples of TurtleBot Family. Image credit: www.TurtleBot.com

After introducing some related works about the overview of autonomous robots, next we will introduce some theoretical background about the functions proposed in our robot.

2.2 Obstacle Avoidance

Obstacle avoidance is one of the most essential functions for an autonomous driving robot for the reason of keeping the robot safe at all the time. The two main concern when implementing obstacle avoidance to a robot includes the choose of sensor and algorithm. There are several types of sensor choices for different applicational scenarios as shown in Table 2.2, while the algorithm chosen might not vary significantly because the principle shows the same.

Table 2.2: Advantages and limitation of various sensors within USVS. [24]

Sensors	Advantages	Limitations
Millimeter-Wave Radar	<ul style="list-style-type: none"> • Long detecting range. • Good velocity estimates. • All-weather and broad-area imagery. 	<ul style="list-style-type: none"> • Limited small and dynamic target detection capability. • Motion distortion;
LiDAR	<ul style="list-style-type: none"> • High depth resolution and accuracy. • Suitable for both indoor and outdoor scenario. • Wide scan range. 	<ul style="list-style-type: none"> • Angular resolution both vertically and horizontally. • Sensitive to motion. • High cost.
Visual Sensor	<ul style="list-style-type: none"> • High lateral and temporal resolution. • Simplicity and low weight. 	<ul style="list-style-type: none"> • Low depth resolution and accuracy. • Challenge to real-time implementation. • Sensitive to light and weather.
Infrared Sensor	<ul style="list-style-type: none"> • Applicable for dark conditions. • Low power consumption. 	<ul style="list-style-type: none"> • Indoor use only. • Impressionable to interference and distance.

Algorithms for obstacle avoidance can be divided to two kinds. The traditional algorithms including Artificial Potential Field (APF) and Virtual Force Field (VFF) [21] usually have satisfactory real-time performance and high safety margin, but it cannot achieve good results in a dynamic environment. The opposite one is intelligent optimization algorithms including Fuzzy Logic Algorithm (FLA), Genetic Algorithm, Rapidly Random-exploring Trees (RRT) and so on.

The most notable advantage for intelligent optimization algorithms is good performance in dynamic environment. The response is rapid for moving obstacles, which can improve the safety. The completeness of these algorithms is to deal with the complex conditions of real roads and possible potential unknown threats.

There is a famous theory in artificial intelligent goes as “There’s no free lunch”. For different practical applicational scenarios, the choose of sensors and algorithms should be flexible to balance the cost and performance. A simple logic for obstacle avoidance can be described as shown in Figure 2.5. This kind of logic is always implemented with grid mapping. The information provided by grid map has lower resolution which is suitable for simple logic. [8]

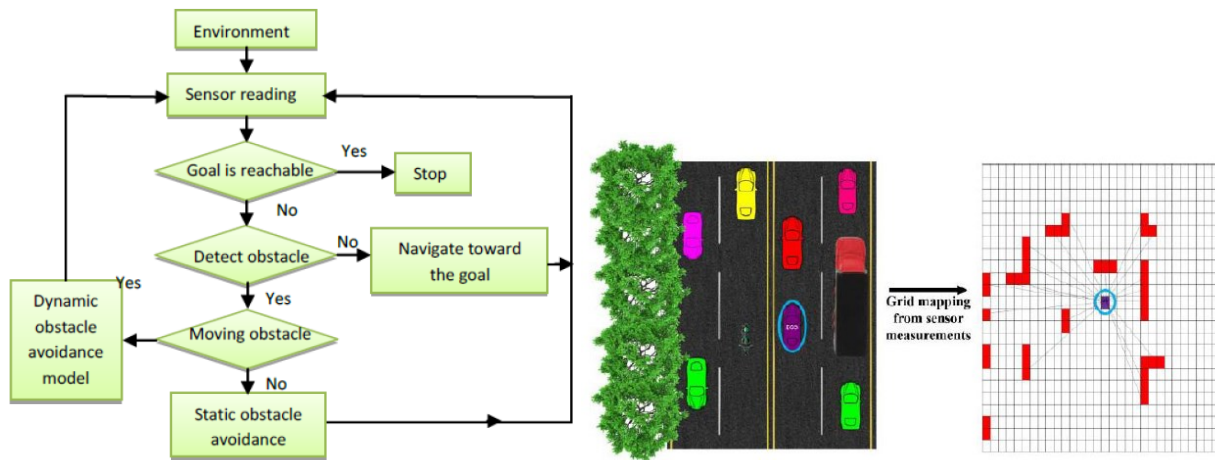


Figure 2.5: One example of obstacle avoidance logic (left) and guiding 2D grid map (right). [8]

There is another logic for obstacle avoidance that published recently called the clearance considering the uncertainty of the robot motion (CURM) as shown in Figure 2.6.[24] This algorithm is a kind of local obstacle avoidance that designed based on velocity control, where CURM is the smallest value in the uncertainty ellipse of the reference velocity. Our method shows similarity with both the CURM and the simple logic in Figure 2.5, and it will be presented later in CHAPTER 3. This kind of logic shows high performance in dynamic environment, where the global map is uncertain, and obstacles are moving.

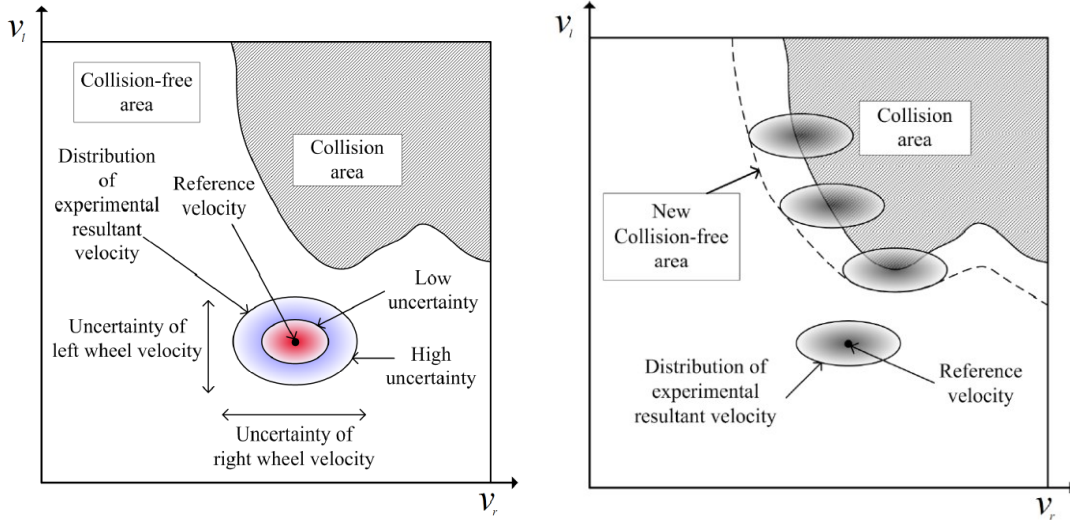


Figure 2.6: CURM configuration. [24]

Obstacle avoidance technology is always implemented with the company of path planning (routing), as part of the decision-making processes.

2.3 LiDAR and Point Cloud

As one of the most crucial sensors in autonomous driving technologies, laser radar, or LiDAR has the ability of sensing the environment comprehensively in stereo. It also has an extensive application in researching and industry. LiDAR can be divided to several different types including 2D/3D, single/multi-channel, 360°/180° and so on, while in this thesis we focus the application of 3D multi-channel LiDAR mainly on autonomous driving.

LiDAR can be useful in almost every functions of autonomous driving. A typical LiDAR for autonomous driving is usually a solid multi-channel one, which uses time-of-flight (TOF) [32] when measuring, as shown in Figure 2.7. The received optic pulses will be decoded as plenty of points containing both position and reflection intensity information, which are called point cloud. A typical point cloud data after decoding is formatted as $[x \ y \ z \ intensity]$.

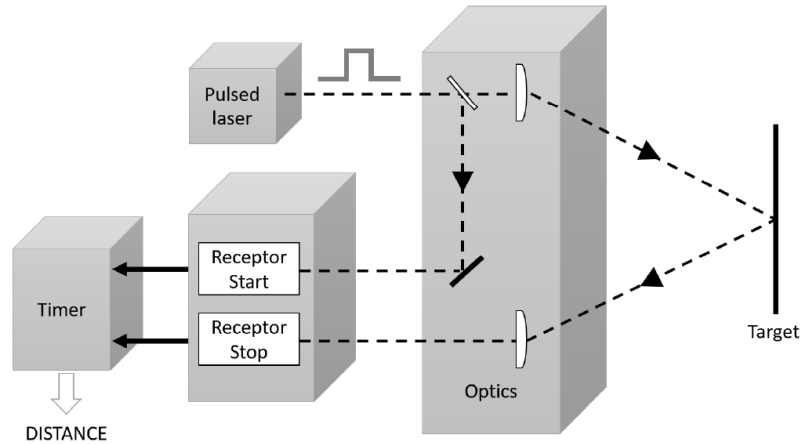


Figure 2.7: FOT configuration of LiDAR. [30]

In this thesis, a 16-channel solid LiDAR will be implemented to our robot for supplying point cloud data of realize Occupied Grid Obstacle Avoidance, Camera & LiDAR Data Fusion and 2D SLAM.

2.4 Sensor Data Fusion

Sensor data fusion is a powerful technology which can combine different type of sensors together, lead them to take advantages and complement disadvantages together. The application of sensor data fusion in autonomous driving can be simple such as object co-detection, or more complicated such as 3D reconstruction, and semantic map construction, as shown in Figure 2.9.

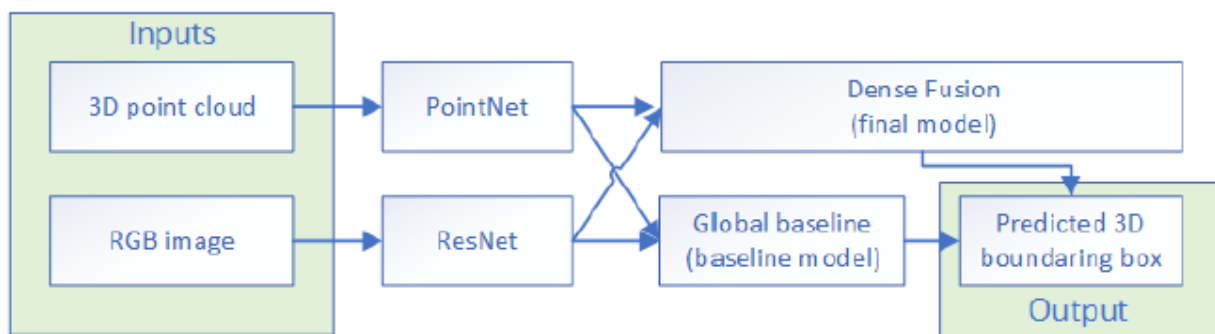


Figure 2.8: One example flow chart of fusing data from LiDAR and camera. [23]



Figure 2.9: One typical application of data fusion, 3D object co-detection and segmentation. The segmented parts also have semantic information. [33]

One example of fusing LiDAR and camera together is to use machine learning, such as the flowchart shown in Figure 2.8. However, for precise calibration, the precondition of fusing sensors is to know the spatial relative relationship, which means the sensors should be calibrated in advance to generate the intrinsic and extrinsic matrices of both camera and LiDAR. For some open-source dataset such as KITTI [2], the intrinsic and extrinsic matrices have already been calibrated, but since this robot was built on our own, the camera and LiDAR needed to be calibrated from scratch.

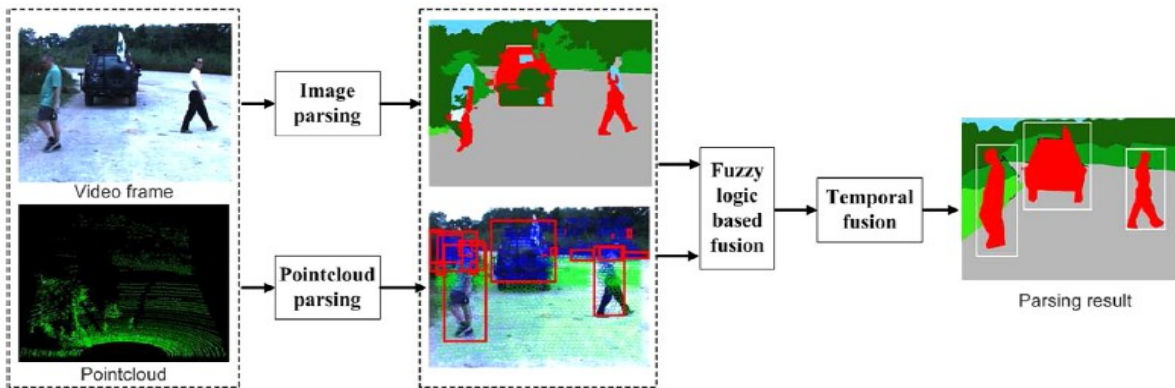


Figure 2.10: One example of fuzzy logic based data fusion. [32]

The key of calibration is to extract the identical features from both image and point cloud then match them together. There are several existing methods for calibration as shown in Figure 2.10 uses fuzzy logic to fuse the parsed image and point cloud together, no matter what kind of object is providing the features. [32] Another method is more suitable and robust for fixed position sensors. The key about this method is to use a checkerboard as the landmark. First it detects and estimate the corner of each checkers both in image and point cloud, then with the intrinsic matrix calibrated from camera alone, it can re-project each corner from point cloud back to image in a same coordinate and calculate the reprojection error. Some improvements can be implemented with the application of refinement and optimization algorithms. For example, in ILCC [31], an optimization cost function based on the constraints of the correspondence between the intensity and color was formulated, as shown in Figure 2.11. Our method of calibration and fusing data from LiDAR and camera is based on ILCC, which will be presented in CHAPTER 3 particularly.

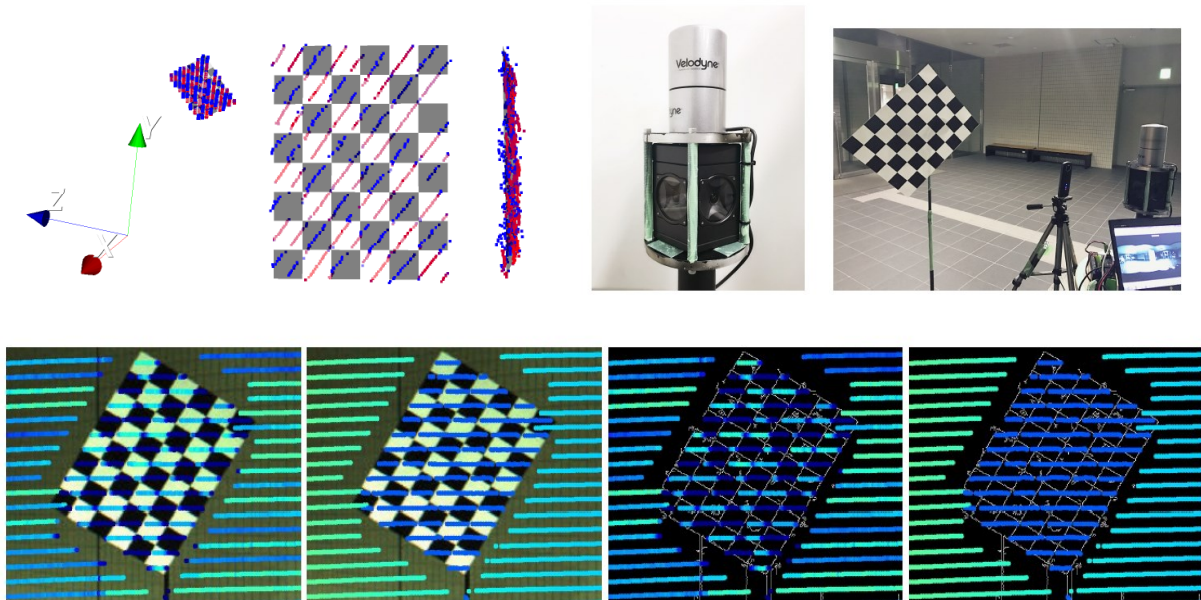


Figure 2.11: Hardware setup and corner extraction for ILCC. [31]

2.5 SLAM and Filtering

Localization and mapping are one of the major focus of autonomous driving and robotics because it is usually the prerequisite of path planning. SLAM comprises the simultaneous estimation of the state of a robot equipped with sensors, and the construction of a model (map) of

the environment that the sensors are perceiving. The need to use a map of the environment is twofold. First, the map is often required to support other tasks; for instance, a map can inform path planning or provide an intuitive visualization for a human operator. Second, the map allows limiting the error committed in estimating the state of the robot. In the absence of a map, dead-reckoning would quickly drift over time; on the other hand, using a map, e.g., a set of distinguishable landmarks, the robot can “reset” its localization error by re-visiting known areas (so-called loop closure). [40]Therefore, SLAM finds applications in all scenarios in which a prior map is not available and needs to be built. In autonomous driving and robotics map can be divided into several types depends on the usage and information contained as shown in Figure 2.12 However, in this thesis we focus only on grid map since the primary application of the map is for path planning.

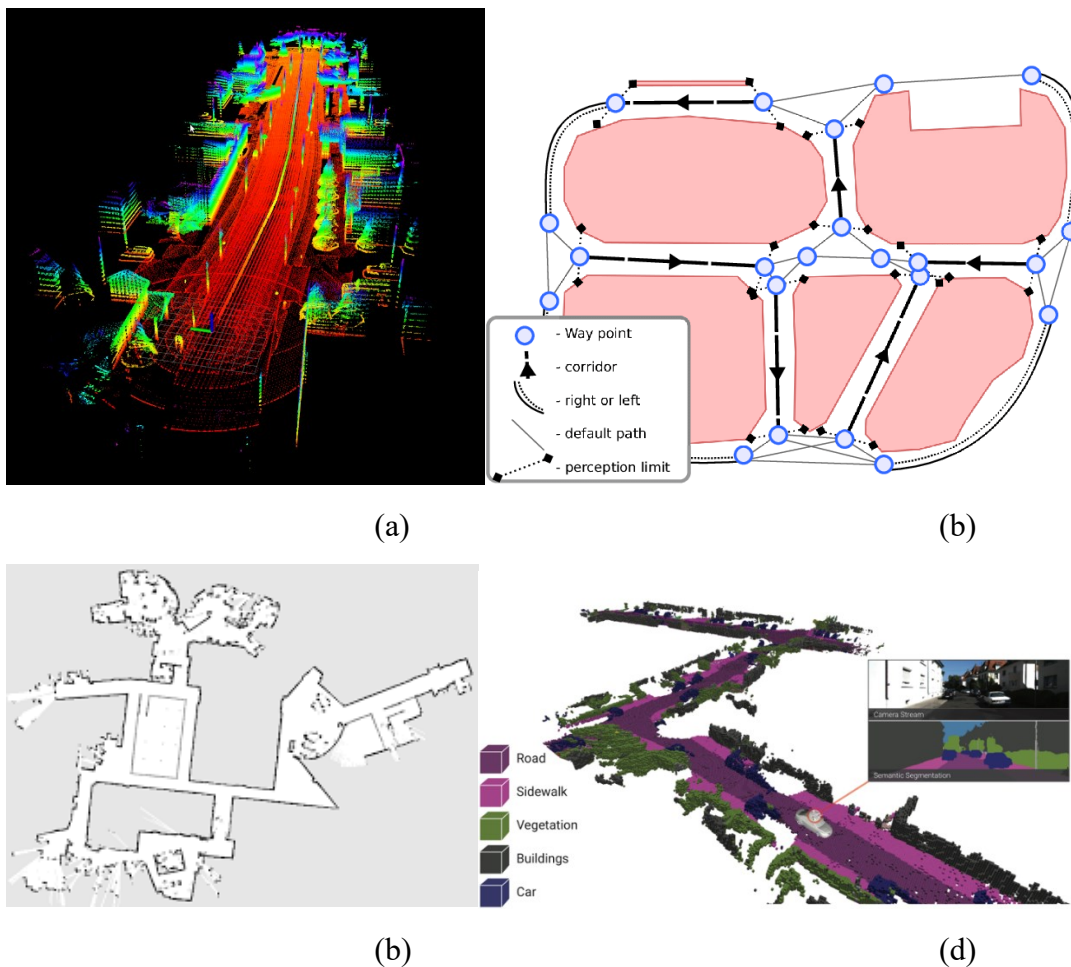


Figure 2.12: Different type of map. (a): 3D point cloud map. (b): 2D semantic map. (c): 2D grid map. (d): 3D semantic map. [38]

SLAM also can be divided into several types. People usually divided SLAM into two manifolds by the usage of sensor including vision SLAM and lase SLAM. Both these two approaches of SLAM have their advantages and disadvantages, as shown in Table 2.3.

Table 2.3: Comparison between lase & vision SLAM in some aspects.

	Lase SLAM	Vision SLAM
Cost	High	Much lower
Application Scenario	More indoor, but can do outdoor	Both indoor and outdoor, high optic dependence
Accuracy of Map	High, can be used for navigation or path planning directly	Lower
Usability	Collect point cloud with depth information directly	Indirectly, more steps for collecting depth information
Hardware Setup	Relatively larger and heavier	Light and portable

Compared with vision SLAM, lase SLAM is much suitable for the application of the robot proposed in this thesis. lase SLAM can be divided into two types including Filter-based and Graph-based depends on the algorithm implemented. Filter-based SLAM [42] modeled the localization and mapping process as a probabilistic problem, which use a probabilistic filter to estimate the robot's pose simultaneously in each frame with the input of lase scan and odometer. Graph-based SLAM [42] create sub-graphs to represent the state and map of the robot and use nonlinear least squares to optimize those graphs. Table 2.4 presents the development and features of lase SLAM.

The quality of map can be critical for affecting the performance of the robot. A poorly constructed map with low accuracy may provide fallacious coordinates for robots and may lead to a crash. Plus, the robot must know the current location of itself, which is a prerequisite of the overall mapping process. Based on the consideration of our robot's practical application and software development environment, the Optimal RBPF 2D SLAM [21] is the most suitable algorithm, which is lighter than graph-based methods with high accuracy compared to other filter-based methods.

The abbreviation RBPF represents Rao-Blackwellized Particle Filter.[46] As an extension of particle filter, RBPF is a powerful tool for solving state estimation problems. As mentioned before,

filter-based SLAM modeled the localization and mapping process as a probabilistic problem, which can be formulated as a posterior distribution over the state variables consisting of the robot map $m = \{m_i\}$ and robot trajectory $x_{1:t} = \{x_1, \dots, x_t\}$ conditioned on the sequence

Table 2.4: Configuration of lase SLAM through development.

Year	SLAM	Sensor	Feature
1988	EKF-SLAM	2D LiDAR	Only feature map built, large computational complexity and poor robustness
2002	FastSLAM	2D LiDAR	First algorithms for building grid map; High memory consumption and particle dissipation
2007	Gmapping	2D LiDAR	Less particle dissipation, high odometer dependency
2010	Optimal RBPF	2D LiDAR	Best performance for filter-based SLAM, relative light-weighted
2010	Karto SLAM	2D LiDAR	First graph-based SLAM algorithm, high time consumption and non-real time
2011	Hector-SLAM	2D LiDAR	No odometer needed; Sensitive to initial value, map will drift when robot rotated
2014	LOAM	3D LiDAR	First 3D lase SLAM algorithm; Assume the robot moves in constant speed; No loop-closure
2015	V-LOAM	3D LiDAR & Vision	High accuracy and robustness; Fuse lase and vision sensor together;
2016	Cartographer	2D LiDAR	Graph-based 2D SLAM; Similar performance with Optimal RBPF
2016	VELO	3D LiDAR & Vision	With loop-closure; Less map drifting
2018	IMLS	3D LiDAR	Pure lase input, no rely on GPS, IMU, vision sensors with less map drifting

of sensor observations $z_{1:t} = \{z_1, \dots, z_t\}$ and control commands $u_{1:t} = \{u_1, \dots, u_t\}$. It uses several numbers of particles to represent the posterior and after each interval it will update these particles and resample them to acquire the right location of robot and forward to the next interval. Based on this assumption, RBPF SLAM split localization and mapping apart, construct the map after the robot's current pose was determined. The relatively simple process of particle filter causes

it lighter than graph-based methods while the resampling step also cause a common failing called particle degeneracy. Particle degeneracy means after several iterations of resampling, some particles carrying correct information might be dumped and the diversity of particles might downgrade. In terms of this problem, RBPF applied two improvements. The first one is called selective resampling, which set a threshold at the resampling step. This will only resample those particles with large weights whose distance between current distribution center is smaller. The second one is to use distribution of improvement proposal, which means it will consider the result of the most recently sensor reading when give a weighting to particles, since the observation of sensors are always more accurate than control commands. With these two improvements, RBPF has fewer times of resampling and numbers of particles to prevent the particle degeneracy problem, while it also gained higher accuracy.

The research about RBPF SLAM (or more widely the filter-based SLAM) is still ongoing. Many researchers proposed vary kinds of methods to improve the performance, e.g. [16] [19] [21] [25] . But those works will not be discussed too much since the theoretical background of our works is already enough. Next in CHAPTER 3, our method will be presented in detail about how we implement RBPF to our robot, what kind of adjustments we have made and the way we make it real-time.

2.6 Path Planning and Reinforcement Learning

In this part, the background of robot path planning will be mainly introduced since the topic of thesis is autonomous driving robot. Besides, the primary purpose of this thesis is to build an autonomous driving robot which can be also used as an algorithm development platform. Hence, some of the advanced technologies related to path planning will also be introduced. In here we will mainly focus on the most popular research direction, reinforcement learning. [38]

Path planning, or as known as routing, navigation technologies are well known while playing an essential role in autonomous driving and robotics. Generally, people describe the path planning problem as a process or activity to plan and direct a route or path from a position to a goal on the map. In fact, the three general problems of path planning include localization, mapping, and motion control, which has been introduced before and will be discussed in detail as the main focus of this thesis. Path planning research of autonomous driving and robotics has attracted attention since the

1970s. [37] Over the past several years and recently, research in this area has increased due to the reason that autonomous robots are now applied in various applications. Thus, through many years of development and evolution, the classification of path planning has been divided into many domains, as shown in Figure 2.13. In here, we focus only on 2D environment since the robot proposed in this thesis can be classified as a ground autonomous mobile robot. Under the 2D environment domain, the path planning technologies can also be divided into several domains according to the emphasis, as shown in Figure 2.13.

