Xi'an Jiaotong-Liverpool University
西交利物浦大學

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EEE311 Final Year Project

# A Visual RGB-D SLAM aplication in AGV with omni wheels

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Engineering

Student Name : Zhenghang Zhong
Student ID : 1508686
Supervisor : Dr. Limin Yu
Assessor : Dr. Leach Mark

**Abstract**

Nowadays, automated guided vehicles (AGVs) are capable of accomplishing complicated tasks in addition to simple and repetitive one. With the development of micro processors and different types of perception sensors, there are many solution plans according to requirements of cost, precision and speed. For this project, a kinect depth sensor is used for perception on a small AGV with omni wheels, to realize the function of building indoor map and localization itself, the technology of PID controller, kalman filter and Robot Operating System (ROS) as well as related packages are utilized. The AGV's motion analysis, odometry and mapping performance is discussed with comparably good precision. The validness of this project is proved in the end.

Keywords: AGV, kinect, visual RGB-D,SLAM, omni wheels, Navigation, ROS, Automation, RTAB-Map

# Acknowledgement

I would like to thank Dr. Limin Yu and Dr. Leach Mark for their guidance and support throughout this final year project. Their expertness and supports helped me stay the course throughout the evolution of my thesis topic. Additionally, I would like to thank Dr. Kyeong Soo Kim who have taught me a lot during the past two Summer Undergraduate Research Fellowship (SURF) projects, which do influence my academic career. This paper was released on 25 May 2019.

# Contents

# List of Abbreviations

| | |
|---|---|
| AGV | Automated Guided Vehicle |
| SLAM | Simultaneous Localization and Mapping |
| ROS | Robot Operating System |
| PWM | Pulse Width Modulation |
| MCU | Micro Controller Unit |
| RTAB-Map | Real-Time Appearance-Based Mapping |

# Chapter 1

# Background

## 1.1 Introduction, Aims and Objectives

### 1.1.1 Introduction and Literature Review

Automated guided vehicles (AGVs) have existed since the 1950s, although they have undergone noticeable improvements in engineering since then. When Barrett Electronics first introduced the AGVs, they were guided by wires on the floor and handled uncomplicated traction tasks. Until recently, AGVs were primarily used in industrial manufacturing environments to carry out repetitive tasks in specific environments with particular constants, as they provide a productive and economical and practical way to transport objects. The complexity of today's AGV comes from the same simple machine that moves goods from a defined location to another defined location, but with more requirements of precision and effectiveness.
Nowadays AGVs are portable robots that navigate based on mounted sensors to percept surrounding environment, i.e., like using techniques such as laser scanning and magnetic tape, spots, natural feature (camera or laser), and inductive wires [1] on the floor, and avoid obstacles while carrying goods to rightful destination. The application areas for AGVs are enlarging which indicates that the need for new technology and surpassing performance is boosting, and AGVs reduce the amount of manual work required.
Indoor localization is an important topic in present research since it can be used in a wide variety of systems, such as AGVs. There exist solutions which can be considered infrastructure-free, for example those using existing Wi-Fi signal or geomagnetic field [2], [3]. For localization. Other solutions, using techniques such as Ultra Wideband (UWB) radio or RFID, require beacons or special infrastructure but are generally more accurate.
Technically, these vehicles must obtain the ability to localize themselves reliably in order to work safely and efficiently, which makes localization precision a major feature for AGV systems. The most frequent method to achieve it is using auxiliary infrastructure in the existing environment, i.e., the usage of color tape, magnetic tape and wires. However, these solutions are not flexible and setting up of them requires extra manual work and is time-wasting if environment changes. Therefore, flexible map-based positioning method with map being updated frequently is becoming significant for future development. With the implantation of a technique called Simultaneous Localization and Mapping (SLAM), it is conceivable to automate the process to percept the environment and significantly reduce the cost and time of mapping process [4].

SLAM have been a main research topic in mobile robotics [5]. SLAM algorithms is capable of running either online or offline. An online SLAM algorithm works by computing a robot pose estimate at running time and mapping an unknown environment while simultaneously finds and keeps track of the operator's location within this map. Offline algorithm works when computational resource is limited, and it computes based on previously recorded sensor data. SLAM has grown to a state in which two-dimensional maps has been frequently used in robotic research but have not been widely accepted in industrial systems [6]. With the increasing popularity of the deployment of AGV systems and 3-D range sensors in more complex scenarios, the development of accurate and reliable 3-D mapping and positioning algorithms is promptly turning into a momentous industry-related research subject.
A complete AGV is a complicated robot system which consists of both hardware and software. With the implementation of various sensors, the data processing step requires much of computational and memory resources, and the system become hard for development and debug. Fortunately, Robot Operating System (ROS) [7] provides a simple way of constructing such a system.
As an open source framework that is powerful when building software for robot system, ROS contains libraries and tools for simplifying the process of building and connecting complex, but modular distributed robot systems. When functions accumulates to some degree the code would become hard to maintain, then a reasonable distribution of code module would simplify development and testing of new approaches and features.
To percept surrounding environment, sensors are most important hardware. There are many types of sensors defined based on work principle, precision and cost. Common sensors are magnetic sensor for detecting magnetic tape which is easy for implementation but hard to handle various tasks; others are lidars as laser sensor, inertial sensor like IMU and visual sensors like monocular camera, stereo camera and depth camera.
Kinect (shown in Fig 1.1) is a RGB-D camera which is a kind of depth combing camera sensors. Kinect consists of several sensors including a RGB sensor, a infrared (IR) depth sensor, multi-array microphones and an accelerometer. To enable the AGV car to navigate freely in indoor environment with high localization precision and carry out tasks, an AGV system which combines SLAM, ROS, Kinect and presented.


Figure 1.1: The structure of depth camera named Kinect

One up-to-date issue that remains to be solved is the requirement of a mature AGV system that combines camera and other sensor data for high-precision localization in industrial manufacturing environment. Usually hardware with high computation ability is needed for graph-based SLAM AGV system, which is usually limited by the budget and makes it impossible to apply to small portable devices.

### 1.1.2 Aims

This project aims to deploy RGB-D sensor (kinect) in AGV system to build a 3-D map of lab environment and based on it to realize appearance-based localization with the help of SLAM, ROS and RTAB-Map SLAM library. To achieve this, several steps are divided. Control of 3-axis omni wheels AGV car with accurate speed and rotate regulation. With thorough understanding of SLAM problem. Based on the AGV system to build an 3-D indoor point cloud map, with the help of Kinect depth sensor. Finally, the AGV car should to localize itself in build map and could navigate and avoid obstacles.

### 1.1.3 Project budget

The overall cost of this AGV system is around 1700 RMB, as shown in Figure. 1.2. With 3-layer AGV frame cost 400 RMB (including motors with mounted encoder and omni wheels). Components are listed with the order of different layers below.

1. First layer (bottom layer): three omni wheels with encoders, 2 L298N motor drivers, one 12V battery.
2. Second layer (middle layer): three Arduino Unos, peg board, two voltage converters, one 12V battery.
3. Third layer (top layer): Raspberry Pi, Kinect V1.

For remote client, the PC is responsible for processing RGB-D data and central controller.

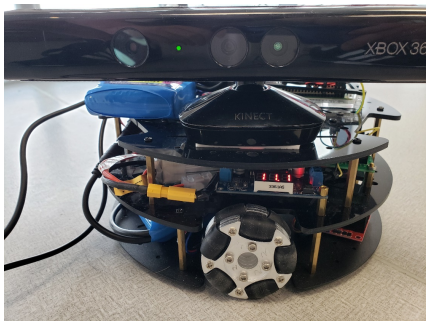


Figure 1.2: The graphical view of AGV

## 1.2 Industrial Relevance

In terms of manipulation, one of the main motivations for incorporating vision in navigation is the demand for increased flexibility of robotic systems [8], which is lacked in the traditional navigation systems based on magnetic tape, inductive wires and spots. Although with a foreseeable market and future, flexible technology like visual navigation is still not mature [9], due to its high technical difficulties (high computation burden, noise in images and complex algorithms) and is susceptible to various lights [10], that is to say, one well-worked indoor navigation system may fail in outdoor environment.

# Chapter 2

# Main Body

This final year project is a rather complicated one which combines both the hardware and software parts, then the methodology section will be discussed in terms of hardware, software and mathematical parts.