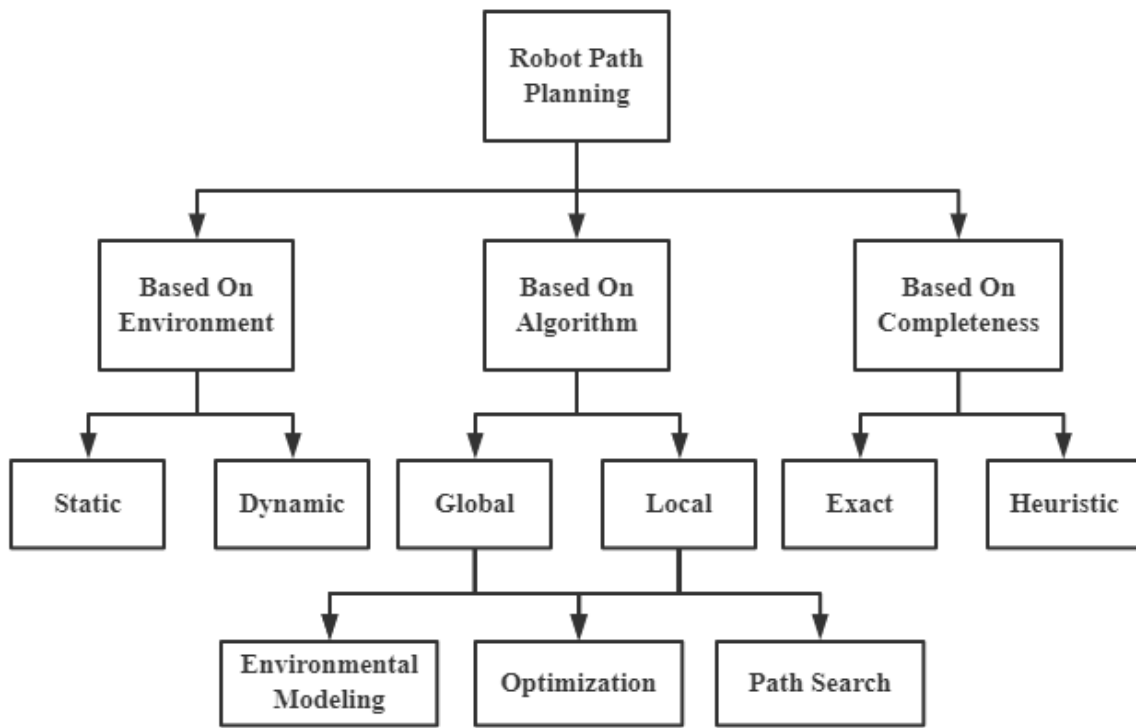


Figure 2.13: General classification of robot path planning.

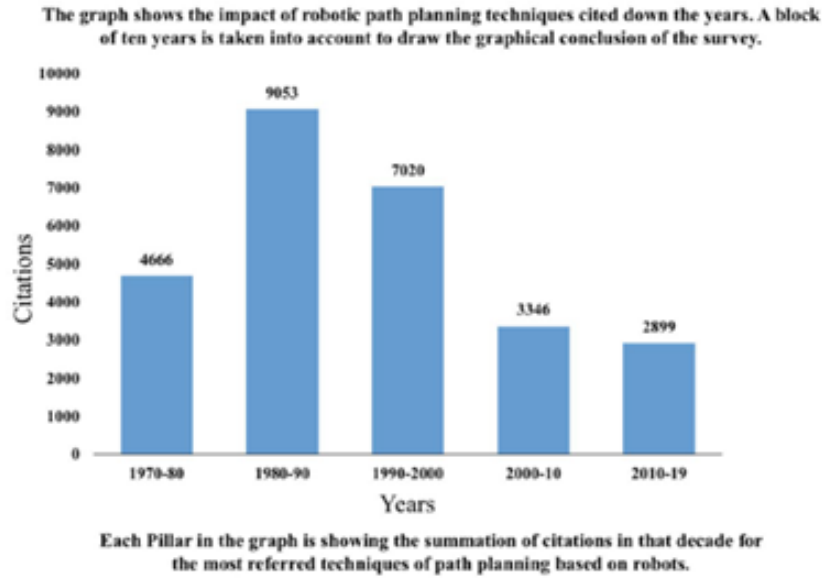


Figure 2.14: Citation of robotic path planning techniques through years. [6]

Figure 2.14 presents a survey about the impact of robotic path planning algorithms cited down the years. Obviously, the heyday of the path planning algorithm development began from 1980s and tend to stable gradually after 2000s. [6] One of the major possible reasons is the fever of nature-inspired algorithms. The nature-inspired algorithms for path planning have a wide application including Artificial Neural Networks, Ant Colony Optimization, Bee Colony, Firefly Algorithm, Particle Swarm Optimization, Bacteria Foraging, BAT Algorithm and so on. [6] However, the cited papers tend to include more on doing advancements on some prominent algorithms such as A star (A^*) algorithm, Rapidly Exploring Random Tree and so on. Also, the rapid development of artificial intelligence also has an impact of path planning technologies, which is embodied as the implementation of Reinforcement Learning (RL).

Hence, in this chapter, some related works about path planning and reinforcement learning under in the context of this thesis will be introduced including A star (A^*) algorithm, Dijkstra algorithm, A star Heuristic algorithm, Greedy algorithm and reinforcement learning.

For A stat (A^*) and A^* Heuristic: A^* is actually a kind of search algorithm, especially in graph traversal and path search. It can be implemented on path planning when using a grid map to represent the environment. More specifically, A^* is an informed search algorithm, or better known

as best-first search, meaning that it is formulated in terms of weighted graphs. Starting from a specific starting node of a graph, it aims to find a path to the given destination node with the smallest cost (shortest path or time, etc.). This is realized by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied. [7]

At each iteration of its main loop, A* needs to determine which of its paths to extend based on the cost of the path and an estimate of the cost required to extend the path to the destination. This can be formulated as a minimization:

$$f(n) = g(n) + h(n)$$

Where n is the next node on the path, $g(n)$ is the cost of the path from start node to n , and $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to destination. A* terminates when the path it chooses to extend is a path from start to destination or if there are no paths eligible to be extended. In here, since we are using grid map instead of graph, an example of implementing A* can be described as shown in Figure 2.15.

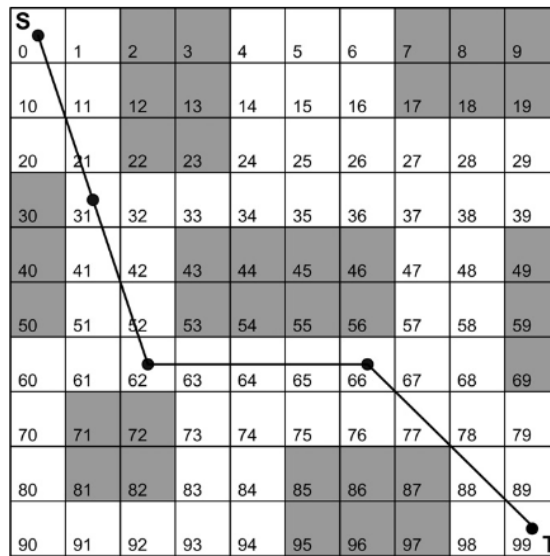


Figure 2.15: One example of path planning based on 2D grid map. [28]

For Dijkstra: Similar with A*, Dijkstra is a kind of search algorithms by minimizing the cost from start to destination. The method of describing the environment used by Dijkstra is the same as A* that, it uses nodes to define each position the robot can reach. However, the difference is,

Dijkstra uses labels that are positive integers or real numbers, which are totally ordered. It can be generalized to use any label that are partially ordered, provided the subsequent labels (a subsequent label is produced when traversing an edge) are monotonically non-decreasing.

The original algorithm uses a min-priority queue. Let the node at which we are starting to be called the initial node, and the distance of node Y be the distance from the initial node to Y . Then the process of Dijkstra algorithm can be described as assigning some initial distance values and trying to improve them step by step:

- Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
- Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. Set the initial node as current.[]
- For the current node, consider all of its unvisited neighbors and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B through A will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.
- When we are done considering all of the unvisited neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
- If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
- Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3. [7]

For Greedy algorithm: Greedy algorithm, or as known as greedy strategy is actually a kind of method about solving problems, which is mostly applied in optimization problems. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the

entire problem. The greedy algorithm can be implemented to solve the problem if both properties below are satisfied:

- Greedy choice property: A global optimal solution can be reached by choosing the optimal choice at each step.
- Optimal substructure: A problem has an optimal sub structure if an optimal solution to the entire problem contains the optimal solutions to the sub-problems.

For path planning problem, greedy algorithm can be applied to Dijkstra algorithm since Dijkstra algorithm satisfied the two properties.

For Reinforcement Learning: The reinforcement learning is a unique classification of machine learning alongside from supervised and unsupervised learning, as shown in Figure 2.16. The entire area about reinforcement learning has already developed into a vast field and implemented to various theoretical or practical application scenarios. So, in here, we will only introduce the part of the reinforcement learning area which will be used for path planning of our robot. The primary process of reinforcement learning can be described as an interaction between an intelligent agent and the environment. The intelligent agent will take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning can also be divided into many different classifications. We will not list and introduce all of these categories but focusing on Q-learning, a model-free reinforcement learning algorithm. [41]

Q-learning algorithm can be described as shown in the pseudocode in Figure 2.17. Before learning begins, Q is initialized to a possibly arbitrary fixed value (chosen by the programmer). Then, at each episode or time t the agent selects an action a , observes a reward R , enters a new state and Q is updated. The core of the algorithm is a Bellman equation is about updating the simple iteration value $Q(S, A)$ using the weighted average of the old value and the new information:

- R is the reward received when moving from state S to S' .
- $Q(S, A) - \alpha Q(S, A)$ is the current value weighted by the learning rate. Values of the learning rate near to 1 made faster the changes in Q .
- αR is the reward $R = R(S, A)$ to obtain if action A is taken when in state S .
- $\alpha \gamma \cdot \max_a Q(S', a)$ is the maximum reward that can be obtained from state S' .
- An episode of the algorithm ends when state S is a final or terminal state.

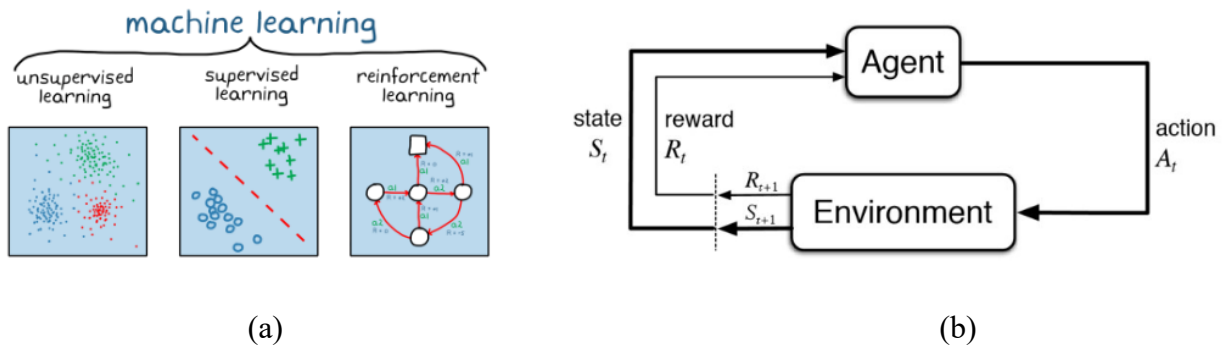


Figure 2.16: (a): General classification of machine learning. (b): Simple representation of reinforcement learning.

The reason of choosing Q-learning is that one of the most classic practical application of Q-learning is the maze puzzle problem, which is similar to path planning on a 2D grid map of this thesis. A typical maze puzzle problem can be described in Figure 2.18.

```

Q-learning: An off-policy TD control algorithm
Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
Repeat (for each episode):
  Initialize  $S$ 
  Repeat (for each step of episode):
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
     $S \leftarrow S'$ 
  until  $S$  is terminal
  
```

Figure 2.17: Pseudocode of Q-learning.

The process of solving this maze problem by using Q-learning is to train an agent to find the optimal path starting from grid (0,0) to (6,6), given no prior knowledge of the environment. To encourage the robot to find the shortest path, a small penalty of 0.04 units is applied each time the robot moves into an empty (white) cell, and obstacles are places around the maze (marked in gray) which result in a larger penalty of 0.75 units if the robot enters a cell containing one of them. The robot can only move up, down, left or right (that is, diagonal moves are not allowed). However, a level of uncertainty is associate with each movement, such that there is only an 80% chance the robot will move in the intended direction and a 20% chance the robot moves at right angles to the

intended direction (split evenly between the two possibilities). The robot is unable to move outside the boundaries of the maze, and if it attempts to do so, bumps into the wall and its position remains unchanged. If the robot successfully makes it to the end of the maze, it receives a reward of 1 unit. Assuming a discount rate of 0.9, a learning rate of 0.3 and an epsilon greedy exploration strategy with (constant) epsilon equal to 0.5, after 50,000 iterations of the Q-learning algorithm we get the following policy. The diagram shows the optimal direction for the robot to take in each square of the grid.

Thus, the key of solving the problem through Q-learning is to keep updating the value on Q-table and making decisions on some states for next movement according to the new value.

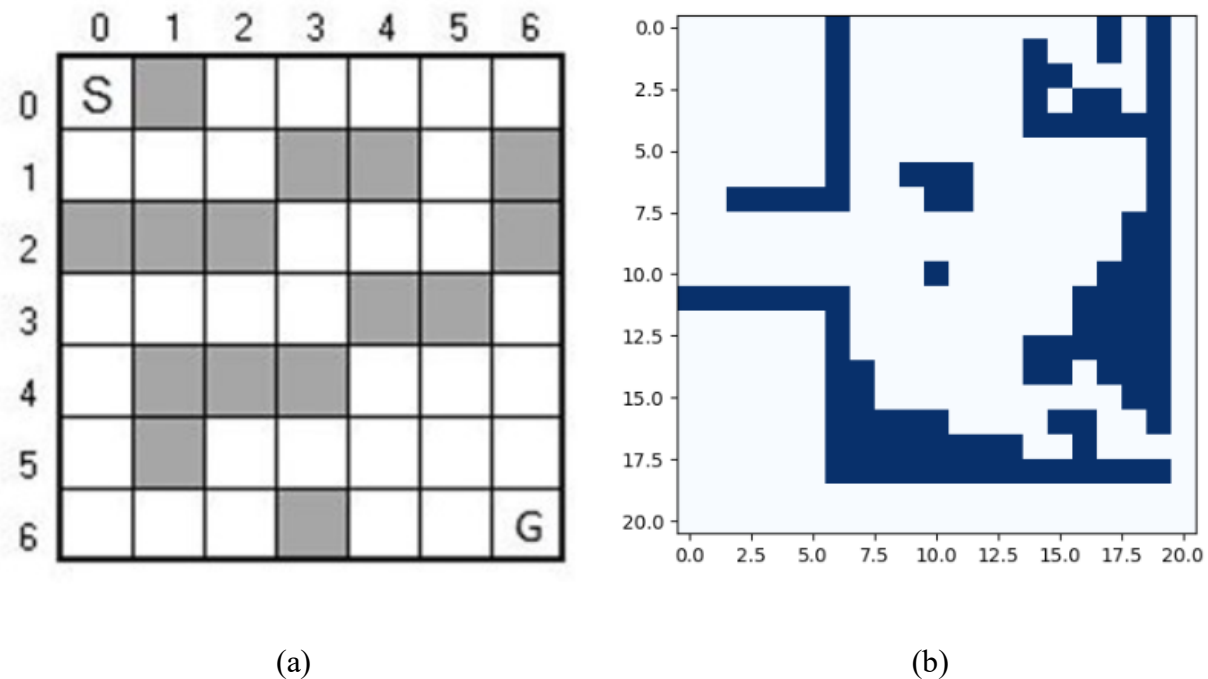


Figure 2.18: (a): One example of typical maze puzzle problem. (b): Local 2D grid map generated from our robot. This can be as similar as the maze puzzle problem. [37]

CHAPTER 3

APPROACH

3.1 Hardware Schematic of the Robot

The hardware setup and selection can be described in Figure 3.1 and Table 3.1. A Segway Ninebot S two-wheel self-balancing scooter was selected as the basic and we modified it except its own driving system. Instead, we installed a slope-driven pendulum mechanism to control the speed and a cross rod to control the steering. Once the robot was powered on, it can keep balancing by itself and we achieve control over the robot through two servos connected to the pendulum and cross rod. At the bottom of the robot there is another servo connected to a holder to keep it standing while power off. More detailed information about how we control the robot will be presented in CHAPTER 3.3.1 after the kinematic and dynamic model was introduced in CHAPTER 3.2.

Table 3.1: Hardware configuration of the proposed robot.

Sensor	Actuator	Other Hardware
LS 16 Channel LiDAR	DS3225 25kg Digital Servo x 3	Arduino Uno x 2
HIKVISION DS-2CD2455FWD-IW Network Camera		Microsoft Surface
WitMotion WT901C-485 9 Axis IMU (Gyroscope)		Magnets x n
S&C 103SR13A-1 Hall Effect Magnetic Sensor x 2		6105-T5 Aluminum T-Slotted Extrusion

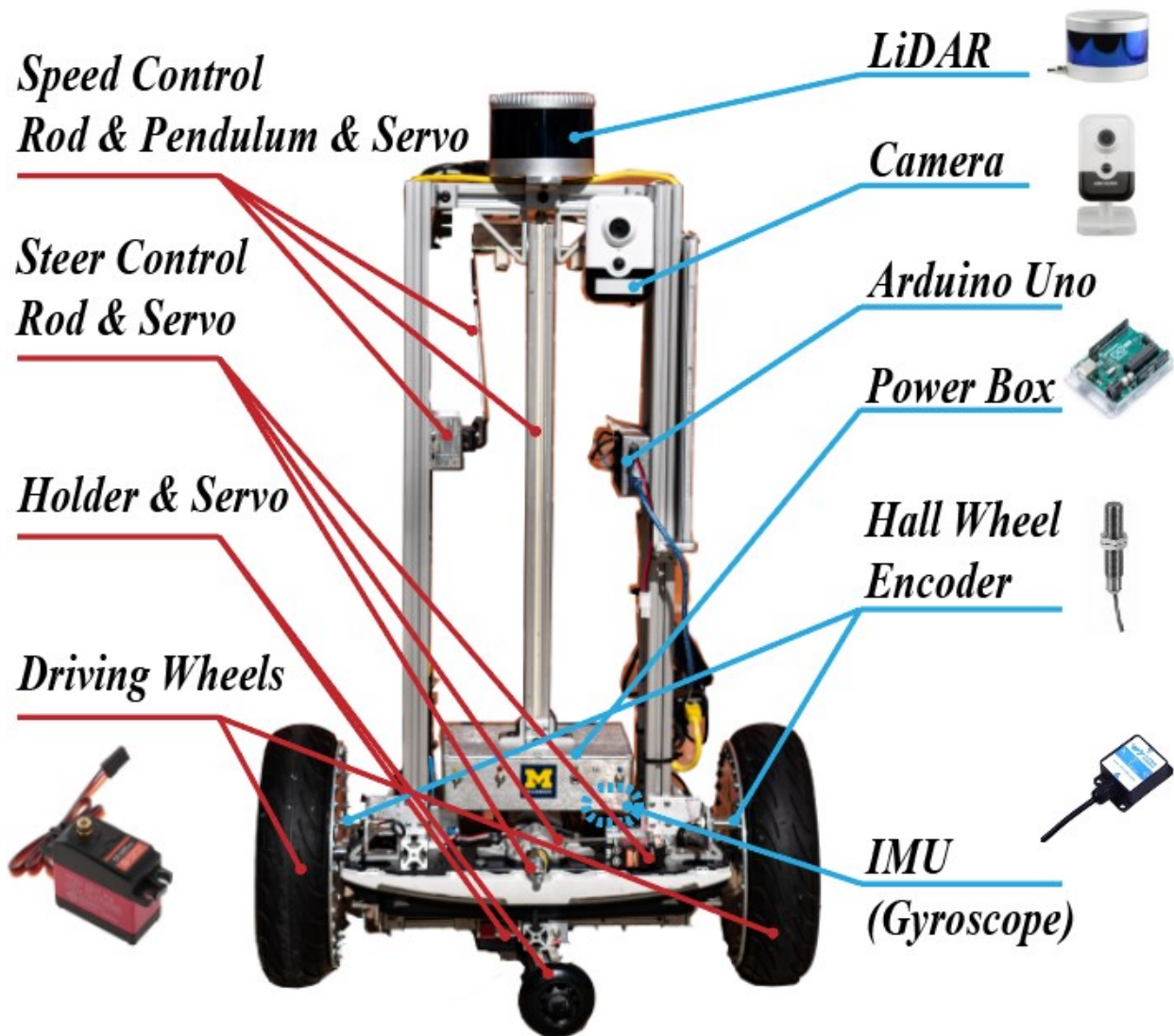


Figure 3.1: Hardware setup for the proposed robot.

The selection of sensors has also been shown in Table 3.1. A LS 16 Channel LiDAR and a HIKVISION DS-2CD2455FWD-IW network camera were selected as the main environment perception system, while a WitMotion WT901C-485 9 Axis IMU (Gyroscope) and two S&C 103SR13A-1 Hall Effect Magnetic Sensors formed the self-state perception system. For integrating those sensors and actuators together, a Microsoft Surface laptop and two Arduino Uno microcontrollers were selected. More detailed information about data format and acquisition will be introduced in CHAPTER 3.3.2.

3.2 Kinematic and Dynamic Model of the Robot

3.2.1 Two-Wheel Differential Kinematic Model

The chasis kinematic model of the Robot can be represented as a two-wheel differential model, [15] from where we can calculate the expected kinematics including pose (X, Y coordinate relative to global and azimuth) angular and linear velocity and so on from the input of sensors.

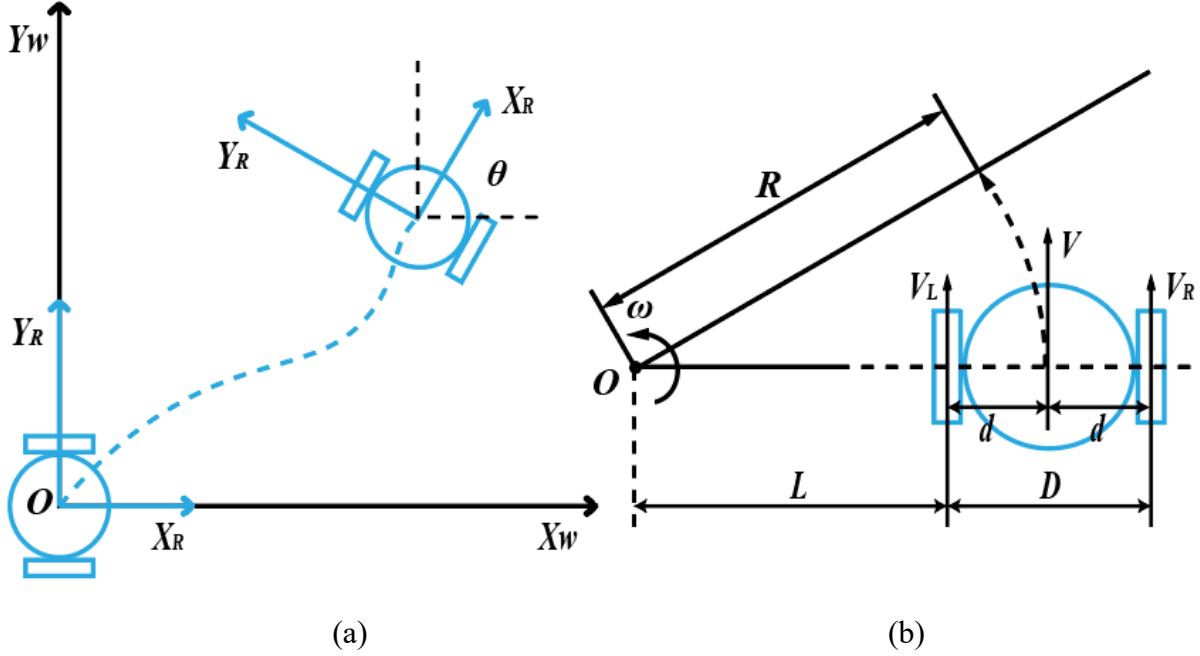


Figure 3.2: The kinematic model of the robot. (a) shows the global and robot coordinate of the robot, and (b) shows the decomposition of the robot's motion.

As shown in Figure 3.2 (b), the motion of the robot can be decomposed as a kind of circular motion. V and ω represent the linear and angular velocity of the whole robot, while V_L and V_R are the linear velocity of the robot's left and right wheel. d is half of the spacing between left and right wheel. If we set V and ω to known, then the velocity of left and right wheel can be determined as:

$$V_L = \omega \times (L+D) = \omega \times (L+2d) = \omega \times (R+d) = V + \omega d \quad (3.1)$$

$$V_R = \omega \times L = \omega \times (R - d) = V - \omega d \quad (3.2)$$

On the contrary, V and ω can be determined from wheel speed too:

$$V = \omega \times R = \omega \times (L + d) = \frac{2\omega L + 2\omega d}{2} = \frac{\omega L + \omega(L + 2d)}{2} = \frac{V_L + V_R}{2} \quad (3.3)$$

$$\omega = \frac{V_L - V_R}{2d} \quad (3.4)$$

Or we can use vectors to represent as:

$$\begin{pmatrix} V \\ \omega \end{pmatrix} = \begin{bmatrix} \frac{r_L}{2} & \frac{r_R}{2} \\ -\frac{r_L}{2d} & \frac{r_R}{2d} \end{bmatrix} \begin{pmatrix} \omega_L \\ \omega_R \end{pmatrix} \quad (3.5)$$

In (a) we use odometry model to calculate the position of the robot. Odometry model can integrate the position and azimuth of the robot relative to the global coordinate at any time. θ is the angle between any current X_R and X_w . There're actually two methods to calculate the position, the first one is to use wheel speed and integration, which has higher error:

$$X_t = X_{t-1} + \Delta x_w = X_{t-1} + \Delta d * \cos(\theta) = X_{t-1} + \Delta t * V * \cos(\theta) \quad (3.6)$$

$$Y_t = Y_{t-1} + \Delta y_w = Y_{t-1} + \Delta d * \sin(\theta) = Y_{t-1} + \Delta t * V * \sin(\theta) \quad (3.7)$$

Or it can be determined directly from the increments of the wheel encoder:

$$X_t = X_{t-1} + \Delta x_w = X_{t-1} + \Delta e * \frac{2\pi r}{S} * \cos(\theta) \quad (3.8)$$

$$Y_t = Y_{t-1} + \Delta y_w = Y_{t-1} + \Delta e * \frac{2\pi r}{S} * \sin(\theta) \quad (3.9)$$

Δe is the increment of wheel encoder pluses in a unit time Δt (Δt usually will be set as 10 or 20ms), S is the total number of pulses of the encoder when the wheel moves one revolution, and r is the radius of the wheel. θ can be read directly from the gyroscope's yaw. In that case, from the odometry model we can determine the pose and trajectory of the robot.