## 2.1 Methodology - Hardware

### 2.1.1 Perception sensor: camera choosing

When considering the camera sensors, we have several options, monocular camera, stereo camera and RGB-D camera. Among the three sensors, monocular camera is the cheapest one, but it has scale-uncertainty problem which results from incapableness of obtaining depth information from only one image, to solve this problem we usually move camera and recover object's structure from a series of images, which requires huge amount of computation resource and sometimes need to be down offline.
Rich visual information is available from passive low-cost visual sensors which LiDAR lacks, but the trade-off is the requirement of more computational resource and more sophisticated algorithms to process the images and extract the necessary information. One of the major reasons Monocular Cameras are used in SLAM problems is the hardware needed to implement it is much simpler, leading to the systems that are cheaper and physically smaller. Suddenly, SLAM is accessible on mobile phones without the need for additional hardware. A weakness, however, is that the algorithms and software needed for monocular SLAM are much more complex because of the lack of direct depth information from a 2D image. Nevertheless, by integrating measurements in the chain of frames over time using a triangulation method, it is likely to jointly recover the shape of the map (and the motion of the camera under the assumption that camera is not still). However, since the depths of points are not detected directly, the estimated camera positions and point on image after processing are zoomed with unknown scale factor with real positions. The map therefore becomes a dimensionless map without any real-word meaning attached to one map unit. In order to address the scale availability issue, there are some alternatives other than stereo cameras. Real metric scale can be introduced by an external scale reference in the form of a pre-specified object or a set with known size that can be recognized during mapping.
For this project a kinect RGB-D Cameras is deployed that generate 3D points could through time-of-flight technology which can provide depth information directly. For a time-of-flight

camera, the camera obtains depth information by measuring the time of flight of a light signal between the camera and objects. Moreover, the central monocular camera work together with depth camera to eliminate the unknown scale factor problem.
RGB-D cameras were found to have various overarching limitations as they dont provide reliable range data for semi-transparent or highly reflective surfaces, and also have a limited effective range. Therefore, it is mainly used indoor with abundant features.

### 2.1.2 Calibration

The Fig 2.1 shows the inner structure of Kinect v1, the source of obtaining the RGB and depth images.


Figure 2.1: The inner structure of Kinect v1 [11]

Tiny hardware difference (i.e. focal length and optical center location) of cameras are unavoidable, and it doesn't result in big difference in captured images, but the tiny difference may result in large variance when we need to implement visual based high-precision algorithm like odometry in SLAM solution. Therefore, it is necessary to obtain the factors for calibration. The most important part of utilizing Kinect camera is the conversion from the 2D images to 3D points could, the C++ realization code is listed in Appendix.
Camera calibration is the process of finding the true parameters of the camera that took photographs. Some of these parameters are focal length, format size, principal point, and lens distortion. As stated before, calibration of the camera is essential in order to get the intrinsic parameters for better perception. For RGB camera and IR camera they need to be calibrated separately. With the help of ROS package 'camera_calibration', a camera calibrator is used to calibrate the RGB camera with a raw image over ROS.
In order to get a good calibration, the checkerboard need to be moved around in the camera frame to satisfy various spatial position and angle requirements, like the position on camera's left, right, top and bottom of field of view with right and tilted angle, as well as filling the whole field of view. With the process of calibration advance, the process bar of title 'X', 'Y', 'Size' and 'Skew' would increase to certain value and become green, all green means the calibration success.

A screen shoot when calibrate RGB camera is shown in Figure. 2.2. The background scene is the lab in IR building 613, the checkerboard has size of $8 \times 6$, which indicating that for inner black squares the total corners have size $8 \times 6$. The colored lines indicate corners got captured.

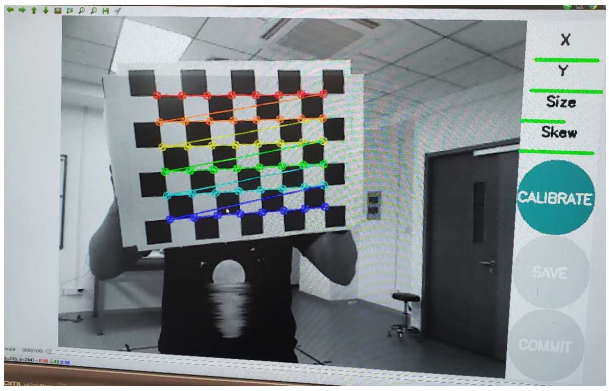



Figure 2.2: Calibration of RGB camera

There is a trick for calibrating IR camera, which is covering the projector for calibration, as the noise caused by projector is severe that fails the calibration, as shown in Figure. 2.3. After covering projector, the calibration success and the screen shoot is shown in Figure. 2.4, as we can see, without the projector the image frame is dark even turning the indoor lights on.

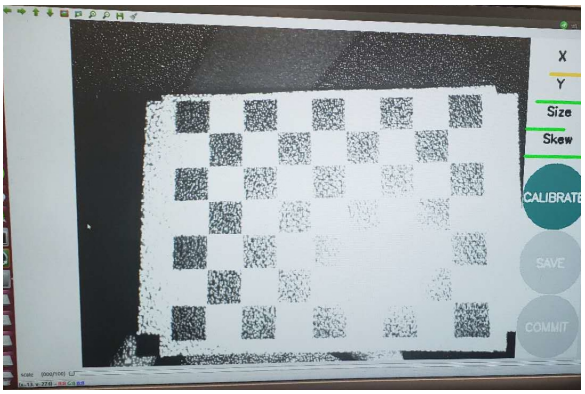



Figure 2.3: Calibration of IR camera with projector work.

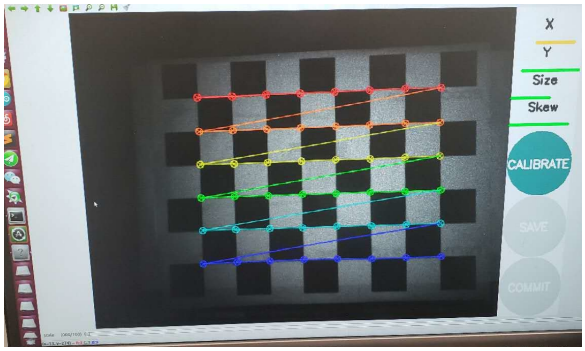



Figure 2.4: Calibration of IR camera with projector covered.

After the calibration,

### 2.1.3 omni wheels

For ordinary cars (Figure. 2.5), it's not easy to change its moving direction while keeping in a small range of eras, and the demanded space increase with the size of AGV, which is limiting when accomplishing tasks in narrow field. Therefore, the huge advantage of omni wheels (Figure. 2.6) attract much attention. Omni wheels have small passive rollers around the circumference which are perpendicular to the turning direction, thus they could be driven with full force while be able to slide laterally with great ease, and the right view of AGV's chassis with omni wheels mounted are shown in Fig 2.7.

Figure 2.5: Four-wheel ordinary car.

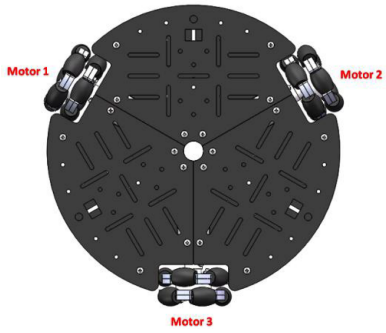



Figure 2.6: 3-axis omni-wheel car: top view

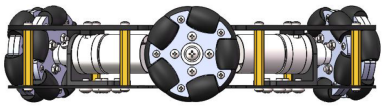


Figure 2.7: 3-axis omni-wheel car: right view

For wire connection of power supply and PWM control of three gear motors, the diagram sketch is shown in Figure. 2.8, the gear motor is driven directly by the L298N motor driving module. Although the encoders mounted on motor are not shown due to sketching software limitation, it has simply 4 wires need to be connected, which are +5V, GND, ENA and ENB, where ENA and ENB are two light sensors for angular speed detection.

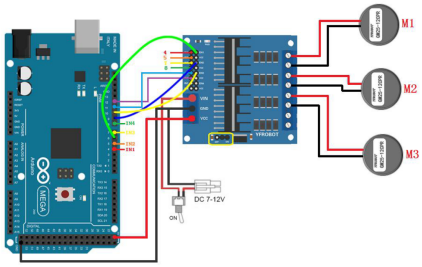



Figure 2.8: The wire connection of gear motors, L298N and Arduino

It may be hard to comprehend the way it moves at first. For illustration, 4 basic moving types of direct motion analysis is shown in Figure. 2.9. The simplest motion is rotating around center of mass (three wheels with same speed and direction) and go forward and backward (two rear wheels move in different directions). Basically, with different combination of three wheels in terms of speed and direction, this special AGV could move toward every direction instantly. Next section mathematical analysis will be given for free motion.
To analyze the motion of omni wheels under ideal situation, three assumptions are made.