3.2.2 Dynamic Model

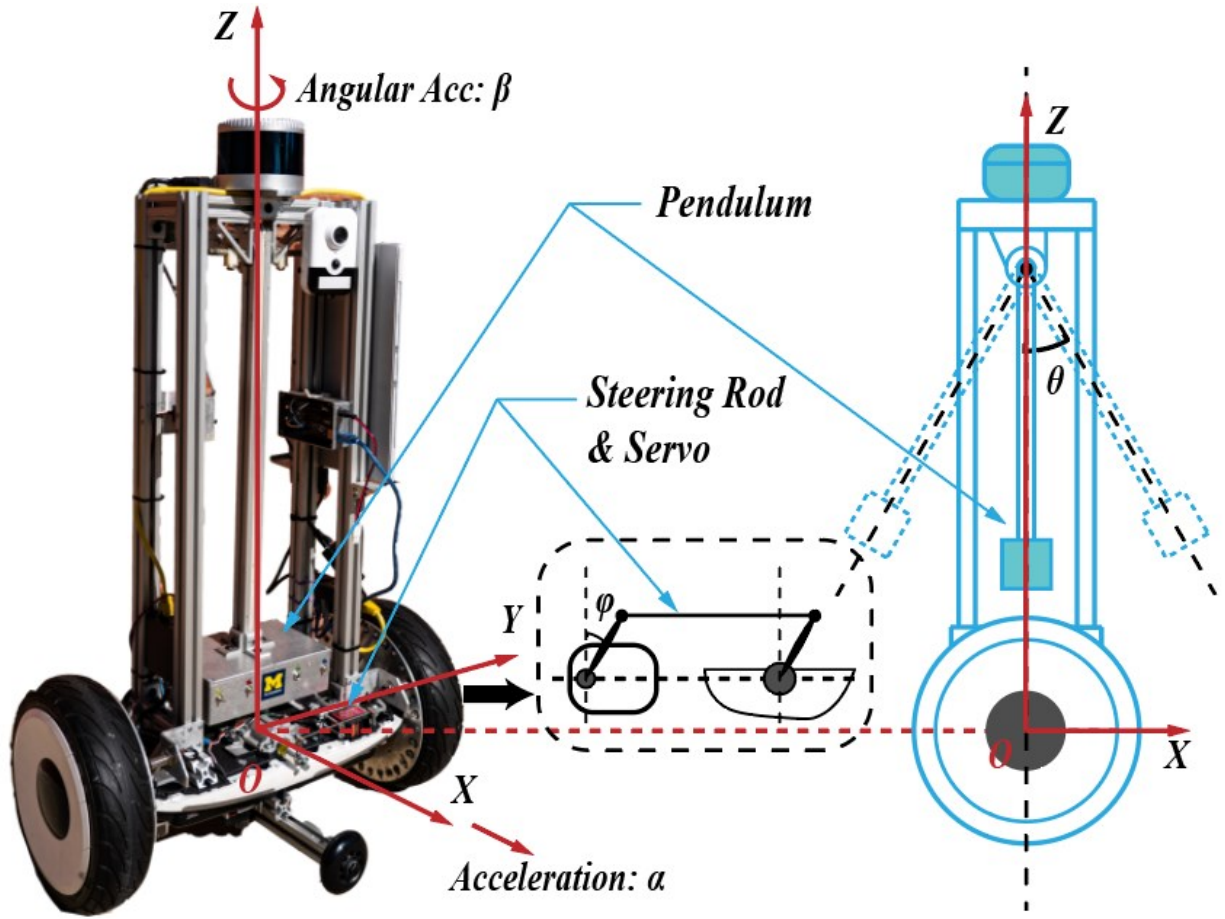


Figure 3.3: Dynamic model of the robot.

The dynamic model of the robot can be described as a slope-driven pendulum speed control mechanism, as shown in Figure 3.3. The velocity and orientation control of the robot were realized through two servos connected to a pendulum and a steering rod. The angle θ in plane XOZ and φ in plane YOZ determine the X direction linear acceleration α and Z direction angular acceleration β . The physical corresponding relationship between these two angles and accelerations can be formulated as:

$$\begin{cases} \alpha = k\theta^2 + p\varphi = 1.22\theta^2 + 0.17\varphi \\ \beta = q\varphi = 1.05\varphi \end{cases}, \alpha \text{ \& \ } \beta \text{ in rad} \quad (3.10)$$

The relationship and coefficients were determined through calibration. We change the pendulum and steering rod to different angles and read the accelerations α and β from the IMU (gyroscope) set on the plane of robot's bed.

With the dynamic and kinematic model of the robot, there is one thing we still need to output the desired velocity and heading precisely, which is the PID controller. The PID controller can provide a closed-loop control system with the velocity and heading feedback from the wheel encoder and IMU (gyroscope). A typical PID controller, which has been implemented to our robot's control system for velocity/orientation control can be described as shown in the flow chart Figure 3.4.

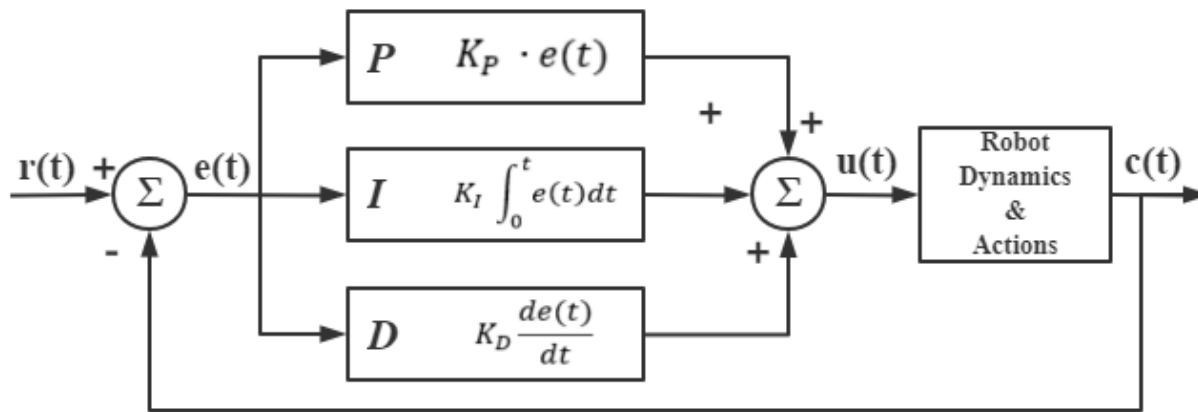


Figure 3.4: The PID controller configuration for the robot.

With the input of desired speed and orientation and feedback of current kinematic state from IMU and encoder, the more precise control of speed and steer can be generated and output to servos.

3.3 Controlling and Sensing

3.3.1 Controlling the Robot

After CHAPTER 3.2, we already know that we need to output the angle of the two servos to achieve control over the robot. Hence, in this part, the method of how we output the angle command to the servos will be presented.

The servo selected in this thesis is DS3225 25KG digital servo. Some useful specifications have been shown in Table 3.2.

Table 3.2: Specification of DS3225 25KG digital servo.

Operating Voltage	4.8 – 6.8 V
Operating Speed	0.15 sec/60 degree (5.0 V)
	0.13 sec/60 degree (6.8 V)
Stall Torque	21 kg/cm (5.0 V)
	25 kg/cm (6.8 V)
Working Frequency	50 – 333 Hz
Working Period	20000 – 3003 μ s
Motor Type	DC Motor

As we all know, the working principle of servo can be simply described in Figure 3.5. The angle of servo will vary according to the change of duty cycle, which is called pulse width modulation. In here we chose two servos with working frequency at 50Hz, working voltage at 6V.

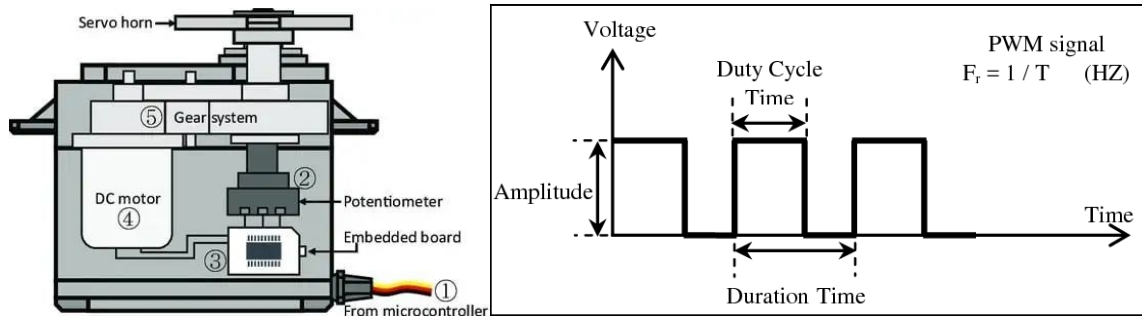


Figure 3.5: Digital servo structure and PWM working principle.

The servo proposed has three terminals including signal, power input and ground. Based on the specification and requirements, we chose an Arduino Uno microcontroller to control these two servos. The wire diagram can be simply described in Figure 3.6. Each servo has an individual power supply, and they share the same ground with the microcontroller. The PWM control of these servo can be simplified through build-in library from Arduino. The Arduino Uno is then connected to upper system, which is the main frame established through Python 3.6 on Surface.

The library PyFrimata enables the communication between Arduino and Surface. The PWM command was digitized to a number in two decimal places through this library and output to pin

#9 and #11. Thus, with the kinematic and dynamic model, we are able to realize the control of velocity and orientation of the robot.

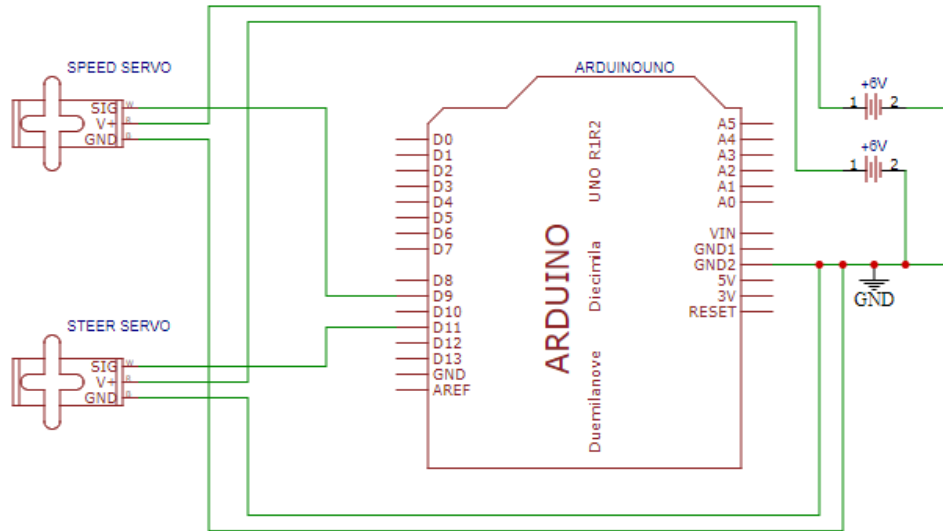


Figure 3.6: Wiring diagram for servos and Arduino.

3.3.2 Sensing

For LiDAR: The data acquired from LiDAR is called point cloud, which has been introduced in CHAPTER 2. In this thesis, the LiDAR selected is LS 16 Channel LiDAR, and the specification has been shown in Table 3.3

Table 3.3: Specification of LS 16 Channel LiDAR.

Laser Wavelength	905 nm
Maximum Range	50 – 70 m
Accuracy	± 3 cm
Angle of Field (FOV)	Vertical: $\pm 15^\circ$ Horizontal: 360°
Resolution of FOV	Vertical: 2° Horizontal: 0.09°
Scan Rate	10 Hz
Data acquisition Speed	320000 points/sec maximum

The connection between LiDAR and Surface is realized through ethernet and UDP protocol. The origin data pack received from LiDAR has three types including main data stream output

protocol (MOP), device information output protocol (DIFOP) and user configuration write protocol (UCWP). Each type of data pack has a length in 1248 bytes and in here we can parse the point cloud data from MOP. After decoding the received data pack, we can collect data in format of $\{Distance, Azimuth, Intensity, Laser ID, Timestamp\}$.

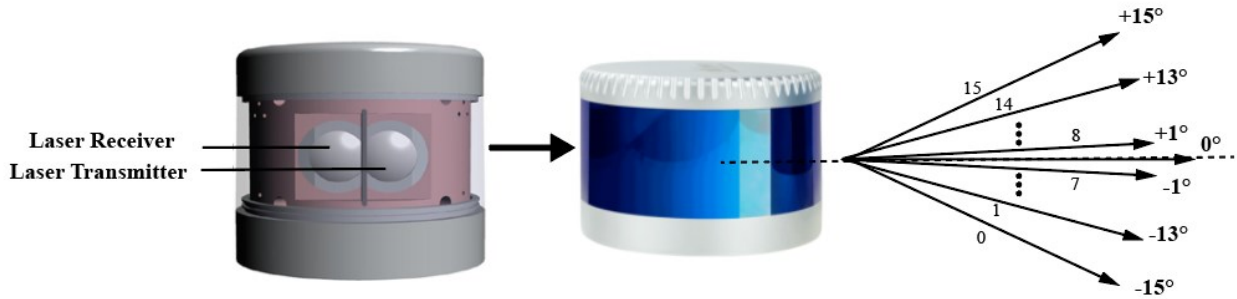


Figure 3.7: Vertical scan angle of LiDAR.

Table 3.4: Vertical angles corresponding to laser ID.

Laser ID	Vertical Angle
0	-15°
1	-13°
2	-11°
3	-9°
4	-7°
5	-5°
6	-3°
7	-1°
8	+1°
9	+3°
10	+5°
11	+7°
12	+9°
13	+11°
14	+13°
15	+15°

Since we are using a 16 channel LiDAR, the vertical angle of each received points can be determined through Figure 3.7 and Table 3.4. Then we can calculate the 3D information for each point and generate point cloud data in format of

$$\{X, Y, Z, \theta, \varphi, Intensity, Timestamp\}$$

Where θ and φ is the vertical angle and azimuth. Note that we can receive 38400 points each scan in scan frequency of 10Hz. As the major environment perception mode, the collected point cloud data can be used for obstacle avoidance, data fusion and SLAM. A typical point cloud scan frame captured from LiDAR can be visualized in Figure 3.8. The points were colored by order of intensity.

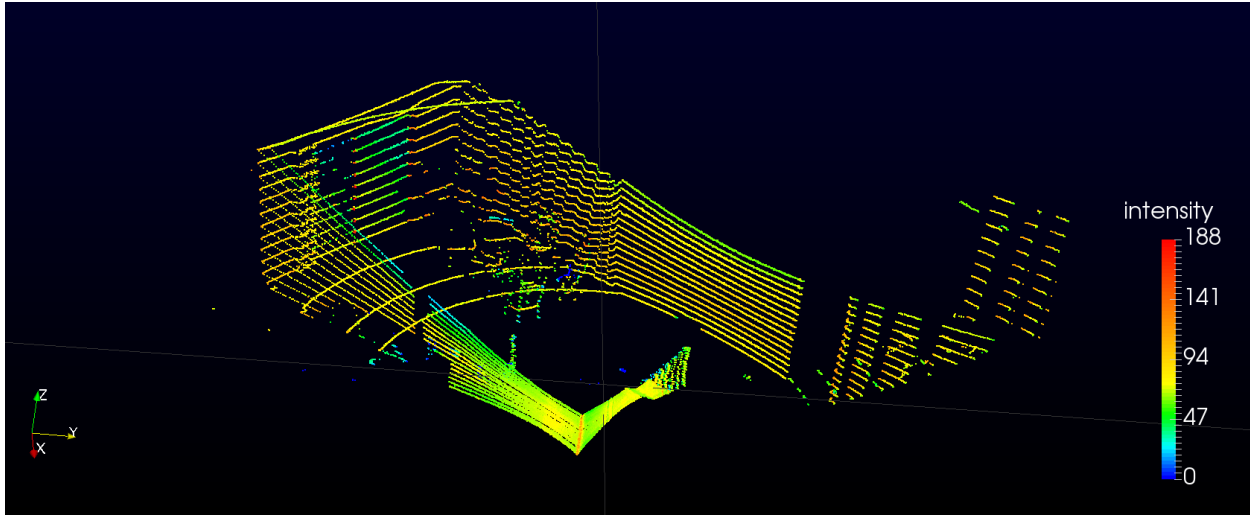


Figure 3.8: Visualization of captured point cloud data.

For Camera: This part is pretty simple since the acquisition of images doesn't need too much steps. The camera chose in here, HIKVISION DS-2CD2455FWD-IW is a monocular network camera, and the useful specification has been shown in Table 3.5.

Table 3.5: Specification of HIKVISION DS-2CD2455FWD-IW camera.

Focal Length	2.8 mm
FOV	Horizontal: 97°
	Vertical: 54°
	Diagonal: 113°
Maximum Resolution	2944×1656
Power Supply	12 V DC, 0.47 A

The camera was also connected to Surface through ethernet as the same as LiDAR, and the open-source library OpenCV from Python has provided build-in functions to capture images. In this thesis, the vision information is mainly used for data fusion unlike other vision-driven robots.

Hence, the requirement about resolution of images is not so strict which has been chose as 1920×1080 . The image can be captured simultaneously with point cloud so that we can perform data fusion later in CHAPTER 4.

For Wheel Encoder: The wheel encoder implemented on this robot was built on our own. The hardware structure and measurement principle can be described in Figure 3.9.

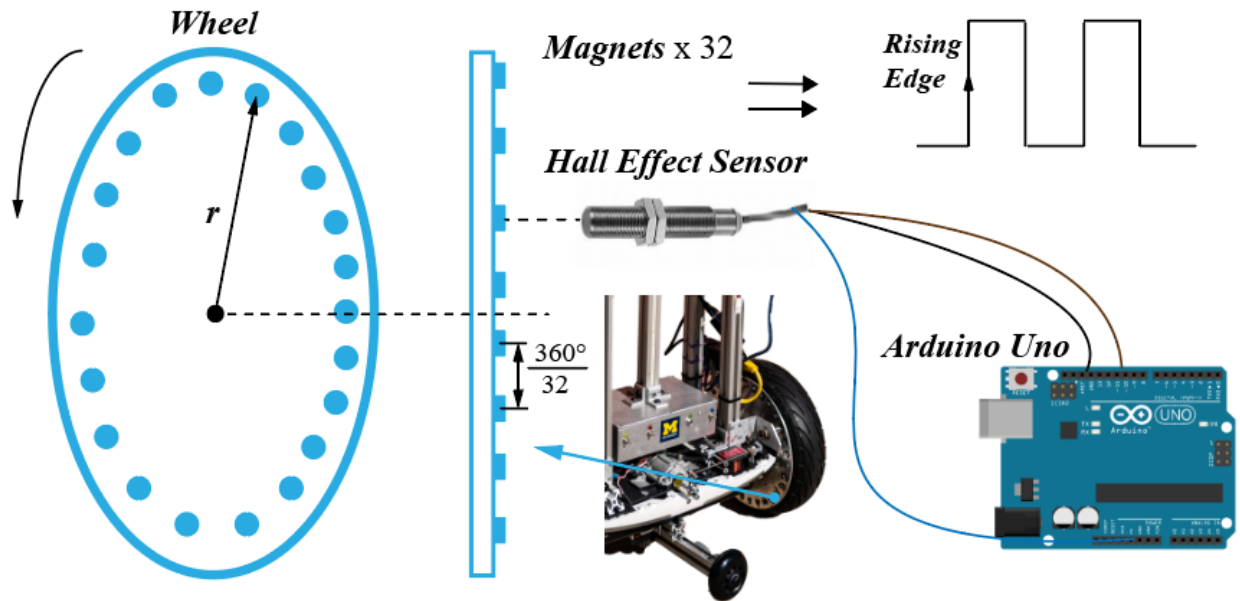


Figure 3.9: Configuration of wheel encoder.

The selected hall effect sensor will generate a pulse when there is a magnet passing by in front of the detecting element of it. We crafted two aluminum plates with 32 magnets evenly and radiatively located on each plate according to the center of the wheel. The plate will rotate as the same speed as the wheel, and this simple system became our wheel encoder. Thus, the key of measuring the wheel speed from this encoder is detecting the rate of rising edge from hall effect sensor. For detecting the rising edge, we connect the hall effect sensor to another Arduino Uno microcontroller and use the digital interrupt pin, which will add the number of pulses detected to the counter automatically. The interrupt interval was set to 0.5s.

Then with some parameters of the wheel we can calculate the wheel speed and odometry through the following eq.:

$$\begin{cases} \omega_L = 60 \cdot \frac{2n_L}{32} \cdot \frac{2\pi}{60} \text{ rad/s} \\ \omega_R = 60 \cdot \frac{2n_R}{32} \cdot \frac{2\pi}{60} \text{ rad/s} \end{cases}, \quad \begin{cases} v_L = \omega_L r \\ v_R = \omega_R r \end{cases} \quad (3.11)$$

$$\begin{cases} O_L = \frac{2\pi r}{32} \cdot n_L \\ O_R = \frac{2\pi r}{32} \cdot n_R \end{cases} \quad (3.12)$$

Where n_L and n_R is the pulses received at each interrupt interval for left and right wheel, r is the radius of the wheel, O_L and O_R is the odometer for left and right wheel. Thus, the wheel rotational and linear speed can be read from this encoder directly. After a simple encapsulation, the information read from encode will be sent to the Surface in the form as:

$$\{SLS\omega_LRS\omega_RLOO_LROO_RE\}$$

Where S, E means starting and ending, LS, RS, LO, RO means rotate speed and odometer from left and right wheel.

Then we simply calibrate the odometer by setting a fixed route as the ground truth, reading the left and right odometer, and taking the mean of multiple tests. The accuracy performs well when using a straight route as it can reach about 98.72%. However, when we try to use the odometer increment to localize the pose of the robot, the accuracy did not achieve the expectation. After many attempts, the reason was found that the rotate speed difference between left and right wheel caused by differential driven mode of Segway while turning must results in an odometer difference. In addition, when the robot is turning around in place, the encoder will still add pulses while the position of the robot is actually not changing. For this kind of situation, a correction factor was added as shown in the eq.:

$$O = |O_L - O_R|/2 \cdot k_o, \quad k_o = \begin{cases} 0.95, & \frac{|v_L - v_R|}{v_L + v_R} \in (0, 0.4) \\ 0.5, & \frac{|v_L - v_R|}{v_L + v_R} \in [0.4, 0.75) \\ 0.2, & \frac{|v_L - v_R|}{v_L + v_R} \geq 0.75 \end{cases} \quad (3.13)$$

The classification of correction factor k_o represents the speed difference between left and right wheel, and it was determined by calibration through many tests. Thus, new odometer can be used to positioning the robot together with the IMU.

For IMU: This part is pretty simple as for camera. The IMU used in here is WitMotion WT901C-485, a 9-axis inertial measurement unit, or called gyroscope. The measurement principle of IMU is not under discussion of this thesis, so in here we just briefly introduce the implementation and how we combine the information from encoder together to determine the pose of the robot. The roll, pitch and yaw axis of IMU can be described in Figure 3.10.

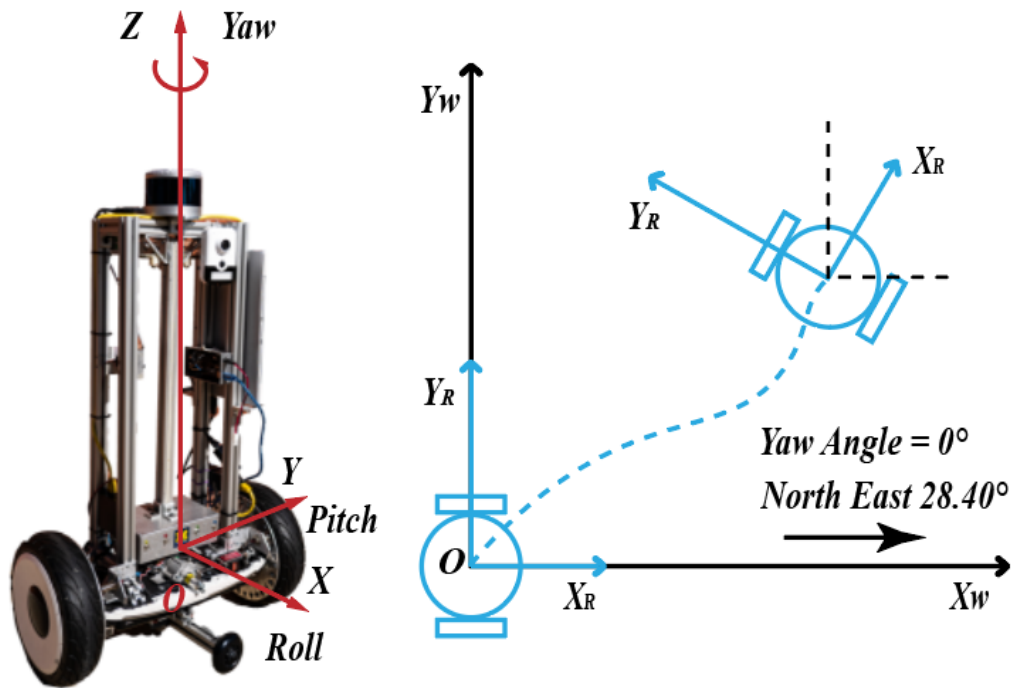


Figure 3.10: Roll, pitch, and yaw of IMU on the robot.

The IMU is connected to the Surface through USB, and we can read the linear acceleration, angle, and angular speed for each axis individually. Note that the yaw angle read from IMU has an absolute zero which corresponds to 28.40° , northeast. Then with the kinematic model mentioned before and odometer read from encoder simultaneously, we can determine the position of the robot through eq:

$$\begin{cases} X_t = X_{t-1} + \frac{\Delta O}{2} \cdot k_o \cdot \cos \theta \\ Y_t = Y_{t-1} + \frac{\Delta O}{2} \cdot k_o \cdot \sin \theta \end{cases} \quad (3.14)$$

Where $\Delta O = |O_L - O_R|_t - |O_L - O_R|_{t-1}$ is the increment of odometer. Also, the linear acceleration for three axes can be used in obstacle avoidance, which will be introduced in detail in CHAPTER 3.5.1.

3.4 Software System Framework

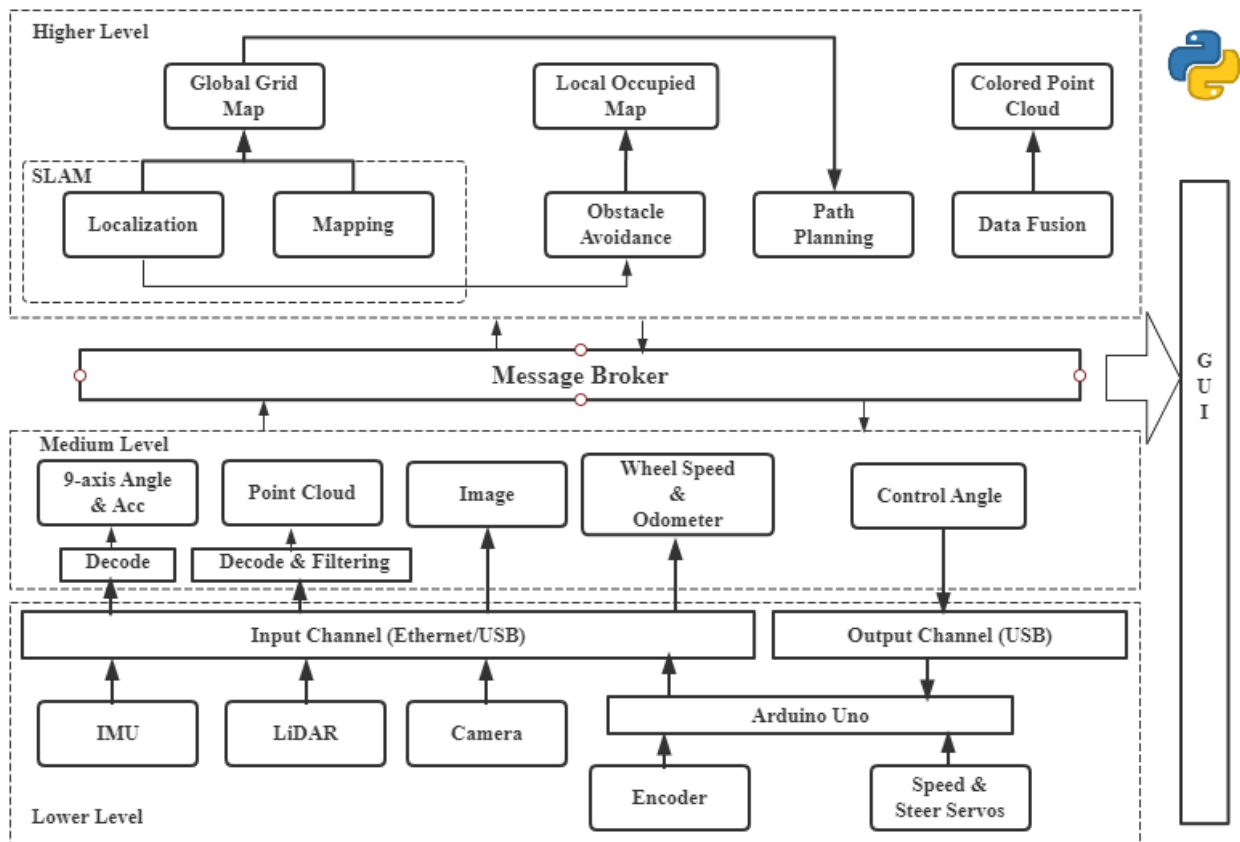


Figure 3.11: Software system framework.