1. The omni wheels obtain enough friction force and do not slip on ground
2. The car is symmetrical and the position of center-of-gravity lies at physical center
3. Three wheels go across one center point

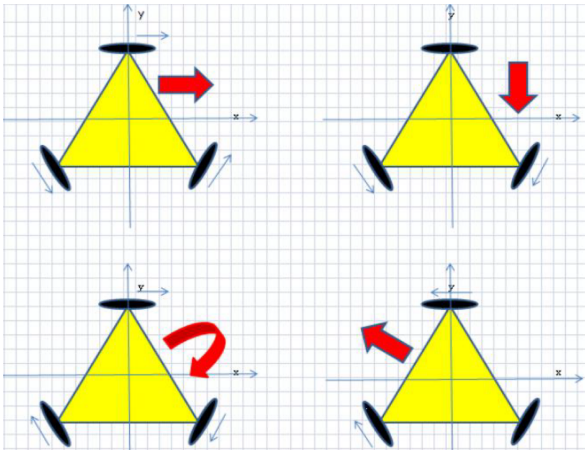


Figure 2.9: Motion analysis

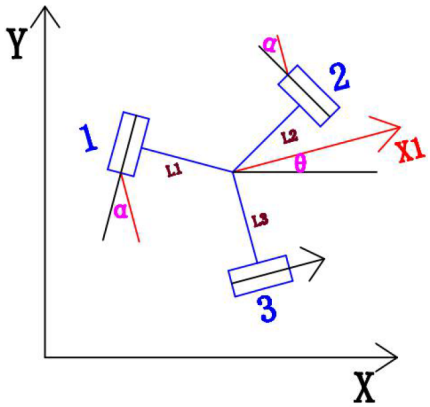



Figure 2.10: The motion force analysis, with direction x1

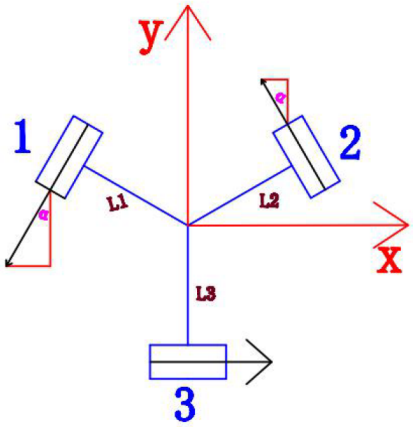



Figure 2.11: The motion force analysis, in AGV's frame

To obtain the transform between world's frame and wheels' frame, which is the ultimate goal, the mathematical model of AGV's frame against world's frame is established as shown in Figure. 2.10, where **X1** represent the X-axis of AGV's frame, which form an angle of $\theta$ against X-axis of world's frame. Therefore, the transformation could be established as shown in Equation. 2.1.

$$\begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} \tag{2.1}$$

After getting the velocity in AGV's frame, it could be easy to figure out every wheel's velocity. Assuming the angle between wheel's moving direction and Y-axis is $\alpha$, and the length between the center of AGV and that of wheel is L, in unit of cm. Therefore, the transform could be expressed as shown in Equation. 2.2. Combine Equation. 2.1 and Equation. 2.2 together comes Equation. 2.3.

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & L \\ 1 & 0 & L \\ -\cos\alpha & \sin\alpha & L \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ w \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & L \\ 1 & 0 & L \\ -\cos\alpha & \sin\alpha & L \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} \tag{2.3}$$

After simplifying and combination, we get Equation. 2.4.

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} -sin(\alpha+\theta) & -cos(\alpha+\theta) & L \\ -sin(\alpha+\theta) & cos(\alpha+\theta) & L \\ cos\theta & sin\theta & L \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} \tag{2.4}$$

Although above transform realize the process from world frame to wheel's frame, in real application we need to obtain real speed in world's frame as feedback for further decision and analysis, like in PID control and odometry. Therefore, we need to reverse above process to establish a transform from wheel velocity to that in world frame. Fortunately, with the help of inverse matrix, as shown in Equation. 2.5. Figure. 2.11 shows the wheels are arranged in symmetrical manner $120^\circ$ apart. Therefore, $\alpha = 60^\circ$, and put this value in Equation. 2.5. Then the final inverse transform is like Equation. 2.6 shows.

$$\begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \cos\alpha & -\sin\alpha & L \\ 1 & 0 & L \\ -\cos\alpha & \sin\alpha & L \end{bmatrix}^{-1} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{2.5}$$

$$\begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix} = \begin{bmatrix} -0.33333 & -0.66667 & 0 \\ -0.33333 & 0 & -0.57735 \\ 0.03367 & 0.03367 & 0.03367 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{2.6}$$

These mathematical formulas are implemented directly, and C code is displayed in Appendix part.

## 2.2 Speed detection- Photoelectric encoder

Speed feedback is significant in PID controlling and odometry, and the hardware component called encoder (as shown in Figure. 2.12) provide a way to detect speed. Encoder is mounted on the gear motor which has code wheel (like Figure. 2.13) inside with optical grating, and it will rotate along with the motor. There are two light sensors inside of encoder which will sense the change of optical grating and generating two series of square waves with speed proportional to that of code wheel.

Figure 2.12: Encoder with gear motor

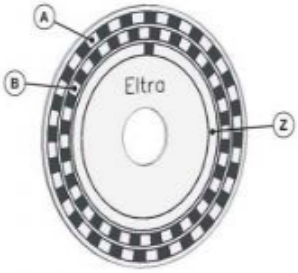



Figure 2.13: Code wheel [12]

Refer to Figure. 2.14, light sensors A and B produce two series of square waves which indicate forward, backward and noise for different combination. To separate them out and prevent the distortion of noise, Algorithm. 1 of speed calculation is designed by me, to realize the idea performance as the Counter wave shows.

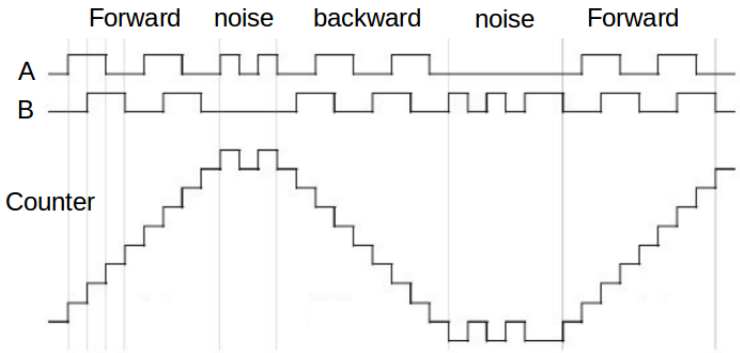



Figure 2.14: Encoder waves and counter waves

**Algorithm 1:** Speed calculation — pseudo code realization

```
Result: Speed = Duration / time
StateA← digitalReadA;
if LastStateA = LOW and StateA = HIGH then
    stateB ← digitalReadB;
    if stateB = LOW and Direction=True then
        Direction←False;
    end
    if stateB = HIGH and Direction=False then
        Direction←True;
    end
end
LastStateA ← StateA
if Direction=False then
    Duration++;
else
    Duration- - ;
end
```

To simply illustrate the idea behind this algorithm, firstly we need to pay attention to the characteristic of forward and backward, pulse in A always appear first in forward direction while pulse in B shows that feature in backward direction, while other situations could be included as noise. Therefore, the algorithm could be going in five steps. Firstly, read sensor A's value and trigger IF statement when a Low-to-High pulse is detected. Then within IF statement, read sensor B's value, if the value is LOW (which means motor is now rotate forward) and variable "Direction" is True (which means last state's motor go backward), then reverse "Direction" value, otherwise do not change it; it is similar when sensor B's value is HIGH. Next, add or minus one to variable "Duration" according the value of "Direction", where "Duration" means the detected valuable pulses with directions. Then refresh last state A's value by new one. Finally, calculate the speed according to the number of "Durations" in a short period of time and known radius of wheels. Another thing need to mention is that for one rotation of this gear motor encoder will produce around 53735 pulses by experiments, therefore a high speed of wheels will result in huge computational burden for MCU. The C code for implementing this algorithm is shown in Appendix.

### 2.2.1 Raspberry Pi

Raspberry Pi is a series of small single-board computers which also integrates the basic communication module like blue-tooth and Wi-Fi. The Operation System (OS) of this computer is Linux, while Raspbian and Ubuntu MATE is the famous one. Although with a limited calculation speed, it could deal with basic robots programs with accessible workload, while in a SLAM solution it gets hard to run the algorithm online with only Raspberry Pi. However, for a small size of AGV in this program due to funds limitation, placing a powerful laptop on the AGV for calculation and processing, like Turtlebot does, is not wise, then the Raspberry Pi is considered as a interchange station of information between sensors and laptop with the help of Wi-Fi module, as well as receiving order from CPU and control the motion of AGV car indirectly. The used one is Raspberry Pi 3B, as shown in Fig 2.15.

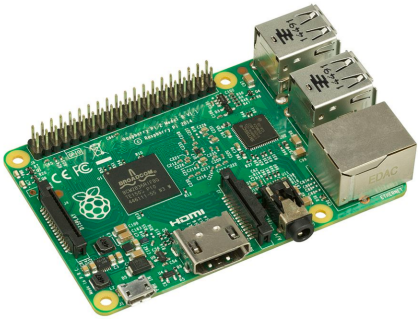


Figure 2.15: The figure of Raspberry Pi 3B

Figure 2.16: The figure of Arduino Mega 2560

### 2.2.2 Arduino boards

The Arduino is an open-source hardware which is famous for easy prototyping and the various microprocessors and controllers. The main usage of Arduino is receiving instructions from Raspberry Pi and information from wheel encoders, and according to them to control the speed of omni wheels. As a kind of MCU with basic functions comparing to Raspberry Pi as small

computer, Arduino boards could satisfy the requirements for controlling motors. At first only one Mega 2560 board is used which has enough I/O pins (54 pins) and a figure of it is shown in Fig 2.16, but latter it collapses when dealing with three encoders' input. Finally, it was replaced by three Arduino UNOs which together offer powerful computational resources.