Instead of using robot operating system (ROS) as most of robots do, we build the whole system based on Python 3.6. The software system structure can be described as the flow chart in Figure 3.11.

As shown in Figure 3.11, there are three levels including higher, medium, and lower level, which are connected by I/O channels and a message broker. The lower level concludes the I/O channel, sensors, and actuators. The mission of this level is perception and action. The collected data from sensors will be streamed to medium level for decoding and pre-processing, while the action commands came from upper levels will be executed. After receiving data from lower level, the medium level will decode and pre-process the data such as filtering, transformation e.g., and publish the processed data to the message broker. Also, the command and decision sent from higher level will be transformed into PWM which can be executed directly by servos. In here, users can read the data straightly from the message broker. The higher level concludes the proposed four main functions, Obstacle avoidance, SLAM, path planning and data fusion. The main objective of this level is to realize these functions by subscribing the data published on message broker. This is convenient because one kind of data can be useful for different functions. For example, the roll-pitch-yaw angle read from IMU can be used both for SLAM and obstacle avoidance simultaneously. In addition, the message broker can be essential for ensuring the synchronism of collected data. Finally, some results and information such as global grid map, local occupied map, and colored point cloud will be generated by higher level and pass to message broker to be presented to users. Meanwhile, the robot will act autonomously such as self-exploring the environment.

3.5 Function Realization

3.5.1 Obstacle Avoidance (2D-LiDAR Occupied Grid Mapping)

The flow chart of obstacle avoidance function has been shown in Figure 3.12. The basic logic of this function can be described as random self-exploring with obstacle avoidance, while the action principle can be described as a 2D-LiDAR occupied grid mapping and kinematic driven fuzzy logic algorithm.

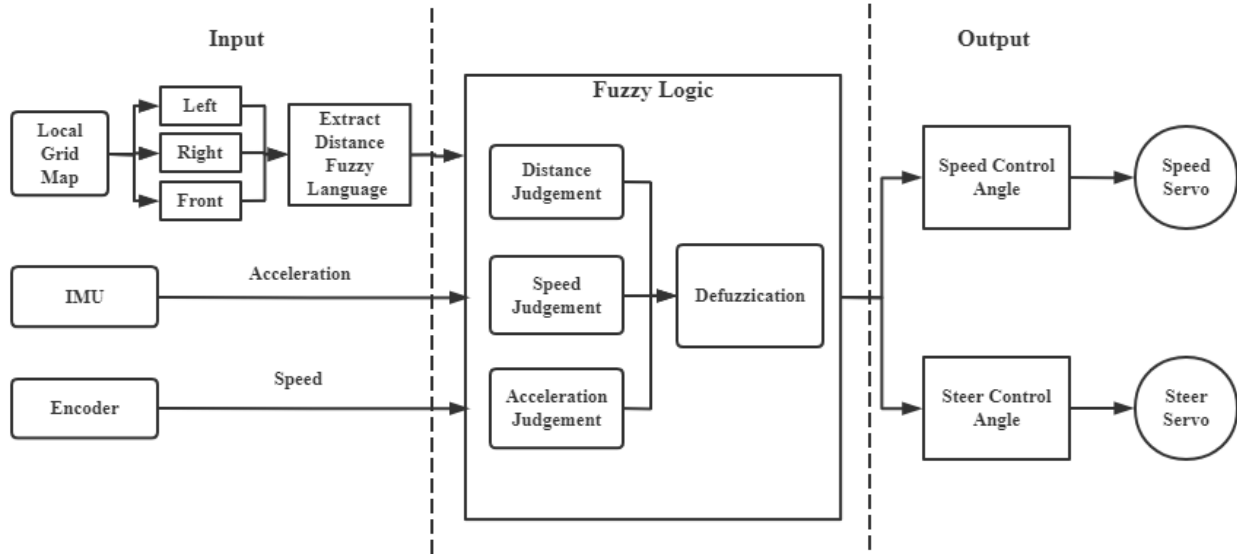


Figure 3.12: Flow chart for obstacle avoidance.

As shown in Figure 3.12, the input of this function includes a 2D grid map generated by LiDAR, wheel speed read from encoder, angles and acceleration read from IMU. After a fuzzy logic judgement mode, the desired control command including speed and steer will be generated and pass to the two servos corresponded. The implication of fuzzy language variables has been defined as:

- Distance: $\{Far, Close\}$
- Current Speed: $\{Fast, Slow\}$
- Steering: $\{Left, Front, Right\}$
- Acceleration: $\{BN, SN, Z, SP, BP\}$, $\{Big\ Negative, Small\ Negative, Zero, Small\ Positive, Big\ Positive\}$

In there, the definition of left, right, and front distance has been shown in Figure 3.13. The grid size of local map has been set to $0.3m$, which is close to the size of the robot, and the map size has been set to $3 \times 3m$. The center of the local grid map is the current position of the robot, while the current heading of robot was set to be the same as the north in global. With the implication of fuzzy language variables, we defined 50 rules as the reference for guiding the action of robot, as shown in Table 3.6. Thus, the next step is to generate the 2D grid map as the perception, and define the control amount of the robot.

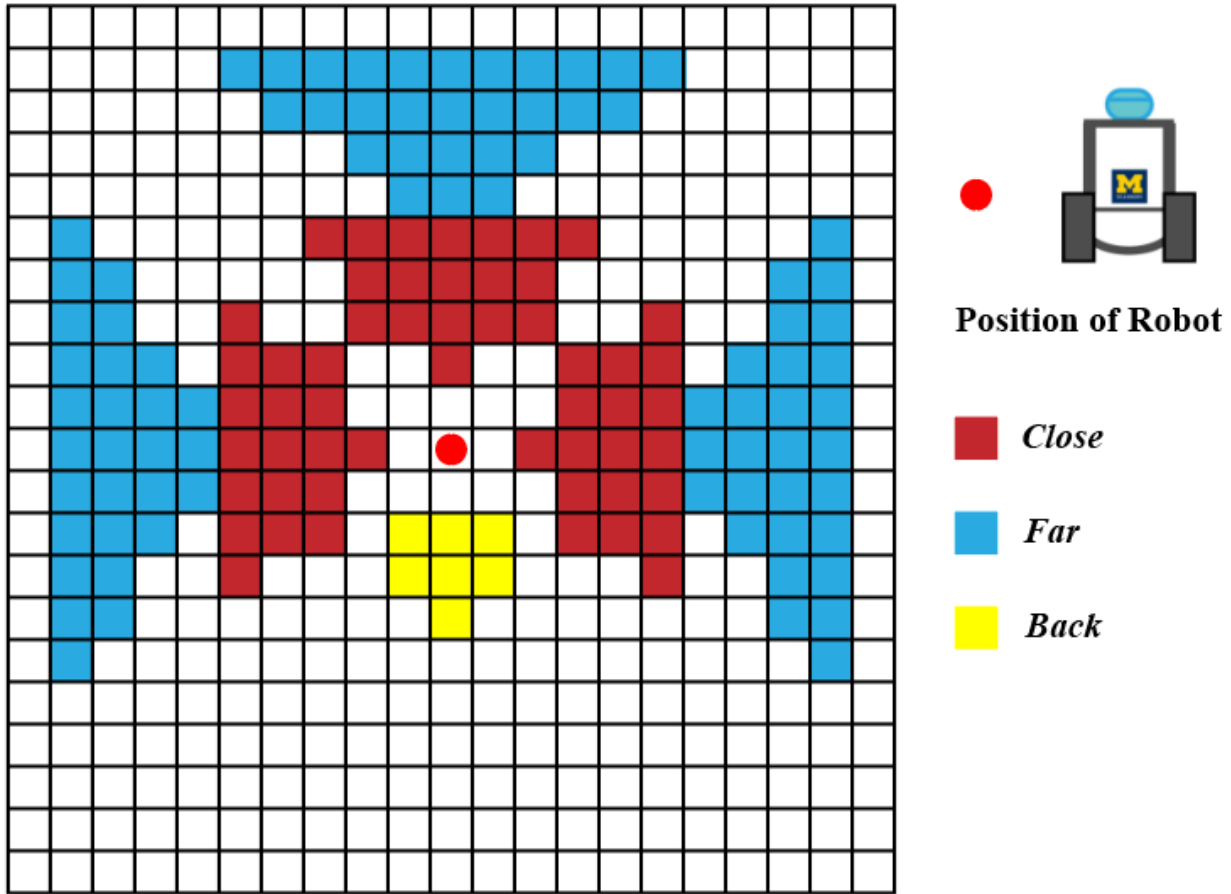


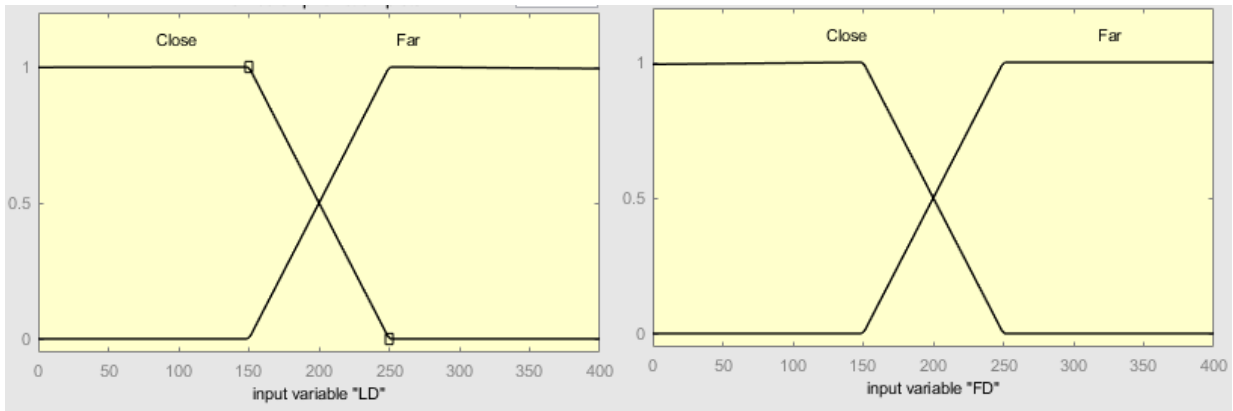
Figure 3.13: LD, FD, RD on grid map.

Note that, to generate the 2D local grid map, there is still another data pre-processing step implemented in order to reduce computation load and increase running speed. As mentioned before, the format of point cloud data received from LiDAR is generalized as:

$$\{X, Y, Z, \theta, \varphi, Intensity, Timestamp\}$$

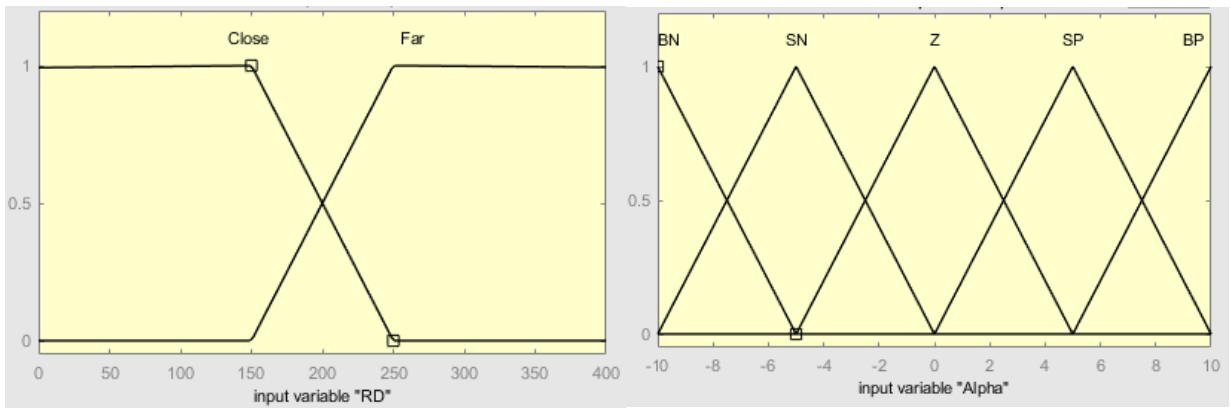
But in this function, the only useful information is the X, Y coordinates. Besides, the construction of local map will only focus on points within $3 \times 3m$. Thus, we first extract the X, Y coordinate and delete all points over $3 \times 3m$. Then, for each grid on the map, we count the number of points within the coordinate and set a confidence threshold to eliminate outliers.

With the local grid map, wheel speed, and acceleration, we are able to determine the fuzzy membership of input variables. In here we chose continuous and triangle-shaped domain function as the membership function of each input variable, as shown in Figure 3.14 and Figure 3.15.



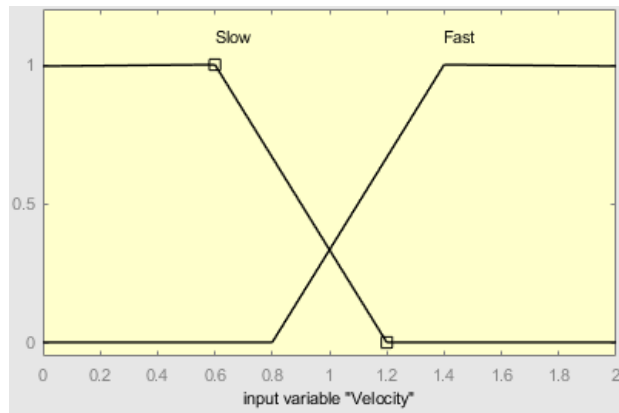
(a)

(b)



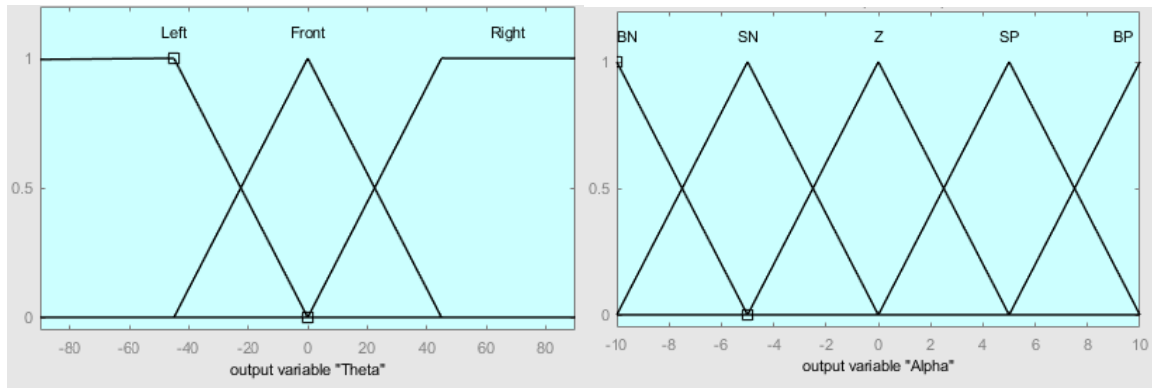
(c)

(d)



(e)

Figure 3.14: Fuzzy membership functions of inputs. (a): LD. (b): FD. (c): RD. (d): α . (e): v .



(a)

(b)

Figure 3.15: Fuzzy membership functions of output. (a): θ . (b): α .

Table 3.6: Fuzzy logic rule sets.

Rule Number	Input					Output	
	LD	FD	RD	a	v	a	θ
1	Far	Far	Far	BN	Fast	SP	Front
2	Far	Far	Far	BN	Slow	BP	Front
3	Far	Far	Far	BP	Fast	SN	Front
4	Far	Far	Far	BP	Slow	Z	Front
5	Far	Far	Far	Z	Fast	Z	Front
6	Far	Far	Far	Z	Slow	SP	Front
7	Far	Far	Far	SP	Fast	SN	Front
8	Far	Far	Far	SP	Slow	Z	Front
9	Far	Far	Far	SN	Fast	Z	Front
10	Far	Far	Far	SN	Slow	Z	Front
11	Close	Close	Close	BN	Fast	BN	Front
12	Close	Close	Close	BN	Slow	SN	Front
13	Close	Close	Close	BP	Fast	BN	Front
14	Close	Close	Close	BP	Slow	BN	Front
15	Close	Close	Close	Z	Fast	BN	Front
16	Close	Close	Close	Z	Slow	SN	Front
17	Close	Close	Close	SP	Fast	BN	Front
18	Close	Close	Close	SP	Slow	BN	Front
19	Close	Close	Close	SN	Fast	BN	Front
20	Close	Close	Close	SN	Slow	SN	Front
21	Close	Far	Far	BN	Fast	SP	Front
22	Close	Far	Far	BN	Slow	BP	Front
23	Close	Far	Far	BP	Fast	SN	Front
24	Close	Far	Far	BP	Slow	Z	Front
25	Close	Far	Far	Z	Fast	Z	Front
26	Close	Far	Far	Z	Slow	SP	Front

27	Close	Far	Far	SP	Fast	SN	Front
28	Close	Far	Far	SP	Slow	Z	Front
29	Close	Far	Far	SN	Fast	Z	Front
30	Close	Far	Far	SN	Slow	Z	Front
31	Far	Close	Far	BN	Fast	BN	Right
32	Far	Close	Far	BN	Slow	SN	Right
33	Far	Close	Far	BP	Fast	BN	Right
34	Far	Close	Far	BP	Slow	BN	Right
35	Far	Close	Far	Z	Fast	BN	Right
36	Far	Close	Far	Z	Slow	SN	Right
37	Far	Close	Far	SP	Fast	BN	Right
38	Far	Close	Far	SP	Slow	BN	Right
39	Far	Close	Far	SN	Fast	BN	Right
40	Far	Close	Far	SN	Slow	SN	Right
41	Far	Far	Close	BN	Fast	SP	Left
42	Far	Far	Close	BN	Slow	BP	Left
43	Far	Far	Close	BP	Fast	SN	Left
44	Far	Far	Close	BP	Slow	Z	Left
45	Far	Far	Close	Z	Fast	Z	Left
46	Far	Far	Close	Z	Slow	SP	Left
47	Far	Far	Close	SP	Fast	SN	Left
48	Far	Far	Close	SP	Slow	Z	Left
49	Far	Far	Close	SN	Fast	Z	Left
50	Far	Far	Close	SN	Slow	Z	Left

3.5.2 Sensor Data Fusion

This part can be separated into two main steps, starting from LiDAR and camera, as shown in Figure 3.16. The first one is to co-calibrate the LiDAR and camera to obtain intrinsic and extrinsic matrices for both. With the extrinsic matrix, the geometric transformations (rotation R and translation T) can be solved to correlate the point cloud and image frame together in a same coordinate.

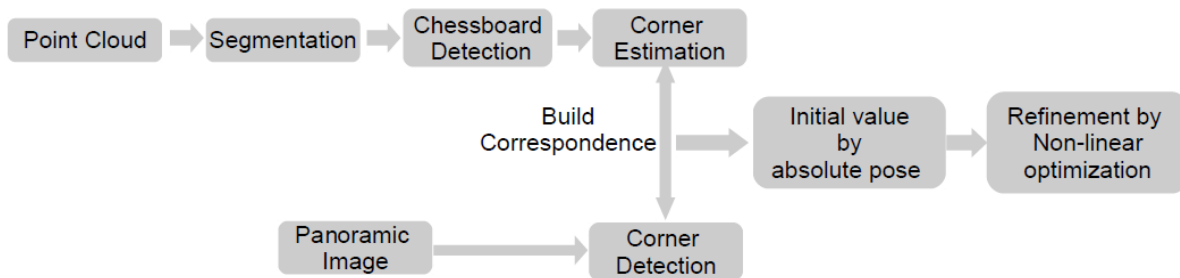


Figure 3.16: Flowchart for data fusion process. [31]

For calibrating the camera, the traditional method is used, which enquires a checkerboard in size of 6×9 , and each checker is $10 \times 10 \text{ mm}$, as shown in Figure 3.17. The resolution of images captured for calibration is 1920×1080 and one image set contains 20 images captured while the checkerboard was at different angles and positions.

The traditional camera calibration method presented in this thesis can be summarized as a process of establishing the relationship between the real world and the pixel coordinate which can be quantified and programmed. In Figure 3.18, all the three coordinates participated in this

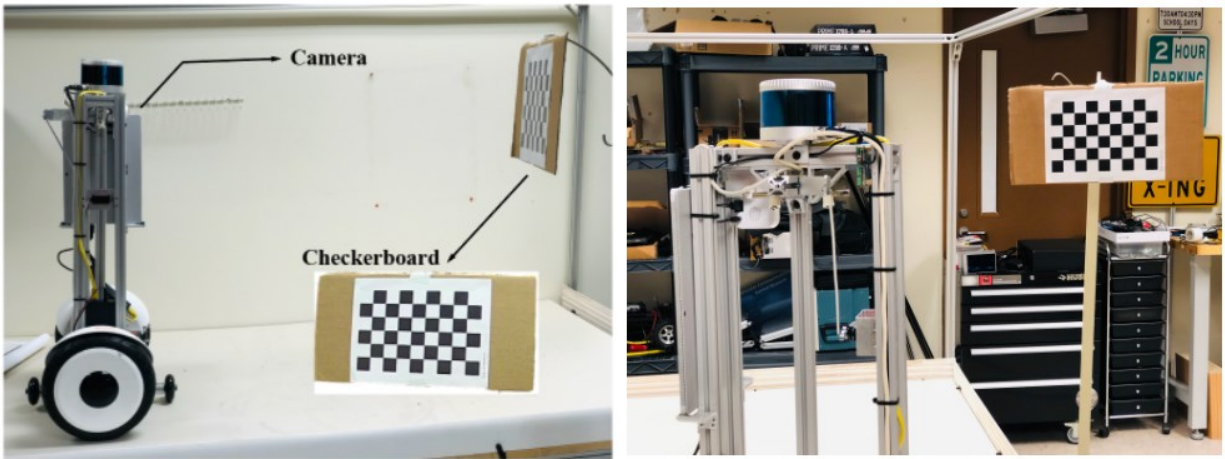


Figure 3.17: One example scene of collecting data by the robot.

process were identified in different colors. The red one is the camera coordinate, in here we use $O_c - X_c Y_c Z_c$ to represent. The green one is the image coordinate (or pixel coordinate) $o - xy$. The yellow one is the real-world coordinate $O_w - X_w Y_w Z_w$. The relationship between these coordinates are some translation and rotation transformations and a physical principle called pinhole imaging principle, as shown in Figure 3.18. Overall, the transformation from image coordinate to world coordinate can be represented by two matrices, which are the alleged intrinsic and extrinsic matrices.

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & u_0 \\ \frac{1}{dx} & 0 & v_0 \\ 0 & \frac{1}{dy} & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R \\ \bar{0} \\ 1 \end{bmatrix} T \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R \\ \bar{0} \\ 1 \end{bmatrix} T \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (3.15)$$

Where f is the focal length of the camera, Z_c is the scale factor, u_0 and v_0 are the principle point. The matrix $\begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ is called the intrinsic matrix and $\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$ is called the extrinsic matrix.

Now by extracting the corner of checkers in the image set captured from our robot, we can obtain the coordinate of the same corner from different images. From this process the intrinsic matrix can be determined, which will be presented as the calibration result in CHAPTER 4.

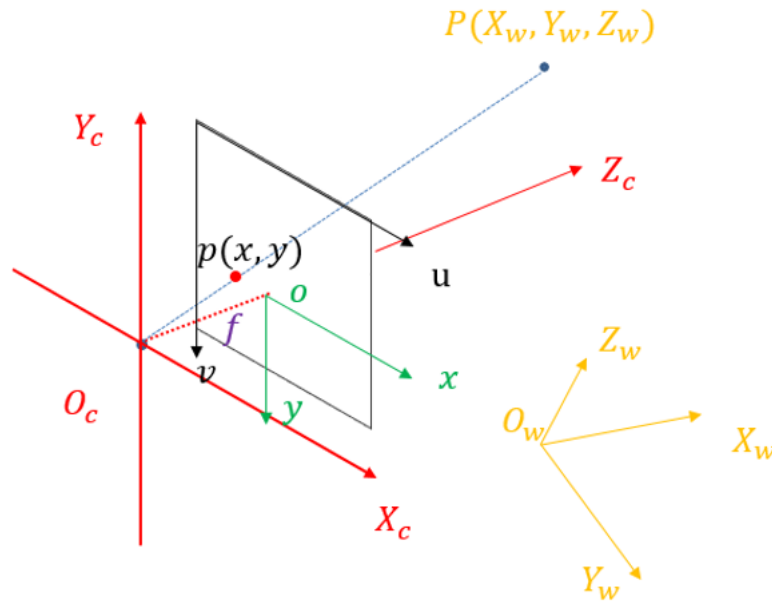


Figure 3.18: Camera, image, and real-world coordinates.

As mentioned before, the key about fusing data from camera and LiDAR is to establish the translation and rotation relationship between them accurately. With the calibrated image, the next step is to place each coordinate from the sensors itself together in a same coordinate. Given the LiDAR point cloud coordinate $P_L = (X_L \ Y_L \ Z_L)$ and the camera coordinate $P_c = (X_c \ Y_c \ Z_c)$, the geometric transformation can be determined as

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = R \begin{bmatrix} X_L \\ Y_L \\ Z_L \end{bmatrix} + T \quad (3.16)$$

where the R and T are the same as the rotation and translation matrices. The translation matrix $T = [t_x, t_y, t_z]^T$ is a 3×1 column vector, and rotation matrix R can be determined with three rotation angles $\{\theta_x, \theta_y, \theta_z\}$ correlated to the coordinate axes:

$$R = R_z(\theta_z)R_y(\theta_y)R_x(\theta_x) \quad (3.17)$$

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

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

$$R_z(\theta_z) = \begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then the point in real world that has being three-dimensional $P_c = (X_c \ Y_c \ Z_c)$ can be back projected onto the image plane in coordinate $p = (u, v)$. From the pinhole imaging principle that has been mentioned before, the projection equation in homogeneous coordinate can be formulated as:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = KP_c = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (3.18)$$

Where s is the scale factor, (f_x, f_y) and (u_0, v_0) are the same as the focal lengths and principal point.