### 2.2.3 Basic Hardware Structure and circuit connection

With a hierarchy architecture, the hardware connection is complicated and a basic structure is shown in Fig 2.17, the diagrammatic sketch is shown in Figure. 2.18.

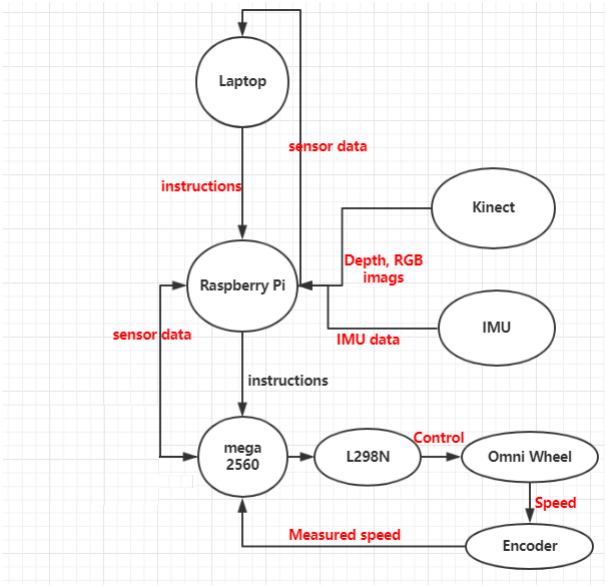



Figure 2.17: The basic hardware structure of the system

For circuit connection, it can be simply divided into two parts, server part and client part, where the left part with PC as server part for major image processing and computation, and the right part work as client part for controlling the AGV. Two parts communicate through Wi-Fi signal.

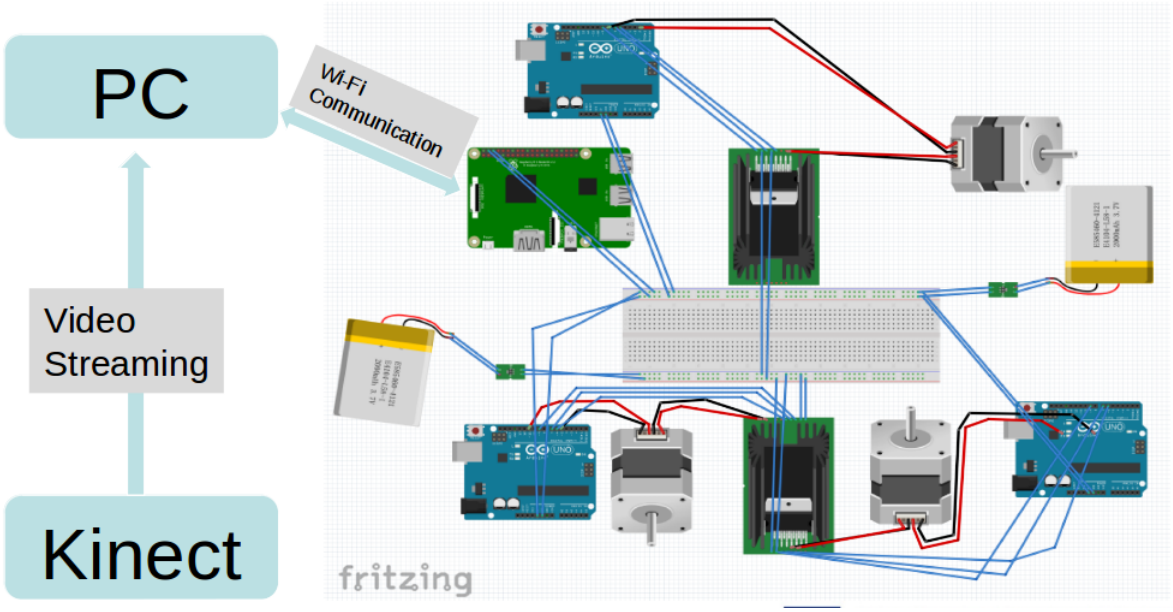



Figure 2.18: Circuit connection

## 2.3 Methodology - Software

### 2.3.1 SLAM

SLAM is a standard abbreviation of Simultaneous Localization and Mapping. It aims to estimate the pose of a robot and the map of the environment at the same time. For localization, it means inferring location giving a map; mapping means inferring a map given locations. Therefore, SLAM means learning a map and locating robot simultaneously and SLAM could be considered as a chicken-or-egg problem.

### 2.3.2 Version of relevant application and libraries

- Ubuntu: 16.04.5 LTS (PC)
- Ubuntu MATE: 16.04 (Raspberry Pi)
- Python: 3.5.1.3
- c++11
- ROS: Kinetic Kame
- opencv: 3.2.0
- eigen: 3.2.10
- g2o: 3.1
- OpenNL: 1.5.4.0

- rtabmap-ros: latest version

### 2.3.3 ROS

Robot Operating System (ROS) is compatible with Ubuntu and Ubuntu is a famous Linux distribution with visual operating desktop, it is friendly and supportive to open source applications which are the essential libraries for SLAM solution. ROS is not an OS but it provides services (as shown in Figure. 2.19) designed for computer cluster such as hardware abstraction, low-level device control and implementation of commonly used functionality. Although not essential in SLAM solution, ROS is always recommended as simplify the process of building up a robot system and maintenance. Node is the basic unity for manipulating, for example, Kinect is packaged as a node which sent out raw and rectified images, including RGB and depth figure; and odometry is also configured as a node which receive actual speed information from encoders and output odometry information. As every node works in separate field, the collapse of one node would not infect another and it's easy to find out which node need to be fixed and where the bug is, this feature would show huge advantages especially when the system is complicated. For the communication between nodes, "Messages" take the lead role. All the data is send out and received via various type of messages with different name called "Topics". There are other communication way like "Services", but for this project, the mainly method is using "Messages".

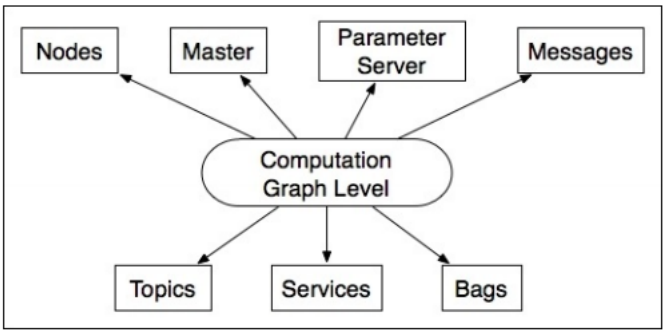



Figure 2.19: Ros structure [13]

## 2.4 Methodology - Mathematics

### 2.4.1 Quaternion for pose calculation

Rather than using Rotation Matrix or Euler Angles which suffer from the famous Gimbals problem of singular points. Take Eular Angles as example, it is normally used to describe the rotation and rigid body in three parameters (yaw-pitch-roll), which are three angles when the body revolving around the Z, Y and X axis respectively. However, the Gimbals problem happens when pitch angle becomes $+90^o$, the first and third rotation would actually around same axis, which result in the system losing one free of degree. Theoretically, we cannot find a rotation description method based on 3D vectors without singularity [14]. That is the reason why we finally using **Quaternion** to solve this problem. Quaternion is a kind of extended

complex number, which has one real part and three imaginary parts.

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{2.7}$$

Where i, k, k are three imaginary parts of Quaternion. Assuming a rotation process is around a unit vector $n = [n_x, n_y, n_z]^T$ with an angle of $\theta$. Then the form of rotated Quaternion is

$$q = [\cos\frac{\theta}{2}, n_x \sin\frac{\theta}{2}, n_y \sin\frac{\theta}{2}, n_z \sin\frac{\theta}{2}]^T \tag{2.8}$$

And conversely, the relating rotation axis and rotation angle could be calculated from Quaternion,

$$\theta = 2arc\cos q_0 \tag{2.9}$$

$$[n_x, n_y, n_z]^T = [q_1, q_2, q_3]^T / \sin\frac{\theta}{2} \tag{2.10}$$

And based on the Quaternion and the fixed global coordinate system, it is convenient to derive the motion vectors according to a same fixed coordinate system. Besides, the library **eigen** provides the tools of dealing with Eular Angle, Rotation Matrix and Quaternion.

### 2.4.2 PID controller

A basic PID speed controller is shown in Figure. 2.20, the input to PID controller is the difference between desired speed and speed feedback, and the cumulative output is a proper PWM value for adjusting speed of motors. With fine-tuning for three omni wheels, the PID parameters of three motors is shown in Table. 2.1. For more explanation of its work principle, the detailed five steps are listed below.

1. Input the desired motor velocity (in rpm);
2. Encoder return the motor velocity (in rpm);
3. The motor velocity is then converted to wheel velocity (in rpm) by applying the gear ratio;
4. The target and measured wheel velocity are then compared;
5. Multiply the difference by Kp and Kd and do integration. For this project, the cumulated difference expressed in unit of rpm is converted to PWM.

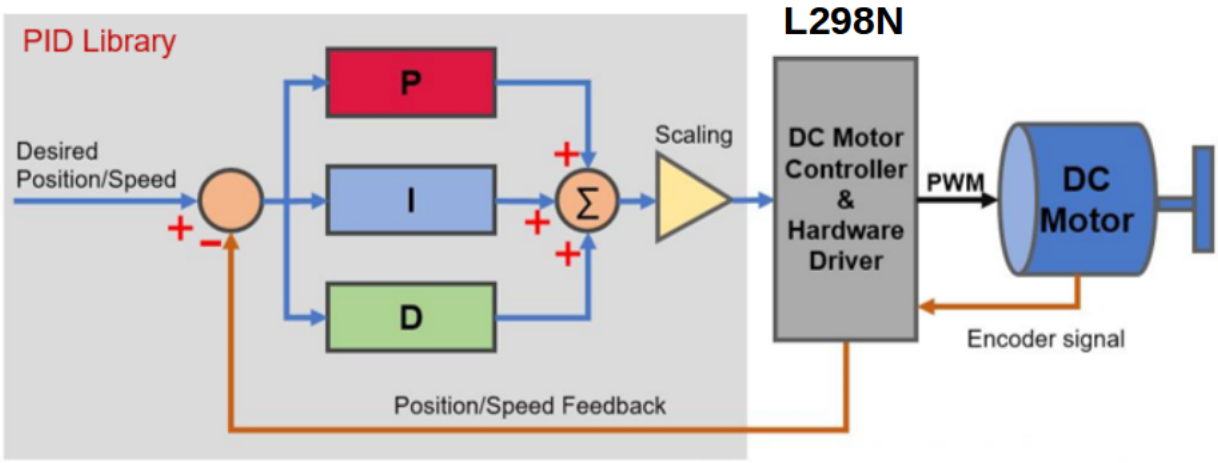



Figure 2.20: PID controller [15]

Table 2.1: PID parameters of three motors

| | P (proportional) | I (integral) | D(derivative) |
|---|---|---|---|
| M1 | 29.8 | 83.12 | 1.286 |
| M2 | 20.5 | 81 | 1.194 |
| M3 | 21.8 | 93.12 | 0.986 |

The implementation of PID controller is based on PID library in C language, which is shown in Appendix.

### 2.4.3 Kalman filter

Kalman filter (KF) is a kind of Bayes filters, and it represents belief by the moments representation that is, the belief at time t is represented by covariance $\Sigma_t$ and mean $\mu_t$. The belief at time $t-1$, which are represented by $\mu_{t-1}$ and $\Sigma_{t-1}$, is feed into KF. For parameters updating, KF requires the control $u_{t-1}$ and the measurement $z_t$. The belief at time $t$ is the output, represented by $\Sigma_t$ and $\mu_t$.
In the Algorithm. 2 of implementation of KF, the predicted belief $\mu_t$ and $\Sigma_t$ is calculated representing the belief which is obtained by incorporating the control $u_t$ at one time step later, but before incorporating the measurement $z_t$.

**Algorithm 2:** KALMAN FILTER – pseudo code realization [16]

**Input** :$x_{t-1}$, $\Sigma_{t-1}$, $u_t$, $z_t$
**Output** :$x_t$, $\Sigma_t$

**Prediction**:
$\overline{\mu}_t = A_t\mu_{t-1} + B_t u_t$;
$\overline{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$;
**Correction**:
$K_t = \overline{\Sigma}_t C_t^T \left(C_t\Sigma_t C_t^T + Q_t\right)^{-1}$;
$\mu_t = \overline{\mu}_t + K_t\left(z_t - C_t\overline{\mu}_t\right)$;
$\Sigma_t = \left(I - K_t C_t\right)\overline{\Sigma}_t$;

This algorithm is implemented in C language and displayed in Appendix.

### 2.4.4 Basic SLAM structure

The basic SLAM structure is shown in Fig 2.21, Where there are five steps.

1. The first step is sensor data input, the data usually include camera images, encoders and IMU data.

2. Visual Odometry (VO), the task of VO is to estimate the motion of camera based on two series frame of images, as well as the structure of partial map. Thus, VO is also called front end.

3. Optimization. This step system gather up estimated camera motion and pose from VO at different time stamp, and optimize based on them to get a consistent trajectory of robots and the surrounding map. Thus it is also called Back End.

4. Loop closing. This step will judge based on processed information if the robots return the former arrived location and return the information to Back End.

5. Mapping, this step it builds up the whole map based on estimated trajectory and task requirements.

The general SLAM solution could be described as

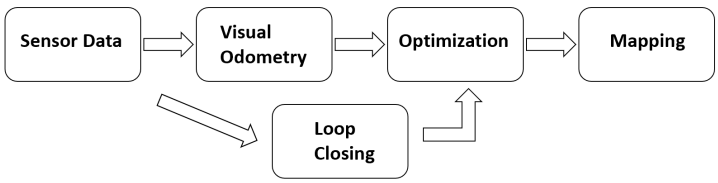



Figure 2.21: The basic structure of SLAM solution

$$\mathcal{X}_k = \mathcal{F}(\mathcal{X}_{k-1}, \mathcal{U}_k, \mathcal{W}_k) \tag{2.11}$$
$$\mathcal{Z}_{k,j} = \mathcal{H}(\mathcal{Y}_j, \mathcal{X}_k, \mathcal{V}_{k,j}) \tag{2.12}$$

The Equation 2.11 is called the motion function, where $\mathcal{X}_k$ is the location of robot (camera) at time instance $k$, while $\mathcal{X}_{k-1}$ is the location at time $k-1$; $\mathcal{U}_k$ is the measurement of motion sensor, which is also called input; $\mathcal{W}_k$ is the noise at the time $k$. The Equation 2.12 is called the observation function, where $\mathcal{Z}_{k,j}$ is a observation data of robot located at $\mathcal{X}_k$ while observing the sign $\mathcal{Y}_i$; and $\mathcal{V}_{k,j}$ is the noise introduced in this observation action. To expand the equations in graphical model, like Figure. 2.22 shows, for time stamp t-1, t and t+1 they have sensor measurement at each moment as input, and combining with last time's location for position estimation; while the observation of sign in environment at each moment produce observation data $\mathcal{Z}$ and help robots to localize itself in map.

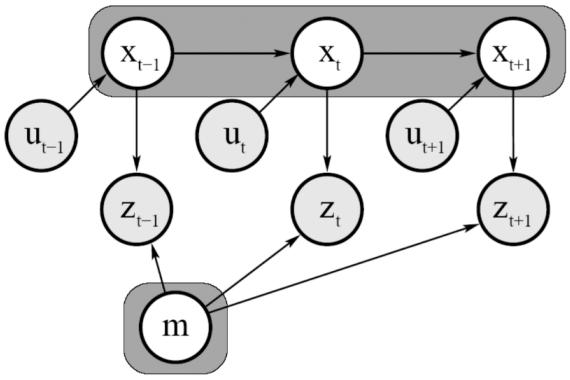



Figure 2.22: Graphical model of SLAM of three states

When considering the visual SLAM problem, the observation function is the process of obtaining observation data in pixels of capturing the signs. The motion function and observation function describe the basic SLAM problem and change it to state estimation problem. Then it is understandable that one famous SLAM solution is based on linear/nonlinear Gaussian/non-Gaussian method according to definition of noise. But later on the Graph Optimization method for state estimation has become a better solution, which leads to the implementation of RTABMap-ros package.

### 2.4.5 SLAM solution: RTAB-Map

To help design our own SLAM system for specific tasks, the RTAB-Map[17], for Real-Time Appearance-Based Mapping, is a good SLAM solution which supports lidar, stereo and RGB-D three modes, which enables us of testing on hardware and algorithms. An overview of this system is shown in Figure 2.23.