To obtain better result of data fusion, the radial distortion caused by lens aberration should also be considered. Similar to [32], we use two distortion parameters k_1 and k_2 to characterize the radial distortions. Then the distortion corrected projection can be formulated as:

$$\tilde{u} = u + (u - u_0)[k_1(X_c^2 + Y_c^2) + k_2(X_c^2 + Y_c^2)^2] \quad (3.19)$$

$$\tilde{v} = v + (v - v_0)[k_1(X_c^2 + Y_c^2) + k_2(X_c^2 + Y_c^2)^2] \quad (3.20)$$

where $\tilde{p} = (\tilde{u}, \tilde{v})$ is a distorted point and $p = (u, v)$ a pixel on a un-distorted image.

Now since the projection equation of placing 3D point cloud onto the image plane has been deduced through, the next step is to estimate the extrinsic parameters $\{\theta_x, \theta_y, \theta_z, t_x, t_y, t_z\}$ and distortion parameters $\{k_1, k_2\}$. We still use the $6 \times 9 \times 10$ checkerboard as the landmark, and the corner of each checker will be the target point for projection. The corners in 3D point cloud P_L will be projected onto the 2D image points \tilde{p} to calculate the absolute difference between these and the real corners p^* in image. Then the estimation of extrinsic and distortion parameters can be derived by minimizing the cost function as followed:

$$C = \sum_{i=1}^n |p_i^* - \tilde{p}_i| \quad (3.21)$$

where i is the point index and n is the total number of points.

For extracting the corners from the 3D point cloud, there are two keys insisted in this thesis. The first one is to use the geometrical features to find the dimension of the checkerboard in 3D point cloud. The second one is using the different LiDAR reflection of the white and black blocks on the checkerboard.

At last, with the estimated intrinsic and extrinsic matrices, the 3D-2D correspondences between the 3D point cloud and the 2D image for data fusion can be determined.

3.5.3 2D SLAM

As mentioned before, the localization and mapping are one of the most essential part of an autonomous robot. In here the methodology and process of how we implement 2D SLAM function to our robot will be introduced in detail.

Based on the uncertainty of movement control and observation, the SLAM problem can be described as a kind of Markov Decision Process (MDP), more specifically as a Partially Observable Markov Decision Process (POMDP), as shown in Figure 3.19. In here, the circles represent:

- x_t : the actual pose of robot.
- u_t : the movement command sent to robot.

- z_t : the observation of environment from sensor.
- m : the actual map or description of real world or environment.

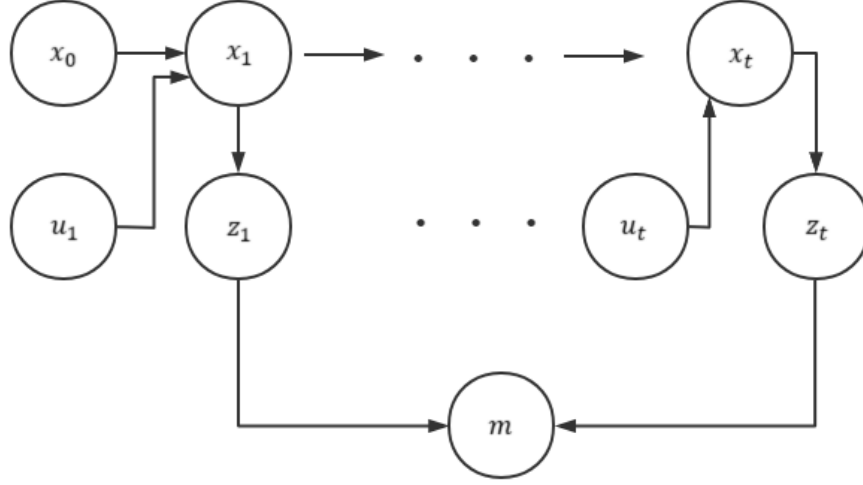


Figure 3.19: POMDP representation of 2D SLAM.

The theoretical structure chart of 2D SLAM has been shown in Figure 3.20. Since the SLAM algorithm proposed in this thesis is based on particle filter, the whole SLAM process can be described as a probabilistic distribution problem about solving the joint probability density of the probability of robot's current position. As illustrated in eq.:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) \quad (3.22)$$

Where $x_{1:t}$ is the trajectory of robot, m is the global map, $z_{1:t}$ is the observation from sensors (in here we use LiDAR 2D point cloud, wheel encoder odometer and IMU angles), $u_{1:t-1}$ is the movement control command.

Based on Rao-Blackwellized Particle Filter specifically, we split the SLAM into localization and mapping these two processes, so that the joint probability density can be factorized into () through Joint probability formula as:

$$p(x_{1:t}, m \mid z_{1:t}, u_{1:t-1}) = p(m \mid x_{1:t}, u_{1:t-1}) \cdot p(x_{1:t} \mid z_{1:t}, u_{1:t-1}) \quad (3.23)$$

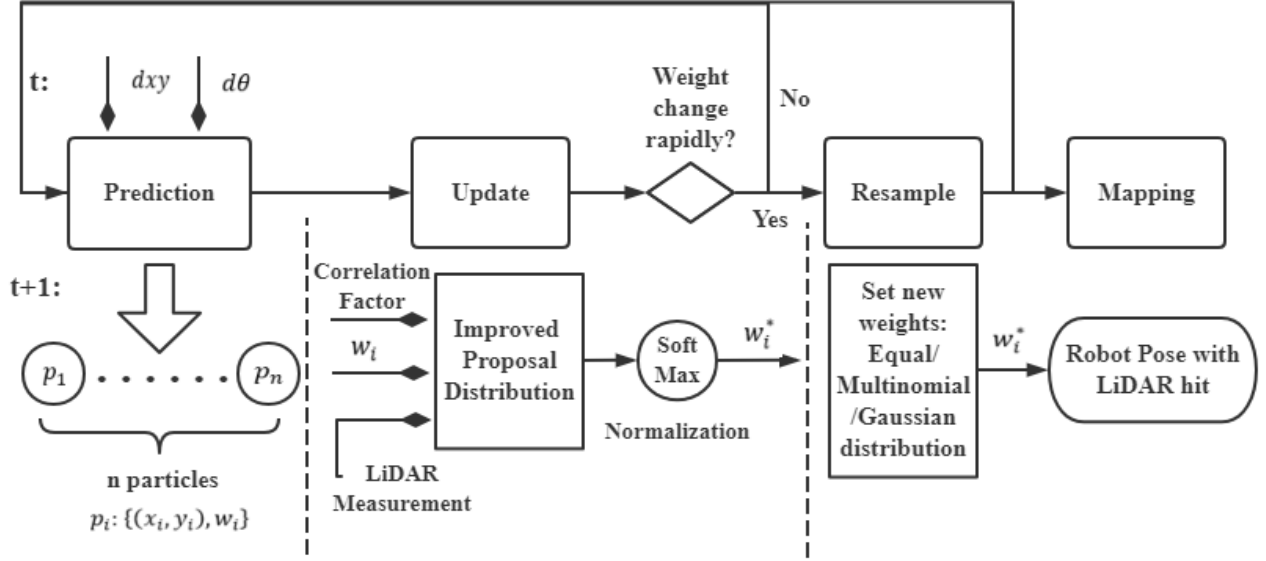


Figure 3.20: Flowchart of 2D SLAM process.

Where $p(x_{1:t} | z_{1:t}, u_{1:t-1})$ is the posterior probability distribution of robot's trajectory at certain known sensor observation and control command. Note that $(z_{1:t}, u_{1:t-1})$ can be considered as the potential trajectories, and the particle filter can be applied in solving this posterior. By solving this posterior, the estimated current pose of the robot can be determined, which means the localization has been done.

The SLAM framework can be presented as a cycle with three main steps. The first one is called prediction or sampling. The input of this step insists the change of angle $d\theta$ and pose (dx, dy) at time t , which can be read directly from gyroscope and odometer. With these inputs a certain number of particles in form of $\{(x_i, y_i), w_i\}$ at time $t + 1$ will be generated to represent the estimated positions where the robot will probably appear. The weight of these particles w_i , which represents the difference between target distribution and proposal distribution, will be given under the principle of importance sampling:

$$w_i = \frac{p(x_{1:t} | z_{1:t}, u_{1:t-1})}{\pi(x_{1:t} | z_{1:t}, u_{1:t-1})} \quad (3.24)$$

$$\pi(x_{1:t} | z_{1:t}, u_{1:t-1}) = \pi(x_t | x_{1:t-1}, z_{1:t}, u_{1:t-1}) \cdot \pi(x_{1:t-1} | z_{1:t-1}, u_{1:t-2}) \quad (3.25)$$

Since the main observation sensor is LiDAR, we use proposal distribution $\pi(x_{1:t} | z_{1:t}, u_{1:t-1})$ instead of target distribution because the computing amount of point cloud data is too heavy, which

cannot be modeled directly. In other words, the target distribution cannot be calculated even nearly. This proposal distribution can be determined through a recursive formulation. Then the weight is calculated as:

$$\begin{aligned}
 w_i &= \frac{p(x_{1:t} | z_{1:t}, u_{1:t-1})}{\pi(x_{1:t} | z_{1:t}, u_{1:t-1})} \\
 &= \frac{\eta p(z_t | x_{1:t}, z_{1:t-1}) \cdot p(x_t | x_{t-1}, u_{t-1})}{\pi(x_t | x_{1:t-1}, z_{1:t}, u_{1:t-1})} \cdot \frac{p(x_{1:t-1} | z_{1:t-1}, u_{1:t-2})}{\pi(x_{1:t-1} | z_{1:t-1}, u_{1:t-2})} \\
 &\propto \frac{p(z_t | m_{t-1}, x_t) \cdot p(x_t | x_{t-1}, u_{t-1})}{\pi(x_t | x_{1:t-1}, z_{1:t}, u_{1:t-1})} \cdot w_{t-1} \tag{3.26}
 \end{aligned}$$

$\eta = \frac{1}{p(z_t | z_{1:t-1}, u_{1:t-1})}$ is a normalization factor resulting from Bayes' rule that is equal for all particles.

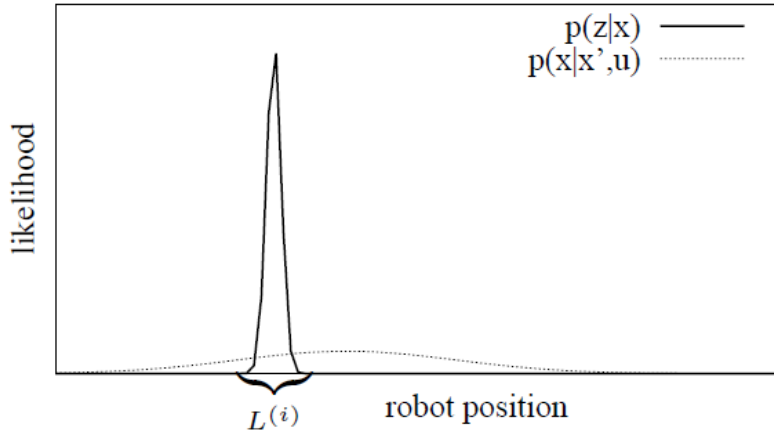


Figure 3.21: The two components of the motion model. Within the interval $L^{(i)}$ the product of both functions is dominated by the observation likelihood in case an accurate sensor is used. [12]

To improve the accuracy of the localization and mapping processes, we add the most recent observation from sensor z_t when generating the next generation of samples. This is because the sensor information, especially from LiDAR, is more precise than the motion estimate of the robot based on the odometry, as shown in Figure 3.21, where $L^{(i)}$ is the likelihood. By integrating sensor observation z_t into the proposal distribution, the sampling will be focused on the meaningful regions of the observation likelihood. The distribution after adding z_t becomes:

$$p(x_t | m_{t-1}, x_{t-1}, z_t, u_{t-1}) = \frac{p(z_t | m_{t-1}, x_t)p(x_t | x_{t-1}, u_{t-1})}{p(z_t | m_{t-1}, x_{t-1}, u_{t-1})} \quad (3.27)$$

Using this optimal improved proposal distribution, the computation of weights turns into:

$$\begin{aligned} w_t &= w_{t-1} \frac{\eta p(z_t | m_{t-1}, x_t)p(x_t | x_{t-1}, u_{t-1})}{p(x_t | m_{t-1}, x_{t-1}, z_t, u_{t-1})} \\ &\propto w_{t-1} \frac{p(z_t | m_{t-1}, x_t)p(x_t | x_{t-1}, u_{t-1})}{p(z_t | m_{t-1}, x_{t-1}, u_{t-1})} \bigg/ p(z_t | m_{t-1}, x_{t-1}, u_{t-1}) \\ &= w_{t-1} \cdot p(z_t | m_{t-1}, x_{t-1}, u_{t-1}) \\ &= w_{t-1} \cdot \int p(z_t | x')p(x' | x_{t-1}, u_{t-1})dx' \end{aligned} \quad (3.28)$$

As mentioned before, when modeling the environment with grid maps, a closed form approximation of an informed proposal distribution cannot be achieved directly due to the heavy amount of computation from laser sensor. But in here we can use sampling to reach the approximated form of the improved proposal. As shown in the framework in Figure 3.20, the first step is to sample a set of potential poses x_j from the motion model $p(x_t | x_{t-1}, u_{t-1})$. Note that if the observation likelihood is peaked, the number of pose samples is high since a dense sampling is needed for covering all the small areas of high likelihood. This will lead to a high number of particles, which means high amount of computation.

The method to solve this problem is that the meaningful area of the observation likelihood will be determined through a scan-matcher firstly, then the sampling will occur only in this meaningful area. For each particle i , the Gaussian parameters including mean μ_t^i and variance Σ_t^i will be estimated individually for K sampled poses $\{x_j\}$ in interval $L^{(i)}$:

$$\mu_t^i = \frac{1}{\eta^i} \cdot \sum_{j=1}^K x_j \cdot p(z_t | m_{t-1}, x_j) \cdot p(x_j | x_{t-1}, u_{t-1}) \quad (3.29)$$

$$\Sigma_t^i = \frac{1}{\eta^i} \cdot \sum_{j=1}^K p(z_t | m_{t-1}, x_j) \cdot p(x_j | x_{t-1}, u_{t-1}) \cdot (x_j - \mu_t)(x_j - \mu_t)^T \quad (3.30)$$

with the normalization factor:

$$\eta^i = \sum_{j=1}^K p(z_t | m_{t-1}, x_j) \cdot p(x_j | x_{t-1}, u_{t-1}) \quad (3.31)$$

Finally, the closed form approximation of the optimal proposal is obtained to generate the next generation of particles. Note that the weights will be calculated by using this proposal distribution as:

$$\begin{aligned} w_t &= w_{t-1} \cdot p(z_t | m_{t-1}, x_{t-1}, u_{t-1}) \\ &= w_{t-1} \cdot \int p(z_t | m_{t-1}, x') \cdot p(x' | x_{t-1}, u_{t-1}) dx \\ &\simeq w_{t-1} \cdot \sum_{j=1}^K p(z_t | m_{t-1}, x_j) \cdot p(x_j | x_{t-1}, u_{t-1}) \\ &= w_{t-1} \cdot \eta^i \end{aligned} \quad (3.32)$$

These weights will also be normalized through a SoftMax and become \tilde{w}_t . Now with these weighted particles, we are able to determine the location of the robot. As shown in the framework in Figure 3.20, we can transform the current 2D lase hit to the mapping coordinate with the current pose as the center to construct the map. The transformation matrix from LiDAR to robot's body frame is:

$$R_{L-b} = \begin{bmatrix} \cos \theta_n \cos \theta_h & -\sin \theta_n & \cos \theta_n \sin \theta_h & 0 \\ \sin \theta_n \cos \theta_h & \cos \theta_n & \sin \theta_n \sin \theta_h & 0 \\ \sin \theta_h & 0 & \cos \theta_h & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.33)$$

where θ_n is neck angle and θ_h is head angle.

The transformation matrix from robot's body frame to global map is:

$$\begin{aligned}
R_{b-G} = & \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta_{yaw} & -\sin \theta_{yaw} & 0 & 0 \\ \sin \theta_{yaw} & \cos \theta_{yaw} & 0 & 0 \\ & 0 & 0 & 1 & 0 \\ & 0 & 0 & 0 & 1 \end{bmatrix} \\
& \cdot \begin{bmatrix} \cos \theta_{pitch} & 0 & \sin \theta_{pitch} & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_{pitch} & 0 & \cos \theta_{pitch} & 0 \\ & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta_{roll} & -\sin \theta_{roll} & 0 \\ 0 & \sin \theta_{roll} & \cos \theta_{roll} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.34)
\end{aligned}$$

where θ_{yaw} , θ_{pitch} , and θ_{roll} correspond to the yaw, pitch and roll read from IMU, x , y , z is the trajectory of the robot.

Also, since the equipped 16-channel LiDAR can provide 38400 points each scan, some operations was added to de-noise and decrease density for the point cloud data. A threshold will be set to remove those point cloud hit on the ground at the beginning as one of the data pre-processing operations.

After each loop, there will be a judgement to assess the quality of our particles for deciding whether the resampling is necessary. In resampling step, those particles with low importance weights w_t will be replaced by particles with higher weights. This step can make sure that the overall number of particles will remain finitely since we do not want too many particles to retard the running speed. On the contrary, resampling may also remove good samples from the filter which can lead to particle impoverishment. In that case, there is no doubt that this judgement step (or as people called ‘‘adaptive resampling’’) is necessary to find a criterion for deciding when to perform the resampling step. The index representing the effective sample size to estimate how well the current particle set represents the target posterior was introduced, and in here this quantity was calculated according to the formulation of Doucet as

$$N_{eff} = \frac{1}{\sum_{i=1}^N (\tilde{w}^i)^2} \quad (3.35)$$

If the sample were drawn from the target distribution, their importance weights would be equal to each other due to the importance sampling principle. The worse the approximation of the target distribution, the higher is the variance of the importance weights. The threshold of resampling was set at $N/2$, which means the resampling will occur when N_{eff} dropped below half number of particles at each time.

Then with the transformed 2D laser hits, the occupied grid will be generated and updated through every circulation to ensure the whole function is real time.

3.5.4. Path Planning

In this thesis, the scenario of implementing path planning can be described and classified as dynamic, global, and exact. The working principle of this function is based on a constructed global map, which is actually streamed from SLAM function as the output.

The pipeline of path planning process can be described in Figure 3.22. As mentioned before, the SLAM function will pass the constructed global occupied grid map to path planning function as a perception of the environment. The resolution of the occupied map has been set on $20 \text{ cells}/m$, and the size of the map is $30 \times 30m$.

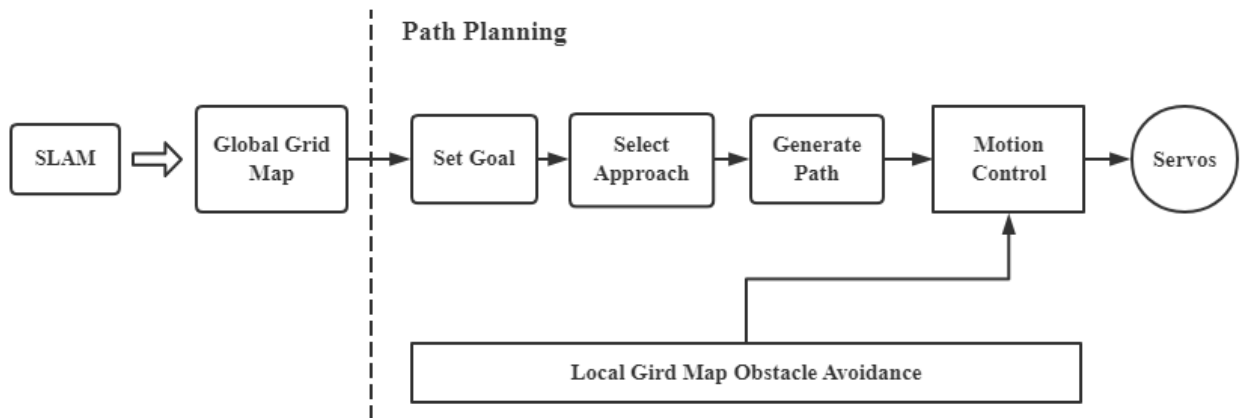


Figure 3.22: Flowchart of path planning process.

Then, the grid map will be transformed to another grid map on the same size and resolution

Table 3.7: Meanings corresponding to different colors on map for path planning.

Color	Meaning
Black	Obstacle
White	Free Space
Red	Visited
Blue	Final Path
Light Blue	Save Zone
Pink	Dangerous Zone

for path planning according to the different colors on the map. Table 3.7 has shown the meaning corresponding to different colors.

The user will be asked to select a destination on white grids. The current location will also be passed from SLAM as the default start point. The save zone and dangerous zone has been set along the edge of the obstacles. The size of save zone is 3 grids while the size of dangerous zone is 2 grids. Note that the purpose of setting save zone and dangerous zone is to leave enough redundancy for safety concern.

Once the path planning function has been activated, the robot will be ordered to stop and wait for command. Users can select the method of planning path at the beginning including:

- A Star (A*)
- A Star Heuristic
- Dijkstra
- Greedy Approach
- Heuristic Weighted
- Reinforcement Learning

The principle of these mentioned path planning method has been introduced in CHAPTER 2, so in here we just code them into our function. No matter the method under selected, the path will be translated into a set of command and send to servos to drive the robot from start to destination. In here, due to the time limitation, we proposed a closed-loop control to ensure the robot was led to the right destination with the feedback from wheel encoder (speed) and SLAM (trajectory), but we did not implement the closed-loop control in this part. Figure 3.23 presents an example of translating path to commands. The green grid is the start point, while the yellow grid is the destination. The current pose of robot is heading the front of the south, so that the command will be:

*F&SP, L&SP, F&SP, F&SN, F&Z, R&SP, F&SP, F&SN, F&Z, L&SP, F&SP, F&SN, F&Z,
L&SP, F&SP, F&SN, F&Z, R&SP, F&SP, F&SN, F&Z*

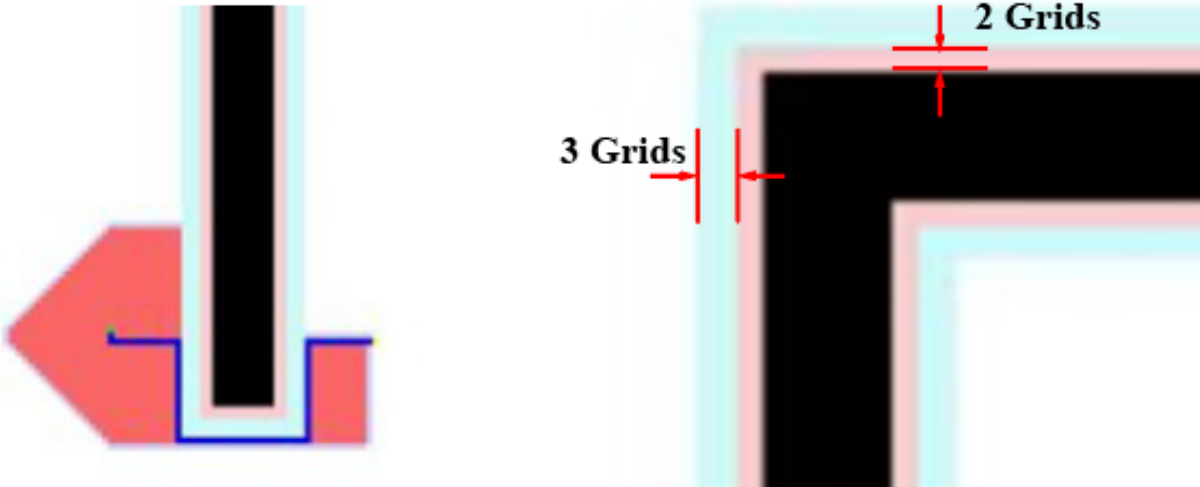


Figure 3.23: Save zone (light blue) and dangerous zone (pink) on map for path planning.

Which means turn left, go forward, brake, turn right, go forward, brake, turn left, go forward, brake, turn left, go forward, brake, turn right, go forward, brake, stop. Thus, the process of path planning and navigation has been completed. Note that, in here we use Manhattan distance from the start grid to destination grid, which is a standard heuristic for a grid map.

Note that the priority of obstacle avoidance is higher than path planning, which means if the local grid map is showing that a static or dynamic obstacle is within the danger distance, the path planning function will be interrupted to ensure the robot will not hit something.

3.5.5 Simulation

This part is actually not under the main topic of this thesis. During the COVID-19 self-quarantine period, the testing environment for the proposed robot in real world is not realistic. In that case, we launched a small project about simulating the robot with whole functionalities except data fusion based on robot operating system (ROS) and Gazebo. This part will not be introduced in detail, but the simulation environment construction and algorithms used will be introduced briefly.

Simulation Environment: The whole simulation environment is based on ROS Kinetic and Gazebo 8.6. The robot in this simulation was built in shape of a single cylinder with two differential

driving wheels and one unpowered omnidirectional wheel, which shares the same kinetic model as the robot proposed in this thesis in real world, as shown in Figure 3.24. The two little black blocks at the top of the cylinder are camera and Velodyne 3D LiDAR. The speed, heading, and acceleration information about the robot dynamics can be subscribed through built-in libraries with ROS. For controlling the robot, ROS also provides the built-in libraries to publish the control command to the model of robot. It can be controlled through keyboard, or we can send commands generated by other functions such as obstacle avoidance or navigation to it.

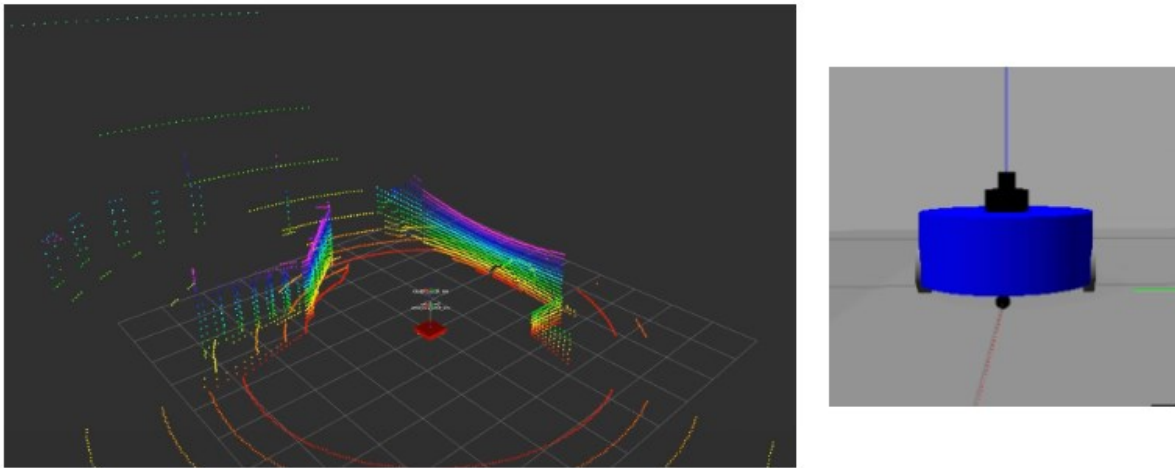


Figure 3.24: Model of robot in simulation (right side) and visualization of LiDAR in simulation (left side).