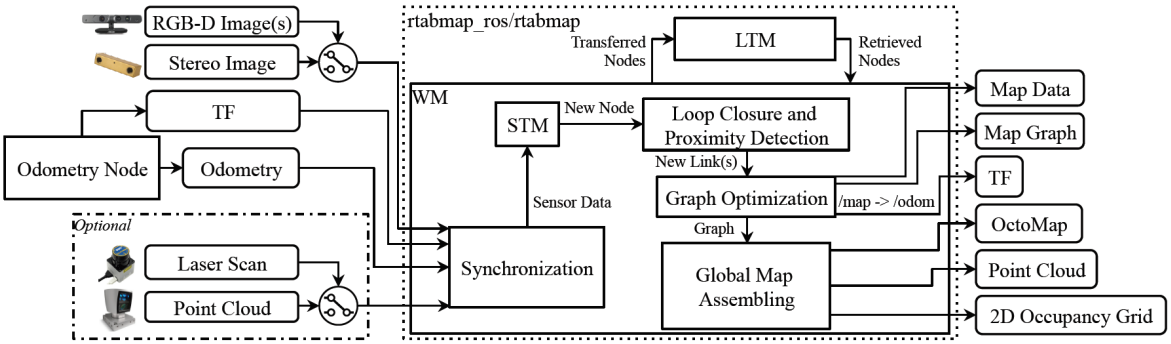



Figure 2.23: RTABMAP system overview [17]

For this project, the solution of using RGB-D camera and wheel encoder odometry is for reference. As SLAM is a complicated system problem, these open source solutions throw a light on the way of solving it.

# Chapter 3

# Result and Discussion

The results of this project as well as discussion will be given in this chapter. It will be covered in the order of firstly the preliminary results by interim progress, then the basic wheel controlling at first, next it includes wheel odometry, and finally with the help of RTAB-Map it will discuss realization of localization and visual odometry, as well as the parts which is worth doing in the future.

## 3.1 Preliminary Results

The basic hardware infrastructure of AGV is built up completely, as shown in Fig 1.2. The communication process between laptop, Raspberry Pi and Arduino could be carried on successfully. The controlling part of omni wheels is build up which means the car could move according to instructions received from laptop to Raspberry Pi.
Then the images of RGB and depth are captured and shown in Figure 3.1, the scenery is the lab environment in IR building.

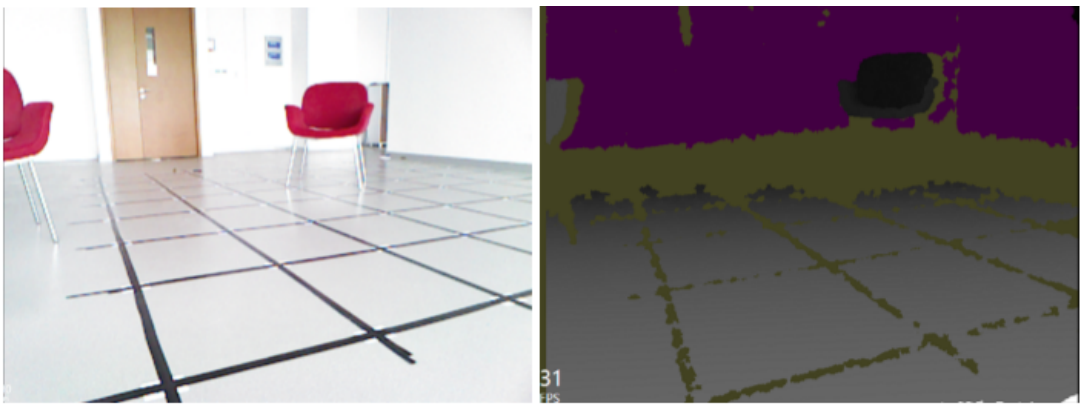


Figure 3.1: The example image from Kinect

As can be seen from Figure 3.1, for the near parts of up to around 3.8 meters, the Kinect depth camera could get a good calibration of depth; but beyond that range the measurement is not valid.
Then to test the process of ORB feature point extraction and mapping, two images from open source dataset is obtained and the feature points are well matched.

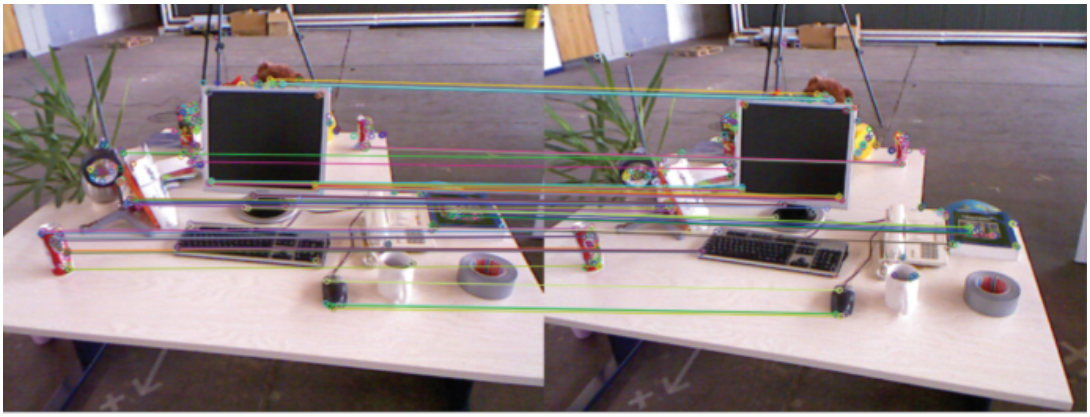


Figure 3.2: The feature extraction and matching from visual odometry

The final result is the ORB feature extraction from continuous videos, the figures of one original ORB image with extracted ORB feature points is shown in the first one. Then 7 figures of inconsistent 7 slices from a video of around 30 seconds with ORB feature points cloud is captured and shown below.

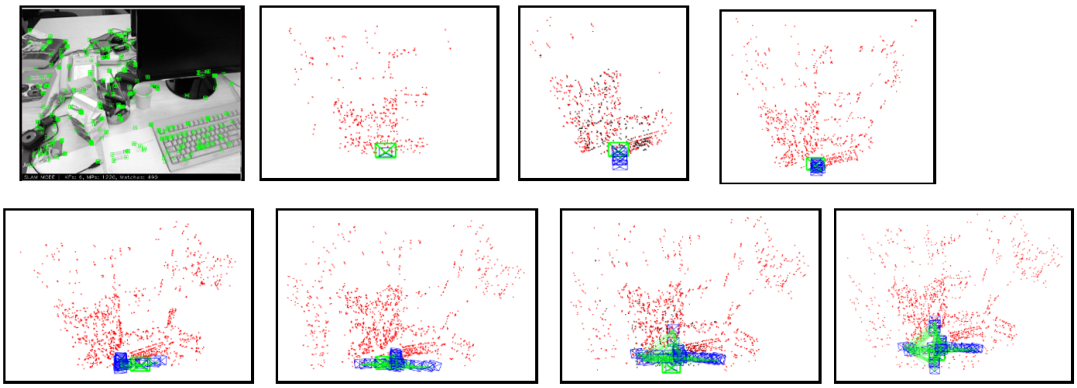


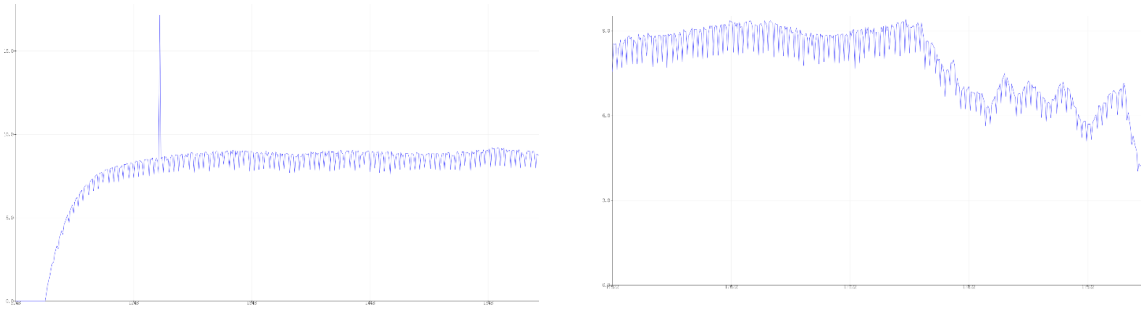
Figure 3.3: Slices from video of calculating ORB feature points cloud

Based on the ORB-SLAM system, a preliminary sparse feature points cloud could be drawn, but it cannot accomplish the task of localization, navigation and avoidance without. To ameliorate this situation, a more popular solution which is graph optimization based SLAM, is introduced. Especially with the combination of IMU (Inertial Measurement Unit) sensor and feature based (i.e, QR code or Hamming code) localization, the localization precision would be extraordinary.

## 3.2 Speed control and control resolution

PWM control with AGV running free is shown in Figure. 3.4a, where the speed reaches around 80% of maximum speed in only 0.2s, which means that the mobile robot reacts fast but the speed is highly influenced by violent acceleration and deceleration due to the power, in other words, speed is not stable and controllable. When putting it on ground, like Figure. 3.4b shows,

the mechanical stress suffered by onboard mechanical and electronic elements is also very high that distort original speed. Then adding PID controller to ameliorate this problem.