Then, with the model of robot, we arranged two simulation scenarios including both indoor and outdoor from built-in models of objects in Gazebo, as shown in Figure 3.25 (indoor) and Figure 3.26 (outdoor). The outdoor scenario simulates a real city with the focus on traffic, which concludes the trafficway, buildings, sidewalks, traffic signs and lights and so on. Note that it also concludes some pre-programmed dynamic objects including pedestrians and moving cars.

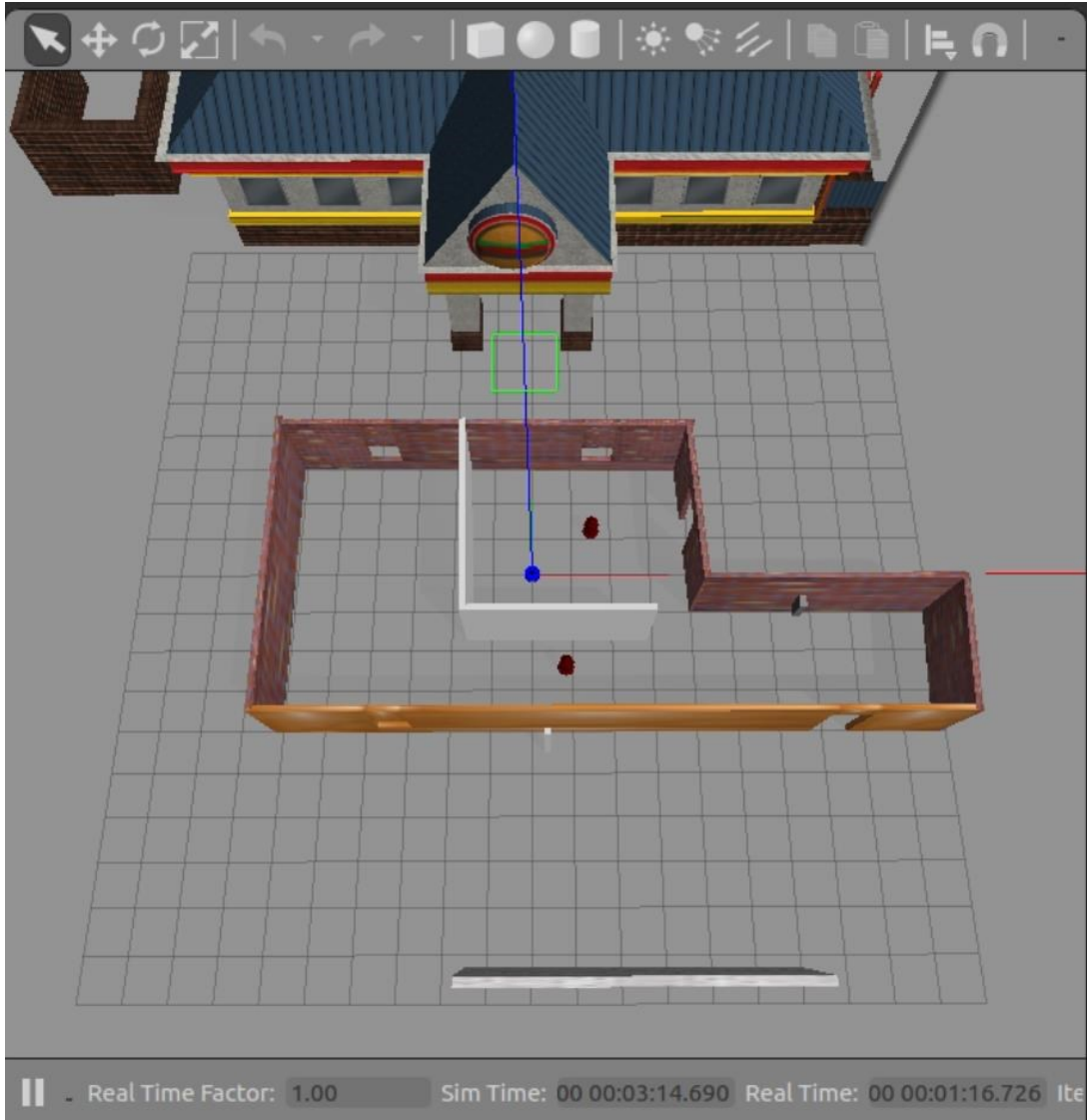


Figure 3.25: Indoor testing environment on simulation. The blue dot is our robot model, and the main obstacle includes wall, fire hydrant, and fast food restaurant.

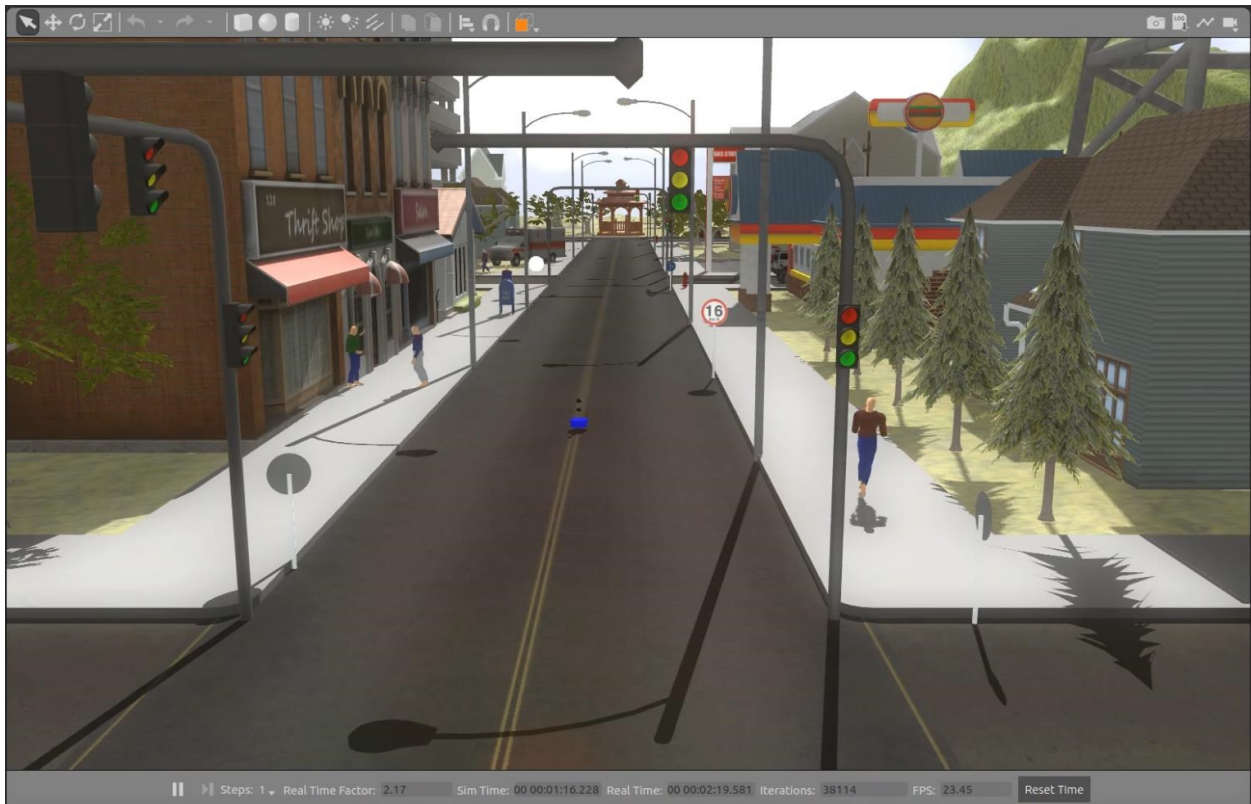


Figure 3.26: Outdoor testing environment on simulation. Overview (bottom side) and street scene (top side).

In this simulation, the functionalities featured in real robot including obstacle avoidance, 2D SLAM, and path planning have been implemented except data fusion. Besides, the 3D SLAM has also been implemented since the ROS provides open-source algorithm for 3D SLAM. Note that all these functions are supported by open-source, and the main work of this project is integrating them together simultaneously in the simulation robot.

- **Obstacle Avoidance:** Differed from the robot proposed in this thesis, the obstacle avoidance in simulation is realized based on path planning. The robot will follow the path generated and circle around the obstacles.
- **2D SLAM:** Open-source library, Gmapping.
- **3D SLAM:** Two open-source libraries, Loam and Lego-loam. [47]
- **Path Planning:** Based on the map constructed by 2D SLAM, manually select destination on the map. The localization algorithm is based on AMCL.

CHAPTER 4

TESTING AND RESULTS

The performance of this proposed robot can be reflected in many aspects. Some aspects of the performance can be quantized through universal indicators such as accuracy of map or run time. However, in this thesis some functions of the robot were just tested to present that the robot has the ability of such autonomous driving technologies, and some further advanced algorithms can be developed based on this platform. Thus, in this chapter, the test environment, procedure and results will be introduced to accord readers an overview about how we present testing on this robot, while not only just implementing those functions mentioned together but also fusing functions together and considering it as an autonomous driving platform.

4.1 Test Procedure and Environment

In this thesis, there is no fixed place as test environment because the robot was built to have the operational capability under different scenarios. On the other hand, different functions of the robot may need distinct environments to test the performance individually. The basic physical environment of testing the robot can be divided into indoor and outdoor mainly including research lab, university building hallway, my personal room (the robot had been tested in my room sometimes due to the COVID-19) and campus parking lot as shown in Figure 4.1. The environments will be introduced in detail together with test procedures according to different uses on below.

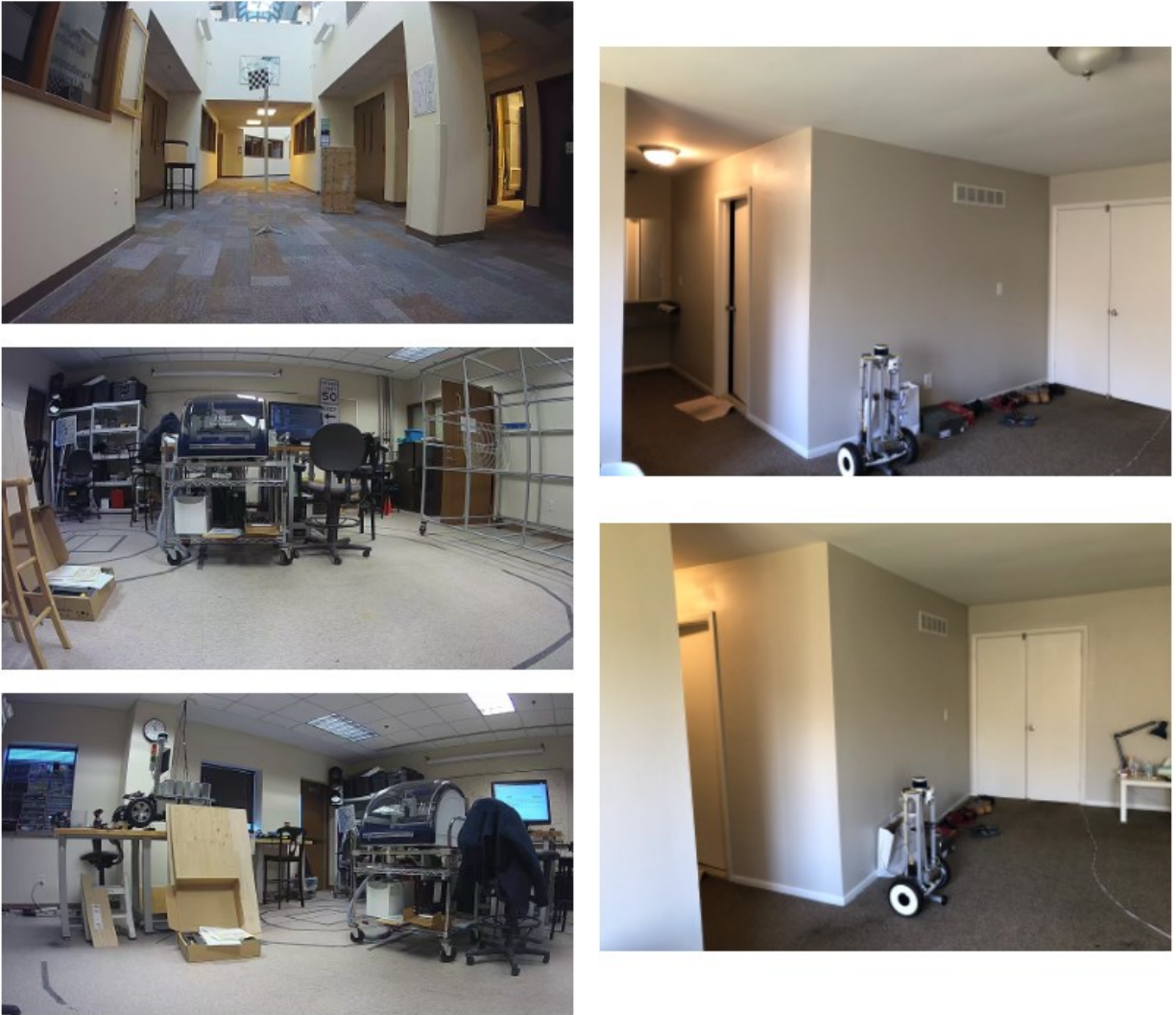


Figure 4.1: Pictures of several test environments. Upper left: university building hallway. Middle and downer left: research lab. Right: Room.

For testing data fusion: The target object can be described as shown in Figure 4.2. Since the target of this function is to back-project pixels from images to point cloud data, we first set a few landmarks such as the $7 \times 10 \times 30$ checkerboard and two rectangle planks on chairs at an indoor scenario (university's building hallway). Then we put the robot outside the apartment, use a sedan vehicle and a walking person as the target objects, as also shown in Figure 4.2.

For testing obstacle avoidance: The indoor environment includes research lab, university building hallway and my room, while the outdoor environment includes the campus parking lot.

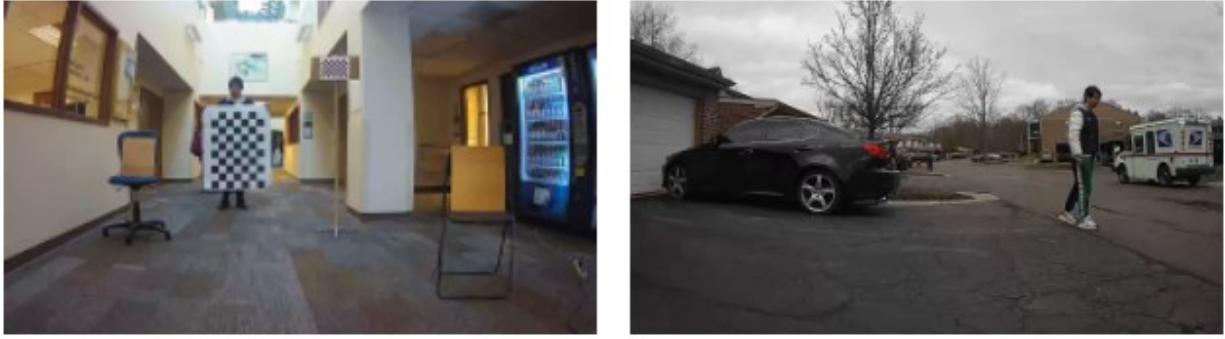


Figure 4.2: Testing environment for data fusion. Left: indoor scenario. Right: outdoor scenario.

The selected obstacles in these scenarios include both static and dynamic as shown in Table 4.1 as well as other specification.

Table 4.1: Specification of different test environments.

	Scenario	Obstacles		Size	Special Feature
		Static	Dynamic		
Indoor	Research Lab	Table, Chair, Wall,	Moving person (me)	Small, around $5 \times 5 m$	Relatively low-friction and even ground. Narrow space, obstacles are placed randomly with different heights.
	University Building Hallway	Wall, Pillar, Vending Machine	Moving people	Medium, around $5 \times 10 m$	With carpet on the ground which results in higher friction. Relatively clear space, obstacles are placed orderly with same height.
	Room	Wall, Bed, Table	Moving person	Small, around $5 \times 3 m$	Same as hallway, carpet on the ground. Narrow space, obstacles are placed randomly with different height.
Outdoor	Campus Parking Lot	Cars, Street Light,	Moving people	Large, around $10 \times 10 m$	Cement and tarmac surface, relatively medium friction but uneven and rugged ground. Space in here is very clear except between cars, obstacles are placed orderly with same height.

For SLAM and path planning: Since the prerequisites of planning path is constructing a map for environment, we test these two functions simultaneously in the same scenario mostly. Also, for testing the autonomous driving ability of the robot, the test environments for these two functions are the same as for obstacle avoidance except research lab because we could not present testing in there due to COVID-19. Note that the complexity of the environment can affect the accuracy of mapping, so that we sort the complexity of these scenarios according to the number and placing of obstacles, as shown as followed:

Campus parking lot > Room > University Building Corridor

Due to COVID-19, the main testing scenario for SLAM and path planning is my personal room. It has a medium complexity and size of 5×3 with obstacles including wall, bed, table, chair, boxes and moving person. The space in this environment is relatively not clear with a narrow corridor on the side. For this scenario, the performance of SLAM and path planning can be shown mainly as mapping accuracy and run time speed, which will be presented later in CHAPTER 4.2.3 and 4.2.4.

4.2 Test Results

4.2.1 Obstacle Avoidance

There is not a specific indicator presenting the performance of obstacle avoidance. However, the general performance of this function can be embodied through testing at these mentioned scenarios. The best way to present the performance of obstacle avoidance will be using a demo video. But in here, we will present the result of constructed 2D local grid map, which is the guidance and foundation for obstacle avoidance.

As one of the mentioned scenarios, the raw 2D point cloud of my room has been shown in the right side of Figure 4.3 below. By comparing the 2D point cloud and real picture of my room, it is obvious that the basic geometrical information has been reflected and restored in detail. For example, the door left open ajar on the side of the corridor can be identified from the 2D point cloud easily. The geometric specification of map has been shown in Table 4.2, which is the same as the size and refresh frequency mentioned in CHAPTER 3.

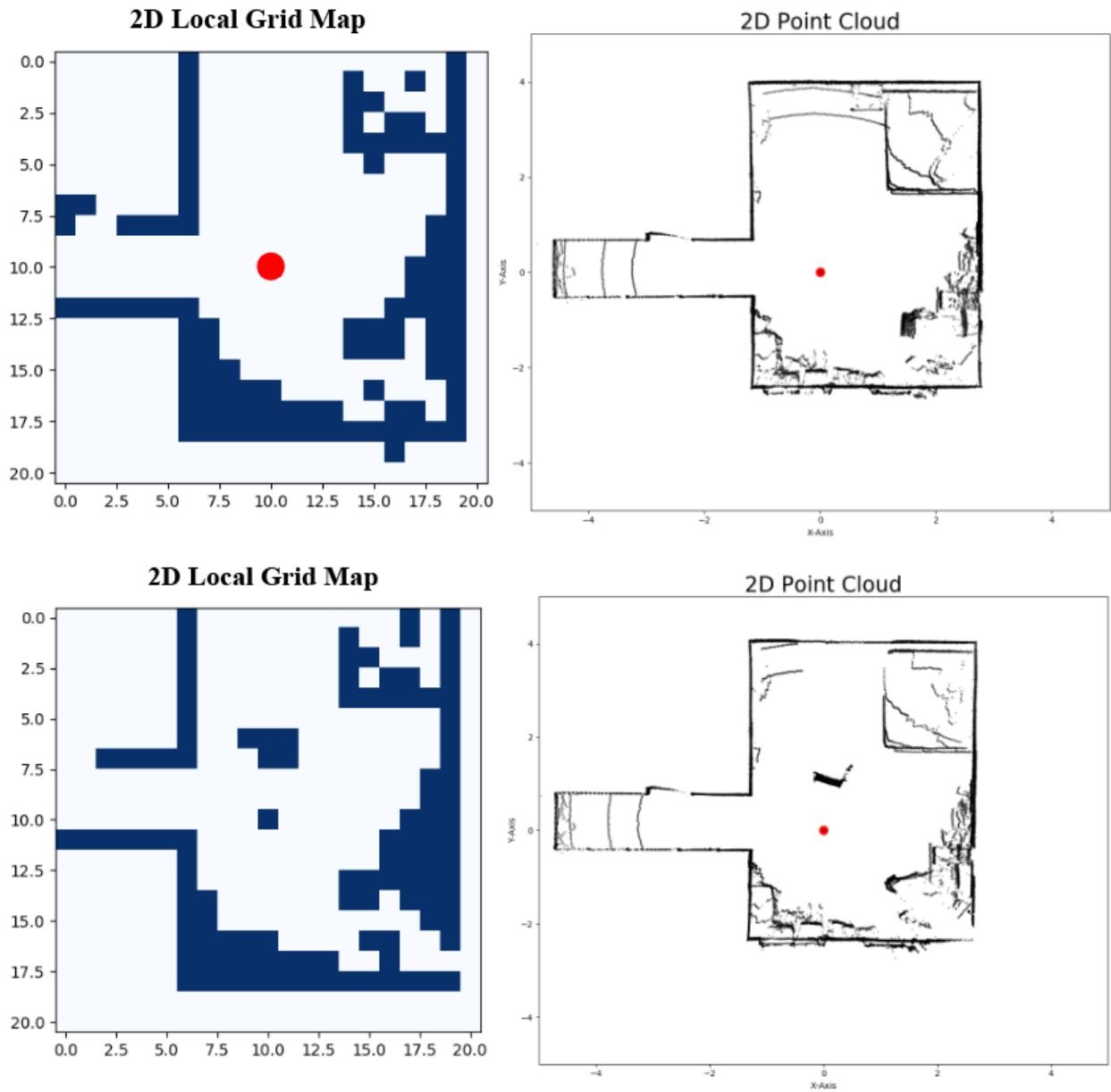


Figure 4.3: Plain view of 2D local grid map and 2D point cloud.

Table 4.2: Specification of 2D local grid map.

Grid Size	0.3m
Map Size	$3 \times 3m$ or 21×21 grids
Map Refresh Frequency	More than 5Hz

However, some outliers have also been collected since the 16 channel LiDAR has a vertical scan range in $\pm 15^\circ$ as mentioned before. For example, the cambered curve appeared at the end of the corridor and near the bed is called ground hit, which is the points reflected by the ground when the laser beam #0 and #1 hit the ground. The appearance of such outliers does not mean that area is occupied by any obstacles. Hence, with the data pre-processing methods mentioned in previous chapter, we eliminate these outliers and set a confidence threshold to simplify the point cloud when transform it to grid map.

As the result, the 2D local grid map has been shown in Figure 4.3 next to point cloud. It is clear that the necessary details of real world have been mostly restored and expressed on the map such as the door left open ajar, while the outliers have been deleted and the whole map performs much more clear than raw data. A map such as this one can be passed for extracting the fuzzy language of $\{Left\ Distance, Front\ Distance, Right\ Distance\}$, and it can be used to guide the robot for avoiding obstacles and exploring the environment. At this scenario, the robot close area of all left, right, and front is clear, so that the robot is free to move around but according to the fuzzy control rule base, the robot should go forward.

If there is another obstacle moving across or around the robot, which is also known as a dynamic obstacle, we ensure the safety of the robot by maintaining an adequate refresh frequency. The refresh frequency of obstacle avoidance depends on and less than the scan frequency of LiDAR, which is 10Hz. Through practical testing, the 2D grid map will update less than 0.3 sec. Considering the lag from sending command to reaction of servos, and the inertia of robot motion, the refresh frequency satisfied the safety concern.

4.2.2 Data Fusion

First the calibration of camera result will be presented as shown in Figure 4.4. The detected points, checkerboard origin, and reprojected points have been marked in Figure 4.4 above, while the image below has shown the result of undistorted image.

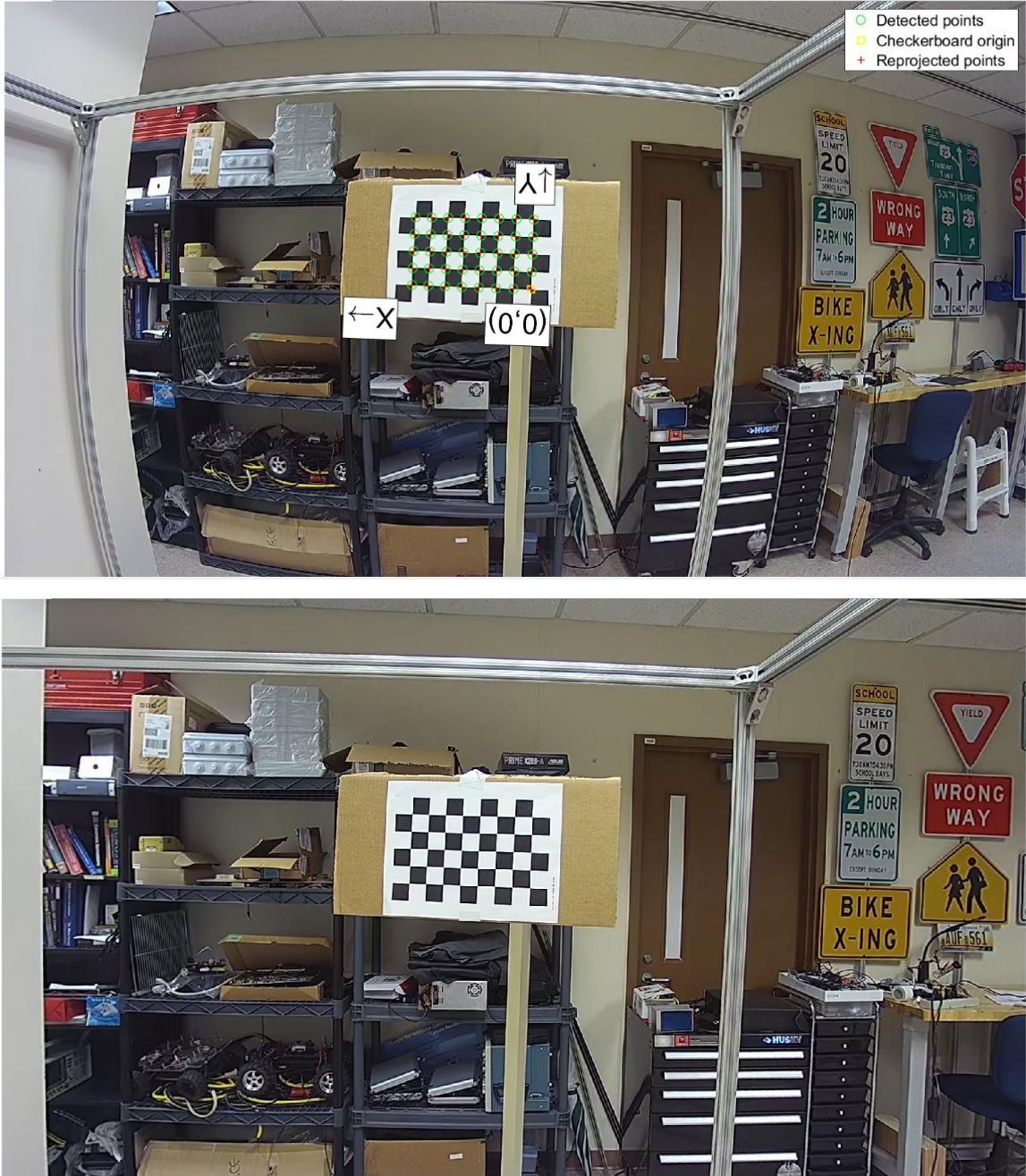


Figure 4.4: Upper: raw image capture from the camera. Downer: undistorted image after calibration.

Then with a set of 25 images, we are able to calculate the reprojection error of camera calibration session. The result has been shown in Figure 4.5, and the overall mean error is 0.13 *pixels*, which is acceptable. The 3D representation of checkerboard in different position at a camera-centric order has also been shown in Figure 4.5.

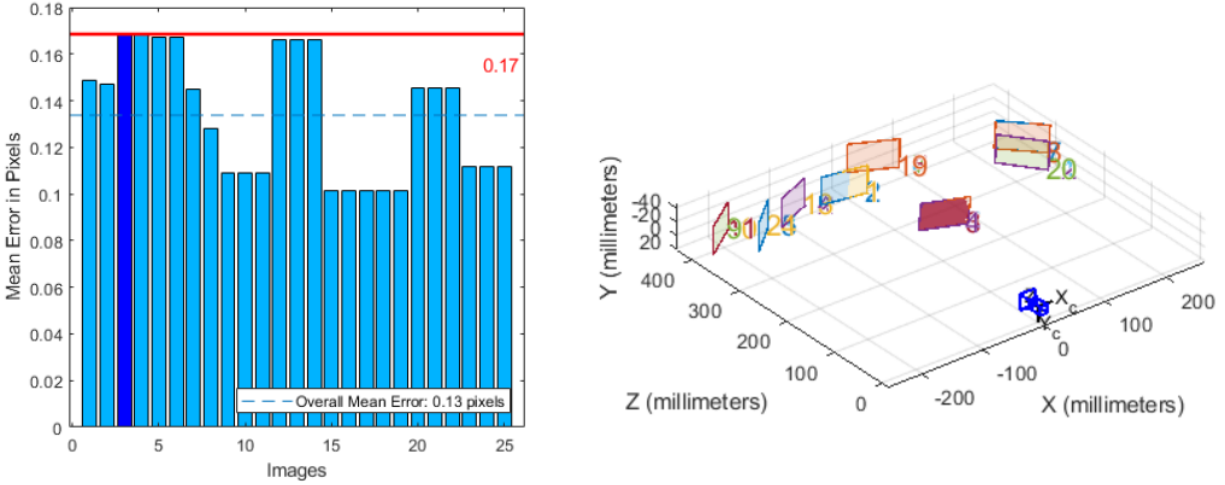


Figure 4.5: Left: mean error in pixels of 25 images. Right: position of checkerboards in a camera-centric basis.

Also, some other numerical results including the intrinsic matrix has been shown as below:

- Intrinsic matrix: $\begin{bmatrix} 1210.70 & 0 & 0 \\ 0 & 1211.87 & 0 \\ 969.62 & 541.14 & 1 \end{bmatrix}$
- Principal Point: $[969.62 \quad 541.14]$
- Radial Distortion: $[-0.3644 \quad 0.1177]$
- Mean Reprojection Error: 0.1338

After these, the result of detecting checkerboard corners has been shown in Figure 4.6. The different color on point cloud means different reflection intensity.

For the larger checkerboard, find the four corners of the whole checkerboard as the reference points, and match those points corresponding to undistorted image can also calculate the extrinsic parameters, as shown in Figure 4.7.



Figure 4.6: Corners of checkers on point cloud.

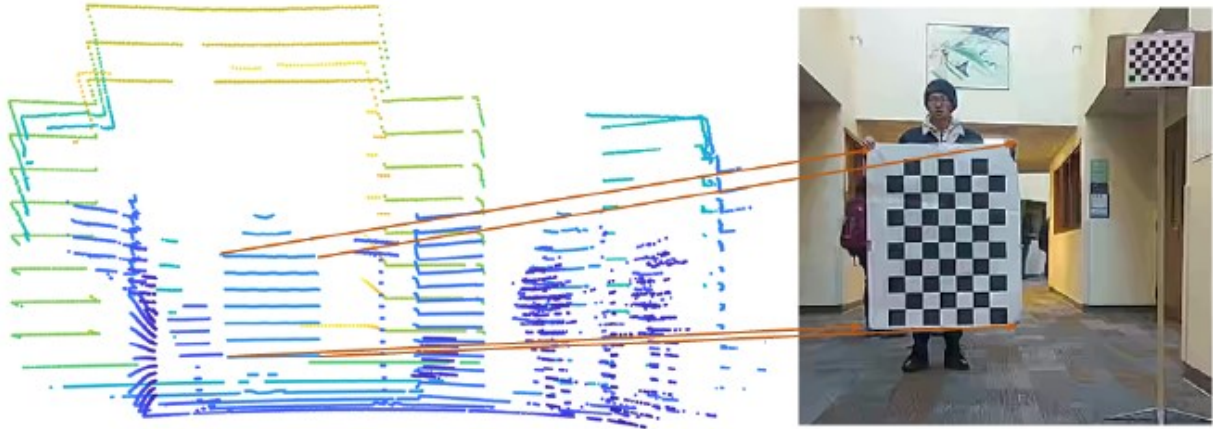


Figure 4.7: Four corners of larger checkerboard with respect to image and point cloud.

Finally, the pixels of image can be back projected to point cloud as shown in Figure 4.8.

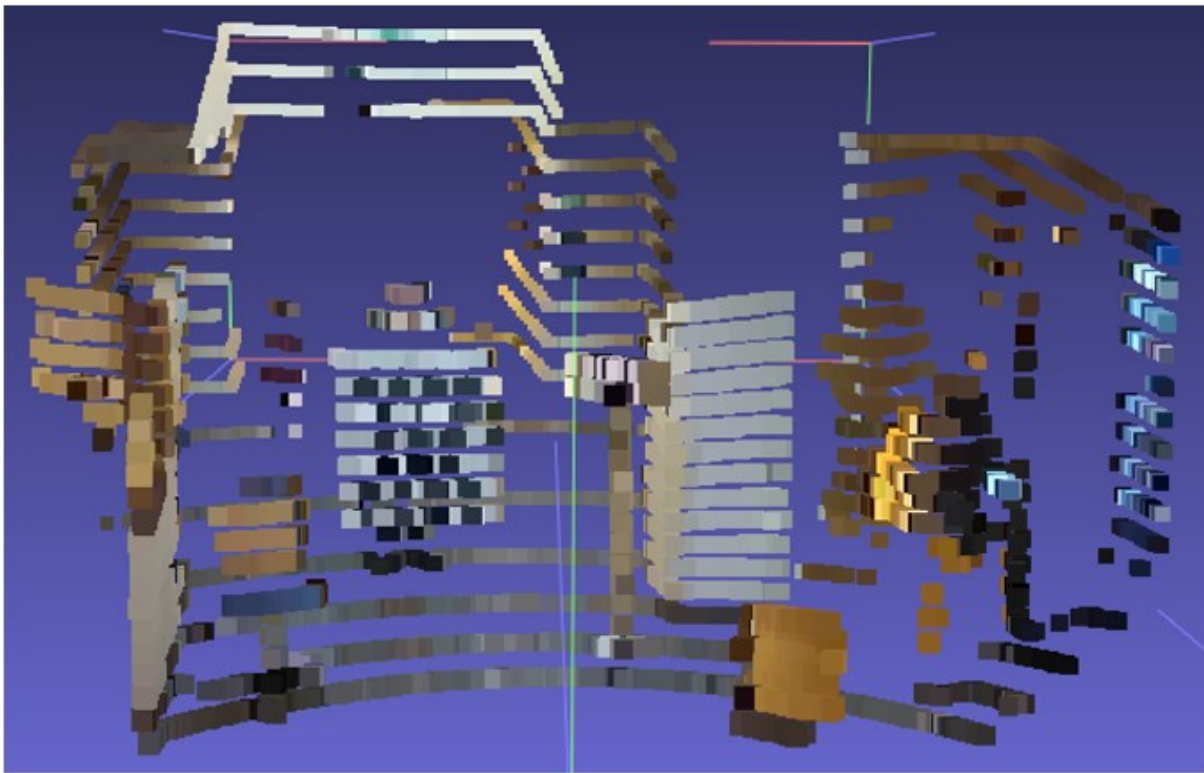


Figure 4.8: Colored point cloud of indoor scenario after transformation.

With the intrinsic and extrinsic parameters, some other data captured by this robot can also be fused since the relative spatial position relationship between LiDAR and camera remains the same.

Here we illustrate this by presenting another example in outdoor scenario. The undistorted image captured from camera has been shown in Figure 4.9. while the 3D point cloud captured from LiDAR in the same scene has been shown in Figure 4.9. The result of fusing the 3D point cloud and 2D image together can be presented as following:

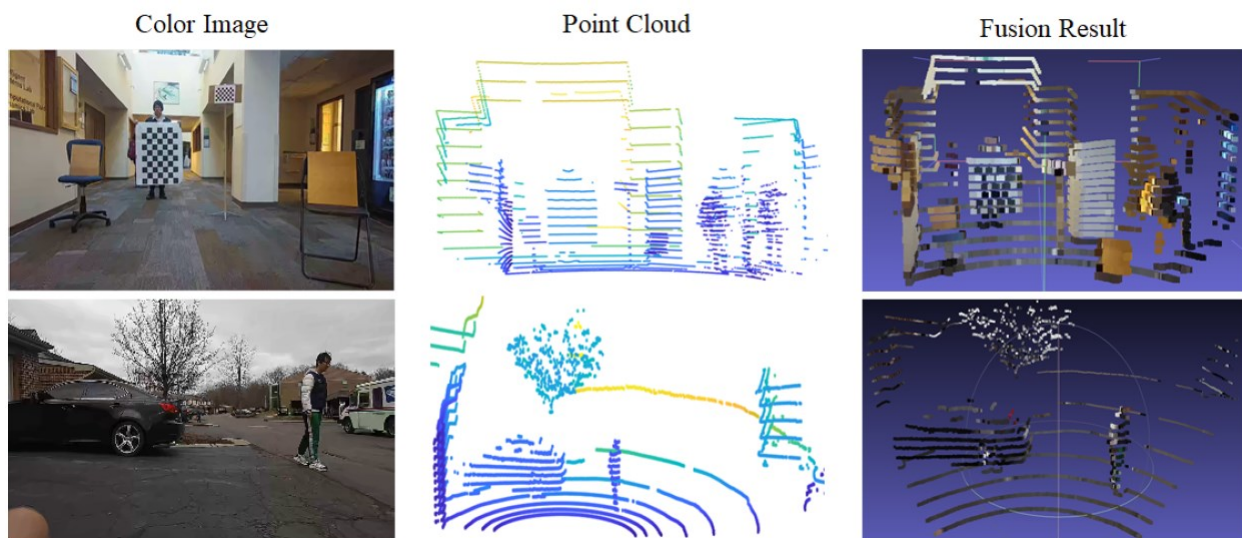


Figure 4.9: Color image, point cloud, and fusion results of indoor and outdoor scenario.

4.2.3 2D SLAM

The 2D SLAM includes two main targets, localization and mapping. In here, since we are using the dataset capture by our own, we did not use a fixed ground truth to verify the accuracy of the map constructed. Besides, we only have performed this function on a single scenario, which is my room.

Table 4.3: Measurements of objects in map and reality.

Object	Map Measurement		Reality Measurement
	In grids	In cm	
Bed (Queen Size)	42 <i>grids</i>	210 <i>cm</i>	205 <i>cm</i>
Wall	51 <i>grids</i>	255 <i>cm</i>	247 <i>cm</i>

However, the accuracy of map constructed can be defined as accurate since the size of some of the landmarks from the map generated can match the corresponding objects in actual world. In

here we chose the bed and wall as the landmark, and we measure the length of objects both in map and reality as shown in Table 4.3.

Here are also some results showing both the trajectory and the map. The meaning of different color represents:

- Grey: Unexplored zone
- Green: Current LiDAR hit
- Black: Obstacles or occupied
- White: Free zone
- Blue: Trajectory

To illustrate the process of map construction, we shielded half of the scan range of LiDAR from 360° to 180° , and then we let the robot move and explore randomly in the environment, as shown in Figure 4.10. The blue grids have recorded the trajectory of the robot when moving. Also, we let the robot stay in place and turn it around in the same environment, and the map as shown in Figure 4.11.

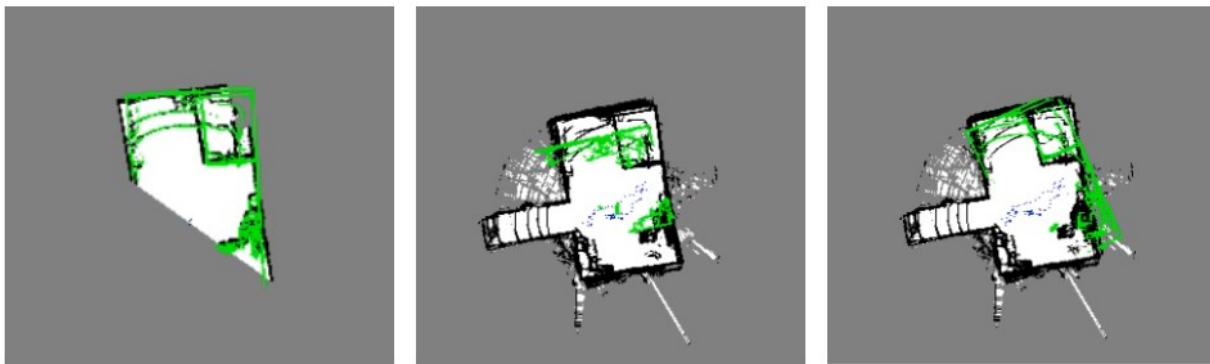


Figure 4.10: Map constructing process with current LiDAR hit (green color) before removing ground hit.

Obviously, the map generated when the robot is stationary has higher quality than moving. The possible reason can be various, but the most influential one is that the LiDAR may capture some outliers when the robot is moving because the vibration caused by the even ground. Although we have already applied filter and threshold for matching different frames captured by LiDAR, some outliers still appeared. Actually, this is a common phenomenon when mapping through a LiDAR

both in 2D and 3D. We improve something is that we removed the ground hit, and we integrate this function together with other functions simultaneously.

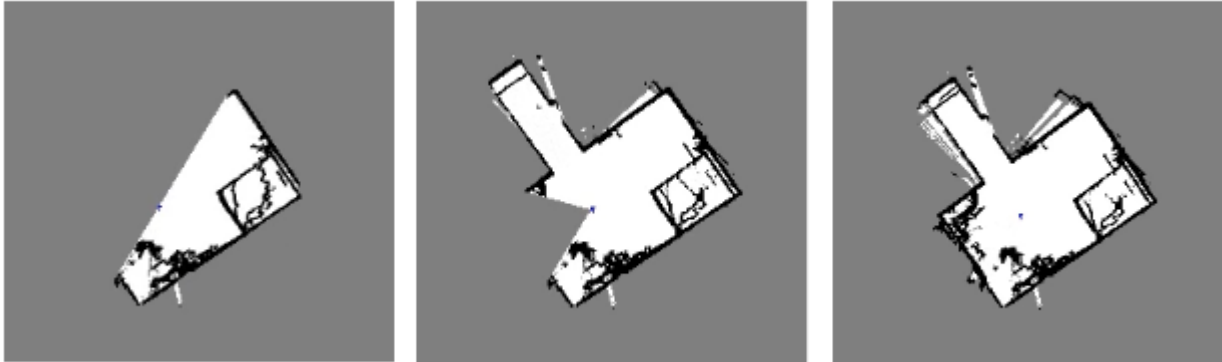


Figure 4.11: Map constructing process without current LiDAR hit after removing ground hit.

At last, the real scale ($30 \times 30 \text{ m}$) of the map and corresponding topographic diagram has been shown in Figure 4.12

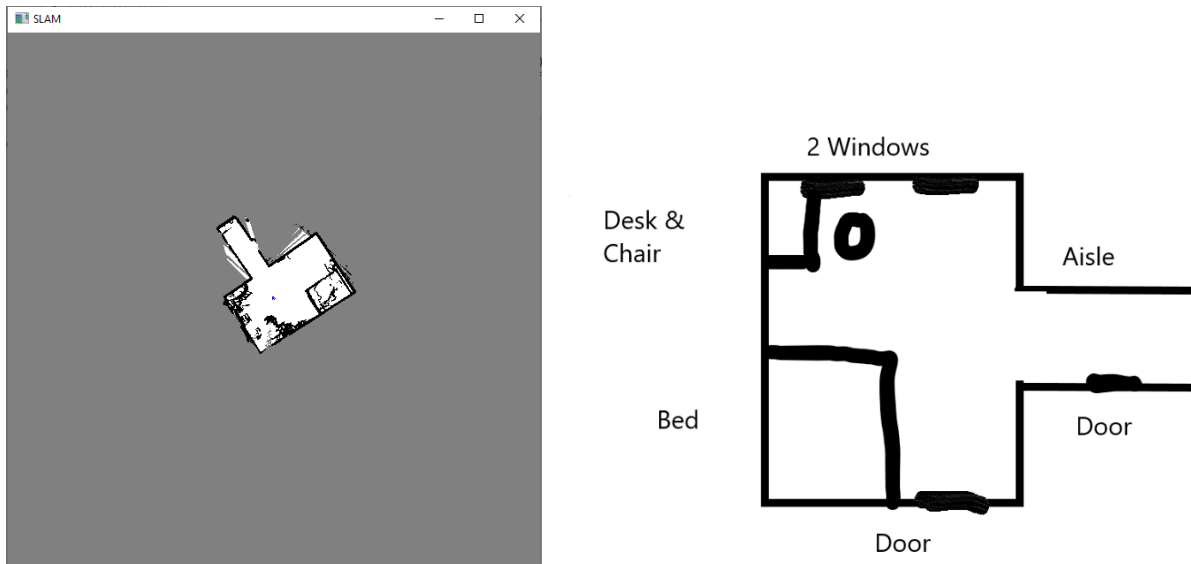


Figure 4.12: Left: result of global 2D grid map in scale of $30 \times 30 \text{ m}$. Right: corresponding scheme of room.

4.2.4 Path Planning

As mentioned before, the input of path planning is the grid map constructed from SLAM. Firstly, the map for path planning converted from SLAM has been shown in Figure 4.13. The map for path planning still share the same scale with the original map, which is 600×600 *grids*, but the difference is that we set the save zone (light blue color) and dangerous zone (pink color) alongside the obstacles.

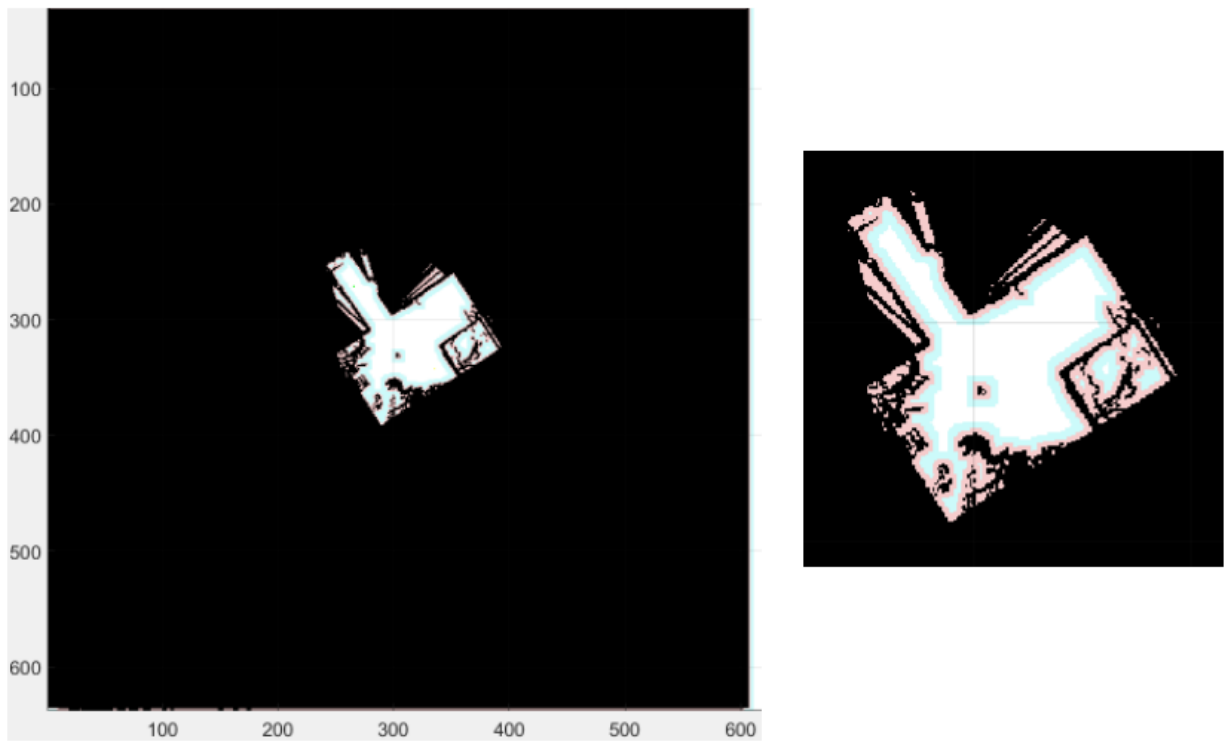


Figure 4.13: 2D global grid map for path planning in scale of 600×600 *pixels* (left) and in detail (right).

Then, after setting the start (default is the current location of robot) and destination and selecting the method, the path will be generated as shown in Figure 4.14. The red color means the area that has been traversed through the planning process. It is clear that the path generated through different method presents the same, which means that under a simple scenario with not too much obstacle to circle around, the results planned by best-first search will not show too much difference.

Table 4.4: Path scoring time for different algorithms.

Method	Path Scoring Time
Dijkstra	1.114s
A*	0.692s
A* Heuristic	0.094s
Greedy	0.044s
Heuristic Weighted	0.043s

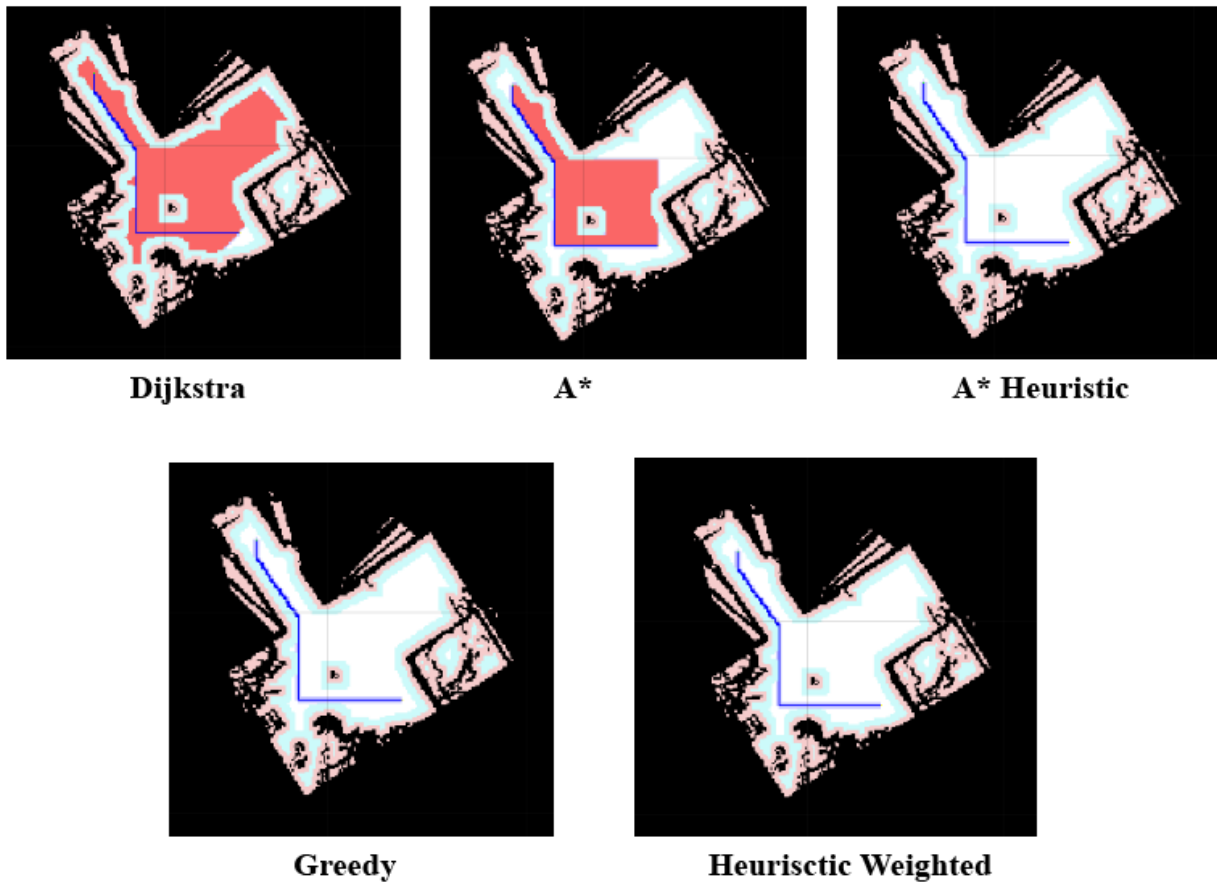


Figure 4.14: Path generated from different algorithm with the same start and destination.

Also, the time consumed through scoring the path under different methods has been shown in Table 4.4 above. Though the path generated presents the same, the time efficiency of different methods illustrates significant difference. The Dijkstra algorithm consumed the longest time, while Greedy and heuristic weighted algorithm planned the path at almost the rapidest speed.

4.2.5 Simulation Results

Here are some results of the simulation. The first one is the 2D map constructed in indoor scenario and path planning. The right side in Figure 4.15 is the model of both the building and the robot (blue one is the robot), while left side is the 2D map constructed and video visualization streamed from camera. The orange spot is the current position of robot, and the green curve is the path planned to the preset destination.