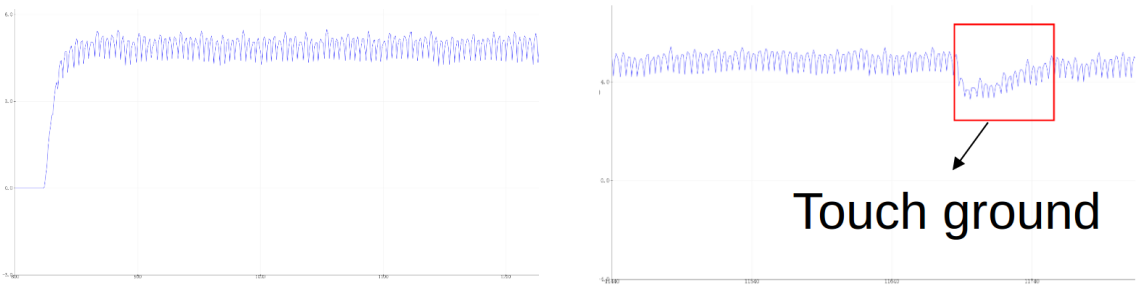


(a) PWM control when running free

(b) PWM control when touching ground

Figure 3.4: only PWM control

With the help of PID controller, the speed regulation is precise and fast as shown in Figure. 3.5a. The speed reaches around 90% of maximum speed in only 0.1s, which means that the mobile robot reacts very fast and noise is reduced in a small range. Although the speed cannot be fixed at one value due to noise and hardware deficiency, it would fluctuate in a small range, and the average speed is near to desired value. Then its performance when touching the ground is shown in Figure. 3.5b, where the motor firstly runs free and then touch the ground, as we can see, after a small period of deceleration this speed bounce back to a new speed smaller but very near the original speed.




(a) PID control when running free

(b) PID control when touching ground

Figure 3.5: With PID control

To reduce the influence and noise in motor rotation and speed detection process, kalman filter is used, and the testing output is shown in Figure. 3.6. As can be seen, the set desire value is 5 cm/s, after rotating the speed changes very fast and smooth, only a small overshoot appears at first convex, and it keeps very stable at around desired speed. But what if it runs on ground with the disturbance of press and friction? Therefore, the experiments using wheel encoder odometry is conducted.

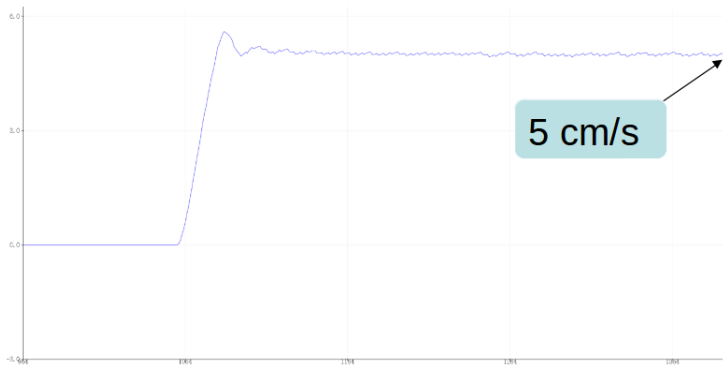



Figure 3.6: Motion analysis

## 3.3 Wheel encoder odometry experiments

This experiment about odometry is conducted by making the AGV move in a line of $8m$ calibrated by roll ruler, as shown in Figure. 3.7. During this process, the information of speed, pose and position change is sent out as form of"Messages" within ROS system, and they are plotted by the software called Matplot in real-time.

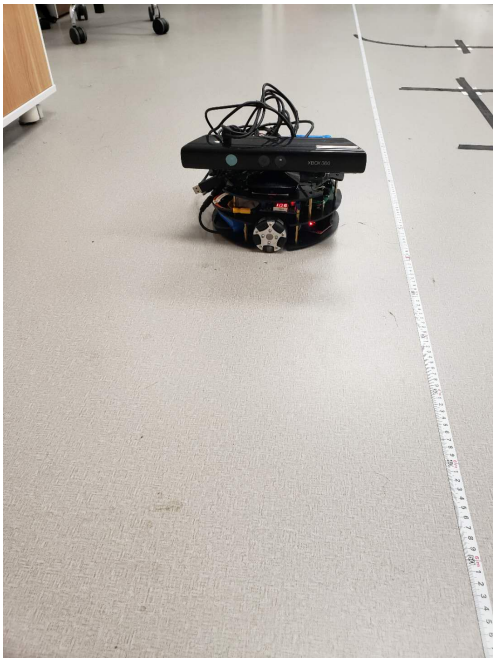


Figure 3.7: Scene shoot during experiment

The output of wheels' speed, position of AGV and orientation of AGV are information shown in Figure. 3.8, Figure. 3.9 and Figure. 3.10, respectively. With the setting speed of $y = 7\ cm/s$ where positive y direction is the head direction. We can see that after the speed calculation, the ideal wheel speed of wheel 1, 2 and 3 are $-6.06218\ cm/s$, $0\ cm/s$ and $6.06218\ cm/s$,

respectively. However, the Figure. 3.8 shows that the absolute value of both wheel 1 and 3 are smaller than ideal value, with a $8.23\%$ and $2.26\%$ of decrease. While the speed is fluctuating in the range of mainly $\pm 0.5cm/s$ for motor1 and $\pm 1cm/s$ for motor3.
Then considering the change of orientation, for Figure. 3.9 the "w" is an auxiliary symbol in Quaternion, as for the whole motion process the value of w changes from $0.993$ to $0.430$, and the value of z changes from $0.136$ to $0.902$. Therefore, the orientation change could be calculated as Equation. 3.1 shows, where the final orientation of AGV has a $-42.11°$ deviation in negative-yaw direction, which is basically consistent with visual results.

$$\Delta\alpha = \frac{0.430 - 0.993}{0.902 - 0.136} = -0.735 = -42.11° \tag{3.1}$$

For the position information in Figure. 3.9, AGV move in y-axis from $130.879cm$ to $1004.58cm$ and in x-axis from $-0.719cm$ to $-18.356cm$ in time period from $10.182s$ to $142.664s$.
The mainly velocity is in positive y-axis and it's speed can be calculated as $y = 6.6cm/s, x = -0.133cm/s$. The speed in y-axis is near desired speed as $7cm/s$.