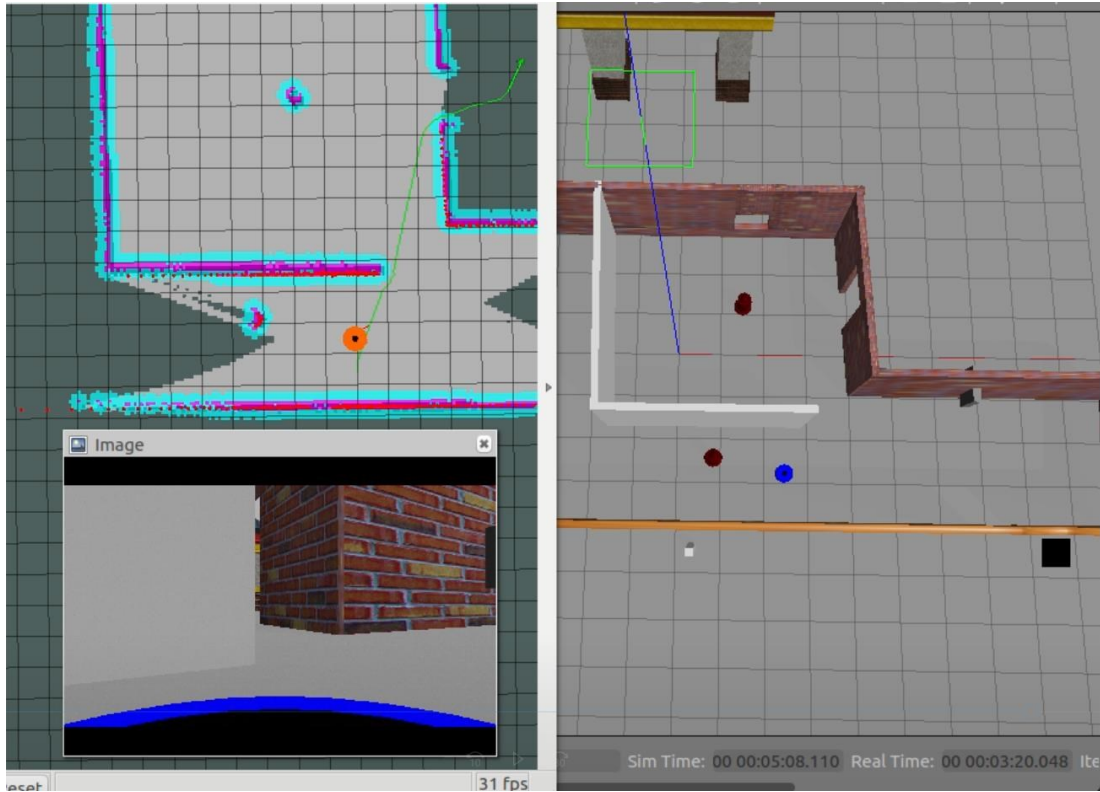


Figure 4.15: Path planning and navigation of simulation robot. The window tagged “Image” presents the visualization of virtual camera set on the robot model.

Then the 2D SLAM result constructed at outdoor scenario has been shown in Figure 4.16. The black outline is the outer shape of several buildings.

The process of 3D SLAM has been shown in Figure 4.17. The robot has been circled in red color on the right side, and the corresponding position of the robot has also been shown on the left side.



Figure 4.16: 2D grid map generated from Gamapping of simulation.

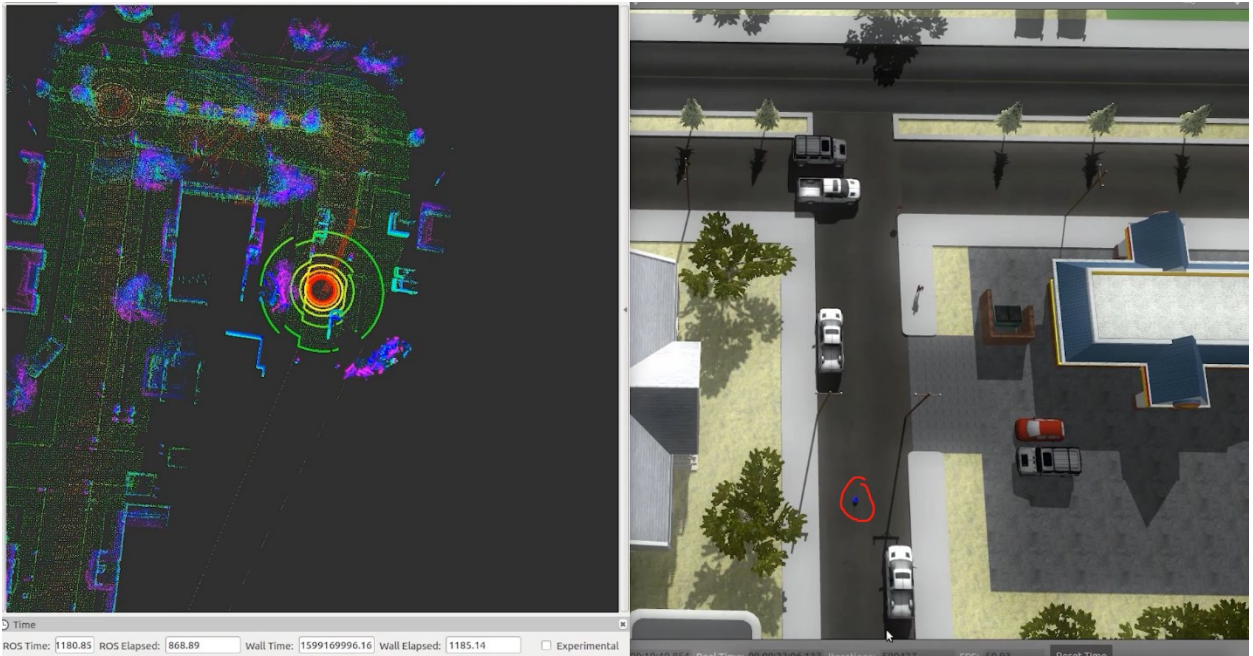


Figure 4.17: 3D laser map constructing process of simulation. The left is the 3D laser map, and the right is the corresponding real scene in simulation environment.

At last, the results of 3D SLAM have been shown in Figure 4.18 and Figure 4.19. It is obvious that the 3D map constructed by Loam algorithm has higher density in point cloud than Lego-loam algorithm. But both two map has excellent quality.

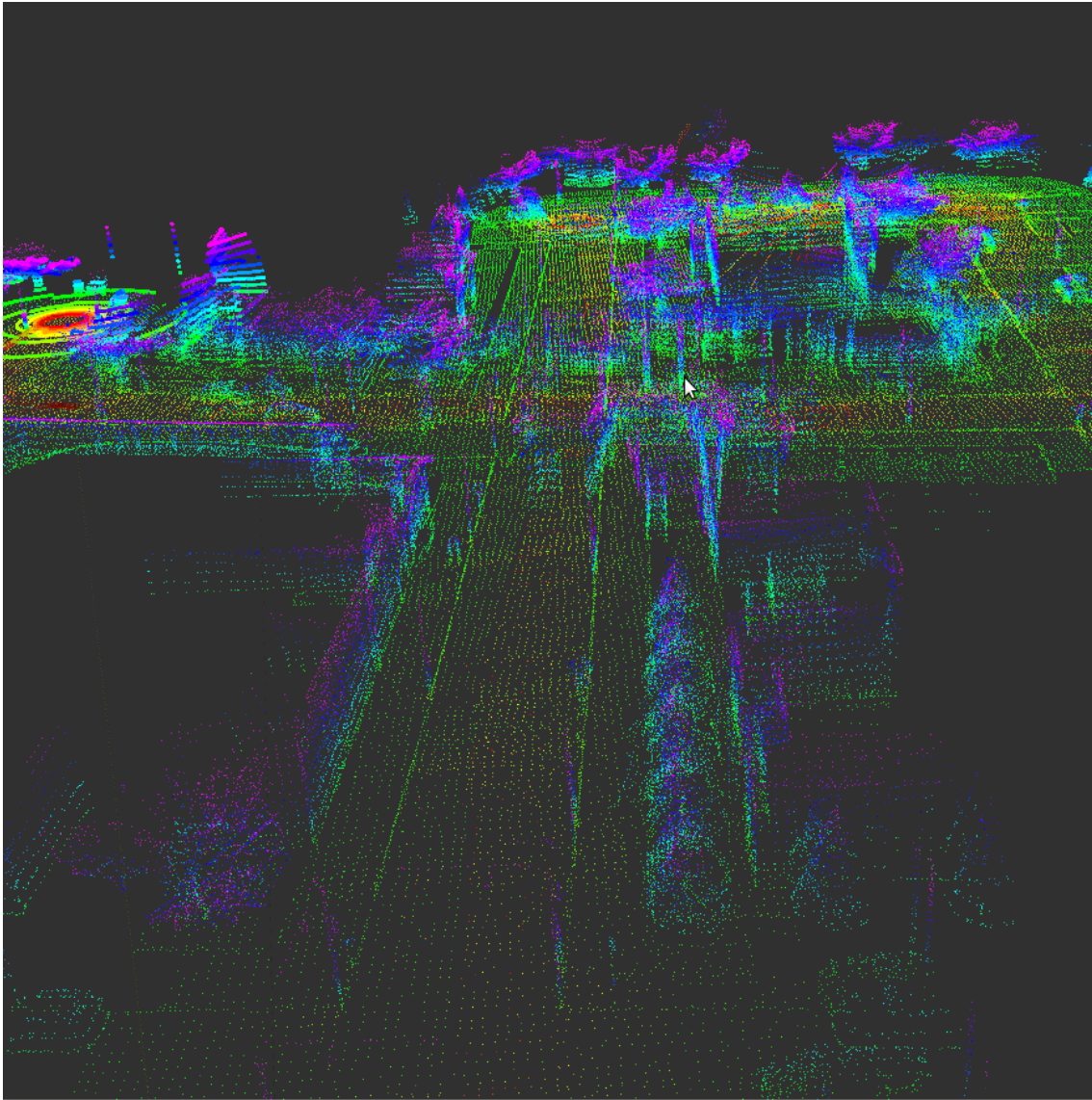


Figure 4.18: Result of 3D lase map generated by Loam algorithm.

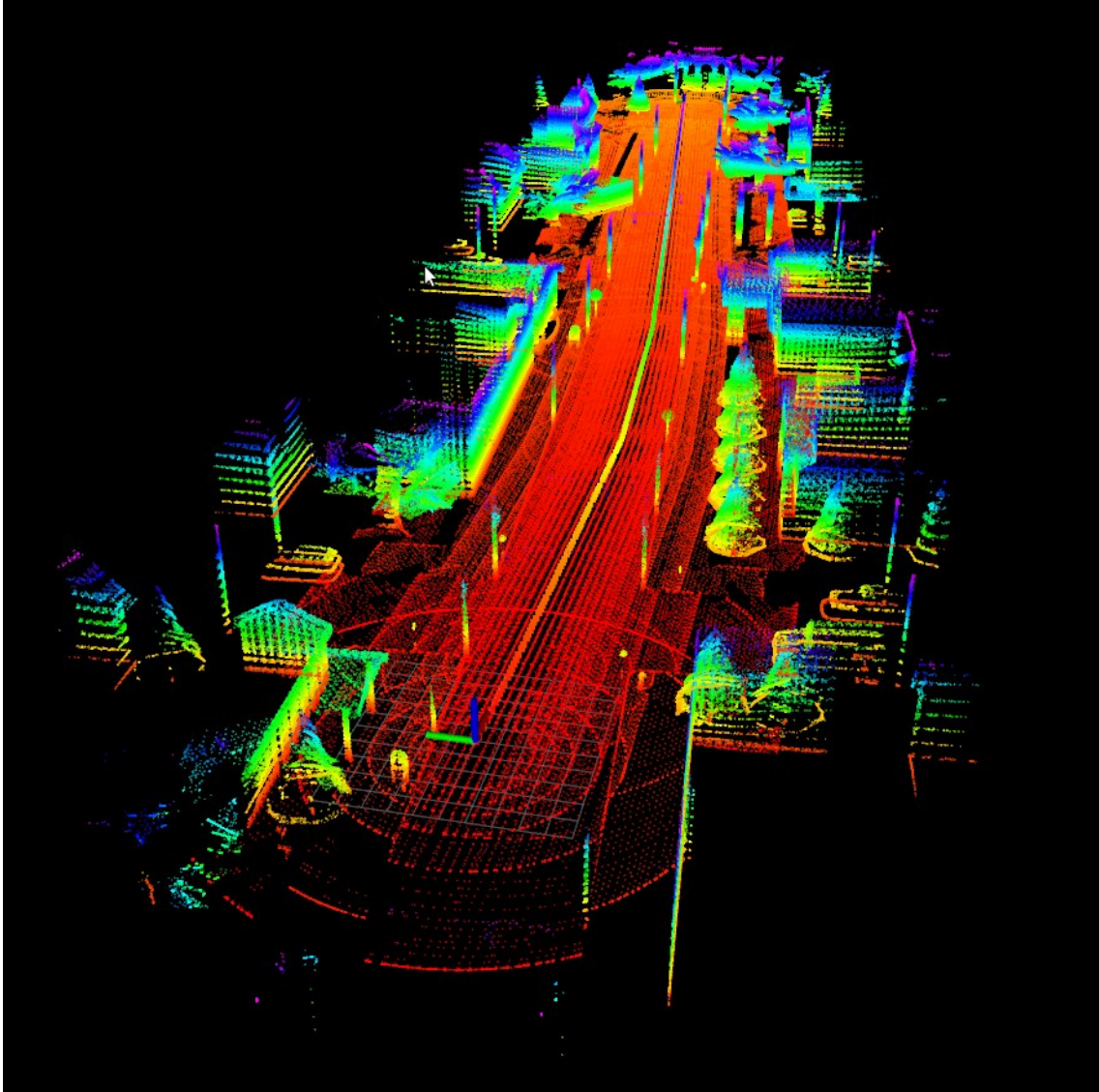


Figure 4.19: Result of 3D lase map generated by Lego-loam algorithm.

CHAPTER 5

CONCLUSION AND SCOPE

In this thesis, an autonomous driving robot was proposed and built based on a Segway self-balancing scooter. The robot was designed under the principle of modern intelligent and autonomous robotics, which consists three main body frames including perception, decision making, and action. This thesis mainly described the hardware/software structure, methodologies for different functions, and testing environment and results. The hardware system of the robot includes sensors, actuators, and processors. Several sensors including a 3D LiDAR, a monocular network camera, an IMU, and two wheel encoders were implemented, which completed the task of perceptual collection and provide data to software system. Two digital servos connected with a pendulum and steer rod formed the actuators, which realized the control over the robot physically. Two Arduino Uno microcontrollers and a Microsoft Surface fulfilled the data processing and communication ability. The software system of the robot realized the assignments including collecting data from the sensors and transporting to higher levels, processing data, several functionalities fulfillment, and send command back to actuators to control the robot. We proposed several functionalities developed under Python 3.6 including obstacle avoidance based on 2D local grid map and fuzzy logic, data fusion based on co-calibration, 2D SLAM based on Rao-Blackwellized particle filter, and path planning based on 2D global grid map. In this thesis, the test procedure and environment has been introduced, and we performed the testing of the robot under several scenarios. The test results illustrated that all the functions had achieved the expected effect.

Overall, the robot has the functional ability of proposed autonomous functions, which can be considered as an autonomous driving robot for some simple tasks. Besides, the modular hardware and software structure of this robot has been proven that it can be considered as a platform for developing further advanced autonomous driving or general algorithms.

5.1 Limitations

Although the robot proposed in this thesis has fulfilled all the functions, it still has some limitations in some respects. For example, we have not performed testing about 2D SLAM in an outdoor scenario, and the method chose for 2D SLAM is not suitable for big scale and open scenarios. Also, the path planning function can be added in a closed-loop control for retrieving the position of robot when following the path to destination.

Besides, the accuracy about some of the results was measured roughly due to the limitation of testing environment. Compared with other autonomous robots, the functionality of our robot is complete, which can be illustrated through CHAPTER 4 and demo videos. But the robot lacks consistent indicators to verify the performance. Hopefully, in the future the robot can be tested in a standardized field or scenario in order to verify the performance.

5.2 Real World vs. Simulation

The other thing worth mentioning is that the comparison between the real world and simulation. From a macro perspective, the whole software frame of the robot in real world was developed under Python 3.6, while the robot in simulation was developed under ROS (C++). Generally, the running speed of ROS was considered as higher than Python, which means the robot in simulation should has better performance in real-time, because the main language used of ROS is C++. However, the test results about map refreshing frequency illustrated that the algorithms developed under python has a running speed comparable to that of ROS. Although we did not make a detailed comparison under strict control of variables, it might be expected that developing some autonomous driving algorithms, especially focusing on mapping, localization, and movement control, can be realized through Python.

Besides, the robot in simulation has more functions (such as 3D SLAM) than in real world, and the performance of 2D SLAM and navigation functions in simulation presents better as well. The reason is that as a well-known open-source developing environment with high maturity especially in robotics, ROS has integrated many developers and algorithms with better performance.

5.3 Scope

As mentioned before, the robot proposed in this thesis can provide a platform for developing autonomous driving algorithms. One example is the project performed by the other graduate student Xiyuan Wang in our lab. Briefly introducing, he has trained an AI model for segmenting the drivable area from video streamed by camera in real-time. The connection between this project and our thesis is that the data for training model and testing was collected through the camera in our robot, and the segmented image output by model can be transformed into another matrix, as the input of obstacle avoidance to provide the perception of environment. This has been tested in campus as a prototype for controlling an autonomous robot based on robot vision.

Besides, one other feature about the design of the robot is modularization. The hardware and software system are all modular, which means the robot can be added or removed with certain functions without affecting other functions. This can be convenient that some hardware can be replaced by other hardware with lower cost, or some more advanced algorithms (e.g. 3D SLAM) can be developed based on this robot since it has the perception both for environment and itself, which meets the basic requirements in autonomous driving area.

The pace of developing more advanced algorithms can not be stopped. As an attempt of autonomous robotics and algorithms, we hope that this robot does its job. Also, we sincerely hope that more and more excellent algorithms will be developed based on this platform in the future to advance the field of autonomous driving.

REFERENCES

- [1] Heikkila, J. and Silvén, O., 1997, June. A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition* (pp. 1106-1112).
- [2] Reichardt, M., Föhst, T. and Berns, K., 2015. An overview on framework design for autonomous robots. *it-Information technology*, 57(2), pp.75-84.
- [3] Geiger, A., Lenz, P., Stiller, C. and Urtasun, R., 2013. Vision meets robotics: The kitti dataset. *The international journal of robotics research*, 32(11), pp.1231-1237.
- [4] Mohanty, P.K. and Parhi, D.R., 2013. Controlling the motion of an autonomous mobile robot using various techniques: a review. *Journal of advance mechanical engineering*, 1(1), pp.24-39.
- [5] Reichardt, M., Föhst, T. and Berns, K., 2015. An overview on framework design for autonomous robots. *it-Information technology*, 57(2), pp.75-84.
- [6] Muñoz, P., R-Moreno, M.D. and Barrero, D.F., 2016. Unified framework for path-planning and task-planning for autonomous robots. *Robotics and autonomous systems*, 82, pp.1-14.
- [7] Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T. and Jurišica, L., 2014. Path planning with modified a star algorithm for a mobile robot. *Procedia engineering*, 96, pp.59-69.
- [8] Faisal, A., Kamruzzaman, M., Yigitcanlar, T. and Currie, G., 2019. Understanding autonomous vehicles. *Journal of transport and land use*, 12(1), pp.45-72.
- [9] Royo, S. and Ballesta-Garcia, M., 2019. An overview of lidar imaging systems for autonomous vehicles. *Applied sciences*, 9(19), p.4093.
- [10] Sharma, K. and Doriya, R., 2020. Path planning for robots: An elucidating draft. *International journal of intelligent robotics and applications*, 4, pp.294-307.
- [11] Chhotray, A., Pradhan, M.K., Pandey, K.K. and Parhi, D.R., 2016. Kinematic analysis of a two-wheeled self-balancing mobile robot. In *Proceedings of the international conference on signal, networks, computing, and systems* (pp. 87-93). New Delhi, India: Springer.

- [12] Grisetti, G., Stachniss, C. and Burgard, W., 2007. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on robotics*, 23(1), pp.34-46.
- [13] Kim, Y., Kim, S.H. and Kwak, Y.K., 2005. Dynamic analysis of a nonholonomic two-wheeled inverted pendulum robot. *Journal of intelligent and robotic systems*, 44(1), pp.25-46.
- [14] Lee, K., Jung, C. and Chung, W., 2011. Accurate calibration of kinematic parameters for two wheel differential mobile robots. *Journal of mechanical science and technology*, 25(6), p.1603.
- [15] Myint, C. and Win, N.N., 2016. Position and velocity control for two-wheel differential drive mobile robot. *International journal of science engineering and technology research (IJSETR)*, 5(9), pp.2849-2855.
- [16] Meng, X., Wang, H. and Liu, B., 2017. A robust vehicle localization approach based on gnss/imu/dmi/lidar sensor fusion for autonomous vehicles. *Sensors*, 17(9), p.2140.
- [17] Kamil, F., Tang, S., Khaksar, W., Zulkifli, N. and Ahmad, S.A., 2015. A review on motion planning and obstacle avoidance approaches in dynamic environments. *Advances in robotics & automation*, 4(2), pp.134-142.
- [18] Ding, Y., Liu, J., Ye, J., Xiang, W., Wu, H.C. and Busch, C., 2020, October. 3D LiDAR and Color Camera Data Fusion. In *2020 IEEE International symposium on broadband multimedia systems and broadcasting (BMSB)*, pp. 1-4.
- [19] Sugiura, K. and Matsutani, H., 2020. An FPGA Acceleration and Optimization Techniques for 2D LiDAR SLAM Algorithm. *arXiv preprint arXiv:2006.01050*.
- [20] Cui, J., Niu, J., Ouyang, Z., He, Y. and Liu, D., 2020. ACSC: Automatic Calibration for Non-repetitive Scanning Solid-State LiDAR and Camera Systems. *arXiv preprint arXiv:2011.08516*.
- [21] Ben-Ari, M. and Mondada, F., 2018. Robots and their applications. In *Elements of robotics* , pp. 1-20. Cham, Switzerland: Springer.
- [22] Zohaib, M., Pasha, M., Riaz, R.A., Javaid, N., Ilahi, M. and Khan, R.D., 2013. Control strategies for mobile robot with obstacle avoidance. *arXiv preprint arXiv:1306.1144*.
- [23] Zhao, X., Sun, P., Xu, Z., Min, H. and Yu, H., 2020. Fusion of 3D LIDAR and camera data for object detection in autonomous vehicle applications. *IEEE Sensors journal*, 20(9), pp.4901-4913.
- [24] Polvara, R., Sharma, S., Wan, J., Manning, A. and Sutton, R., 2018. Obstacle avoidance approaches for autonomous navigation of unmanned surface vehicles. *The Journal of navigation*, 71(1), pp.241-256.

- [25] Santos, J.M., Portugal, D. and Rocha, R.P., 2013, October. An evaluation of 2D SLAM techniques available in robot operating system. In *2013 IEEE International symposium on safety, security, and rescue robotics (SSRR)*, pp. 1-6.
- [26] Kocić, J., Jovičić, N. and Drndarević, V., 2018, November. Sensors and sensor fusion in autonomous vehicles. In *2018 26th Telecommunications forum (TELFOR)*, pp. 420-425. IEEE.
- [27] Huang, W.H., Fajen, B.R., Fink, J.R. and Warren, W.H., 2006. Visual navigation and obstacle avoidance using a steering potential function. *Robotics and autonomous systems*, 54(4), pp. 288-299.
- [28] Oroko, J. and Ikua, B., 2012. Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: a review. *Sustainable research and innovation proceedings*, 4.
- [29] Jin, J. and Chung, W., 2019. Obstacle avoidance of two-wheel differential robots considering the uncertainty of robot motion on the basis of encoder odometry information. *Sensors*, 19(2), p. 289.
- [30] Royo, S. and Ballesta-Garcia, M., 2019. An overview of lidar imaging systems for autonomous vehicles. *Applied sciences*, 9(19), p. 4093.
- [31] Wang, W., Sakurada, K. and Kawaguchi, N., 2017. Reflectance intensity assisted automatic and accurate extrinsic calibration of 3d lidar and panoramic camera using a printed chessboard. *Remote sensing*, 9(8), p. 851.
- [32] Zhao, G., Xiao, X., Yuan, J. and Ng, G.W., 2014. Fusion of 3D-LIDAR and camera data for scene parsing. *Journal of visual communication and image representation*, 25(1), pp. 165-183.
- [33] Li, J., Zhang, X., Li, J., Liu, Y. and Wang, J., 2020. Building and optimization of 3D semantic map based on Lidar and camera fusion. *Neurocomputing*, 409, pp. 394-407.
- [34] Tokekar, P., Vander Hook, J., Mulla, D. and Isler, V., 2016. Sensor planning for a symbiotic UAV and UGV system for precision agriculture. *IEEE Transactions on robotics*, 32(6), pp. 1498-1511.
- [35] Siegwart, R., Nourbakhsh, I.R. and Scaramuzza, D., 2011. *Introduction to autonomous mobile robots*. Cambridge, MA: MIT press.
- [36] He, B., Zhang, S., Yan, T., Zhang, T., Liang, Y. and Zhang, H., 2011. A novel combined SLAM based on RBPF-SLAM and EIF-SLAM for mobile system sensing in a large scale environment. *Sensors*, 11(11), pp. 10197-10219.

- [37] Wei, K. and Ren, B., 2018. A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm. *Sensors*, 18(2), p. 571.
- [38] Muñoz, P., R-Moreno, M.D. and Barrero, D.F., 2016. Unified framework for path-planning and task-planning for autonomous robots. *Robotics and autonomous systems*, 82, pp.1-14.
- [39] Hendrich, N., Bistry, H. and Zhang, J., 2015. Architecture and software design for a service robot in an elderly-care scenario. *Engineering*, 1(1), pp. 027-035.
- [40] Limsoonthrakul, S., Dailey, M.N., Srisupundit, M., Tongphu, S. and Parnichkun, M., 2009, February. A modular system architecture for autonomous robots based on blackboard and publish-subscribe mechanisms. In *2008 IEEE International conference on robotics and biomimetics* pp. 633-638.
- [41] Lee, H. and Jeong, J., 2021. Mobile Robot Path Optimization Technique Based on Reinforcement Learning Algorithm in Warehouse Environment. *Applied sciences*, 11(3), p. 1209.
- [42] Sariff, N. and Buniyamin, N., 2006, June. An overview of autonomous mobile robot path planning algorithms. In *2006 4th student conference on research and development*, pp. 183-188.
- [43] Canedo-Rodríguez, A., Alvarez-Santos, V., Regueiro, C.V., Iglesias, R., Barro, S. and Presedo, J., 2016. Particle filter robot localisation through robust fusion of laser, WiFi, compass, and a network of external cameras. *Information fusion*, 27, pp. 170-188.
- [44] Martinez, A. and Fernández, E., 2013. *Learning ROS for robotics programming*. Birmingham, UK: Packt Publishing Ltd.
- [45] Quenzel, J., Nieuwenhuisen, M., Droschel, D., Beul, M., Houben, S. and Behnke, S., 2019. Autonomous MAV-based indoor chimney inspection with 3D laser localization and textured surface reconstruction. *Journal of intelligent & robotic systems*, 93(1-2), pp. 317-335.
- [46] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. and Leonard, J.J., 2016. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6), pp. 1309-1332.
- [47] Zhang, J. and Singh, S., 2014, July. LOAM: Lidar Odometry and Mapping in Real-time. In *Robotics: Science and Systems X*, 2(9), pp. 109-111.
- [48] Arulampalam, M.S., Maskell, S., Gordon, N. and Clapp, T., 2002. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on signal processing*, 50(2), pp. 174-188.

- [49] Antonelli, G., Chiaverini, S. and Fusco, G., 2005. A calibration method for odometry of mobile robots based on the least-squares technique: theory and experimental validation. *IEEE Transactions on robotics*, 21(5), pp. 994-1004.
- [50] Tomasi, D.L. and Todt, E., 2017, November. Rotational odometry calibration for differential robot platforms. In *2017 Latin American robotics symposium (LARS) and 2017 Brazilian symposium on robotics (SBR)* pp. 1-6.
- [51] Censi, A., Franchi, A., Marchionni, L. and Oriolo, G., 2013. Simultaneous calibration of odometry and sensor parameters for mobile robots. *IEEE Transactions on robotics*, 29(2), pp. 475-492.
- [52] Thrun, S., 2002. Probabilistic robotics. *Communications of the ACM*, 45(3), pp. 52-57.