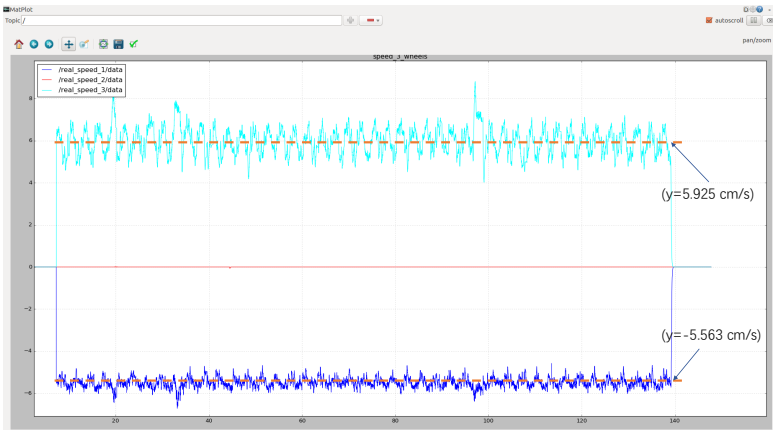



Figure 3.8: Speed of wheels: set 7 cm/s forward

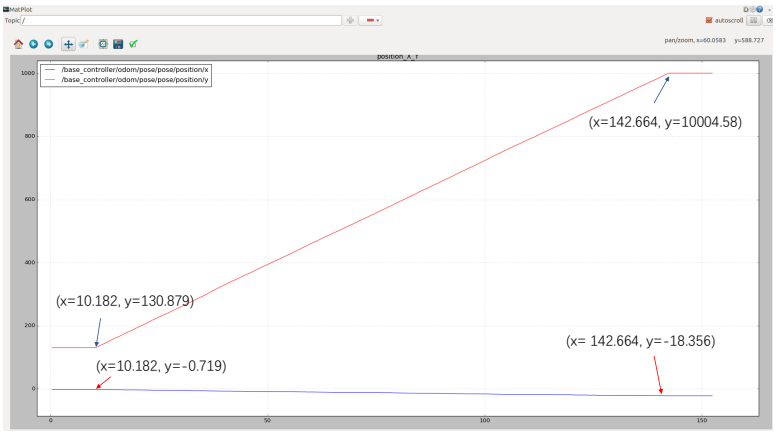



Figure 3.9: Position of AGV: set 7 cm/s forward

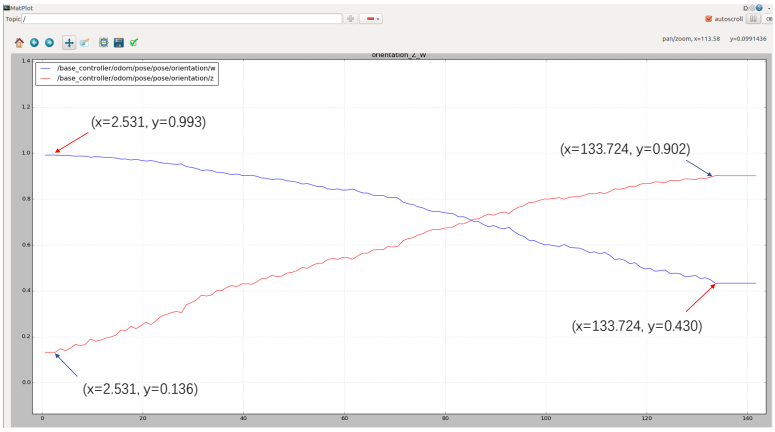



Figure 3.10: Orientation of AGV: set 7 cm/s forward

As with set speed of $7cm/s$ the output is not very ideal, a second group of experiments are conducted with set speed $5cm/s$. The output of wheels' speed, position of AGV and orientation of AGV information are shown in Figure. 3.11, Figure. 3.12 and Figure. 3.13, respectively. With the setting speed of $y = 5\ cm/s$, it's obvious that after the speed calculation, the ideal wheel speed of wheel 1, 2 and 3 are $-4.33013\ cm/s$, $0\ cm/s$ and $4.33013\ cm/s$, respectively. However, the Figure. 3.11 shows that the absolute value of both wheel 1 and 3 are smaller than ideal value, with a $12.15\%$ and $10.57\%$ of decrease. While the speed is fluctuating in the range of mainly $\pm 0.4cm/s$ for motor1 and $\pm 1cm/s$ for motor3.
Then considering the change of orientation, for Figure. 3.12 during the whole motion process the value of w changes from $0.428$ to $0.462$, and the value of z changes from $0.903$ to $0.888$. Therefore, the orientation change could be calculated as Equation. 3.2 shows, where the final orientation of AGV has a $-25.28°$ deviation in negative-yaw direction, which is basically consistent with visual result and is acceptable.

$$\Delta\alpha = \frac{0.888 - 0.903}{0.462 - 0.428} = -0.441 = -25.28° \tag{3.2}$$

For the position information in Figure. 3.12, AGV move in y-axis from $1333.23cm$ to $2168.45cm$ and in x-axis from $-13.366cm$ to $-19.048cm$ in time period from $13.089s$ to $208.673s$.
The mainly velocity is in positive y-axis and it's speed can be calculated as $y = 4.27cm/s, x = -0.029cm/s$. The speed in y-axis is near desired speed as $5cm/s$, with a decrease of $14.6\%$.

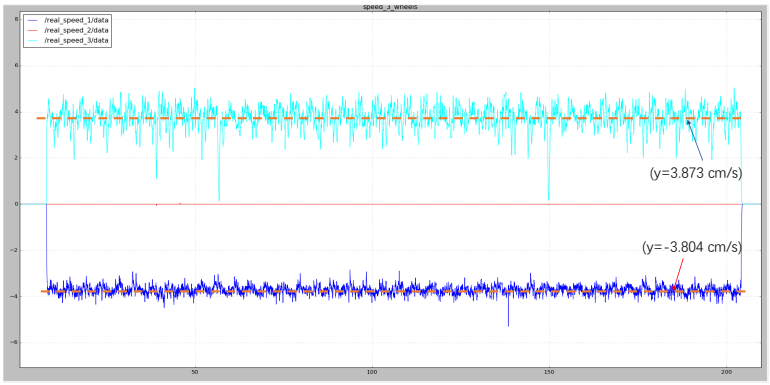



Figure 3.11: Speed of wheels: set 5 cm/s forward

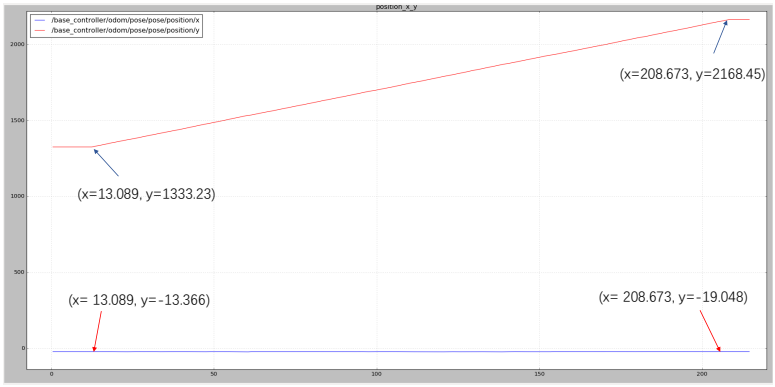



Figure 3.12: Position of AGV: set 5 cm/s forward

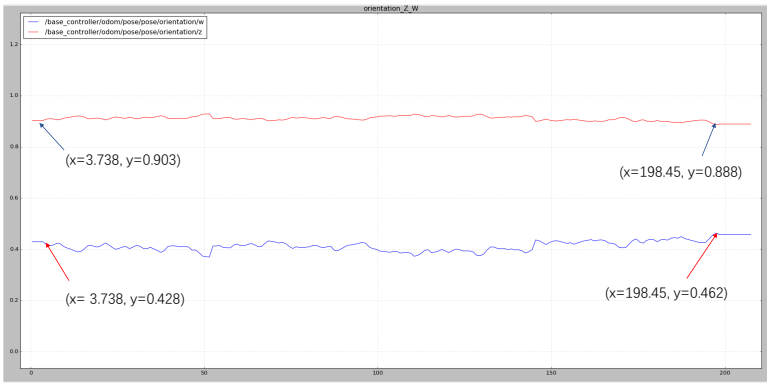



Figure 3.13: Orientation of AGV: set 5 cm/s forward

## 3.4 ROS project structure

A simplified system structure in ROS, which ignore transform and other relationship, is shown Figure. 3.14, with kinect, odometry, depthimage_to_laserscan and rtabmap works as nodes

within ROS, and the line with arrow indicates direction of data flow. With main messages shown along with arrow lines, the left name before slash is type of message and right one is the name of topics. For the depthimage_to_laserscan ROS package it aims to project the depth information scanned by kinect at specific height to ground, forming a layer of 2-D laser simulation it could be used for distance detection and avoidance.

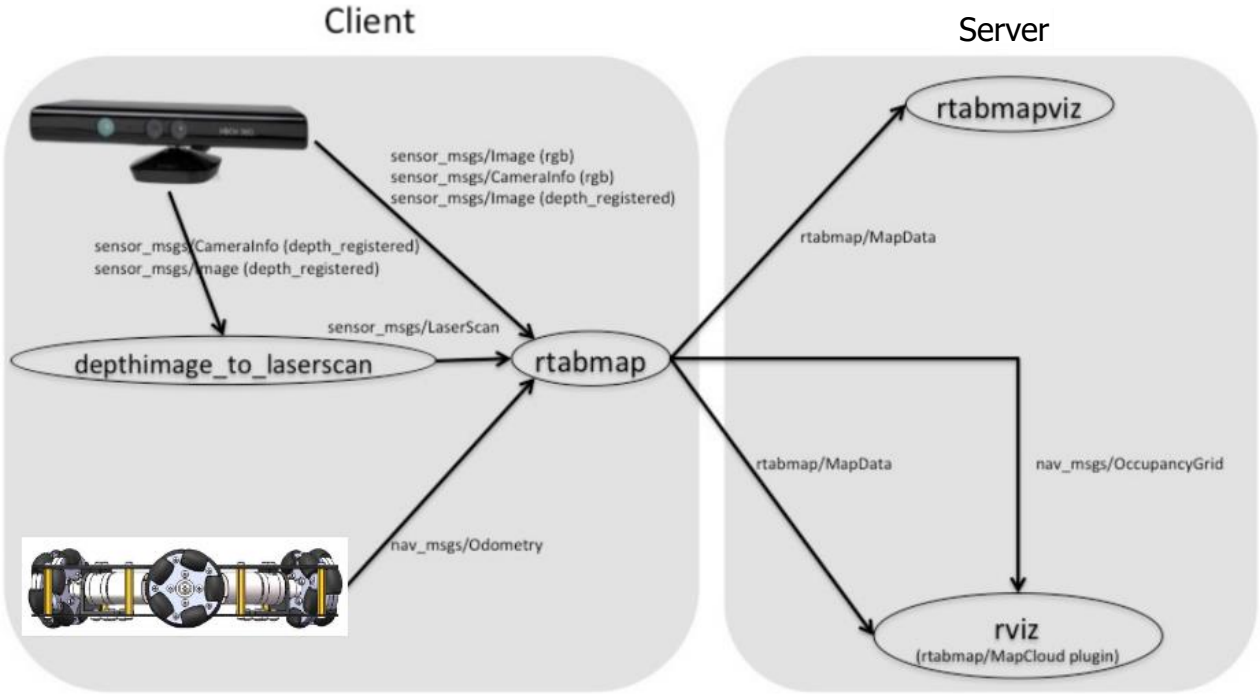
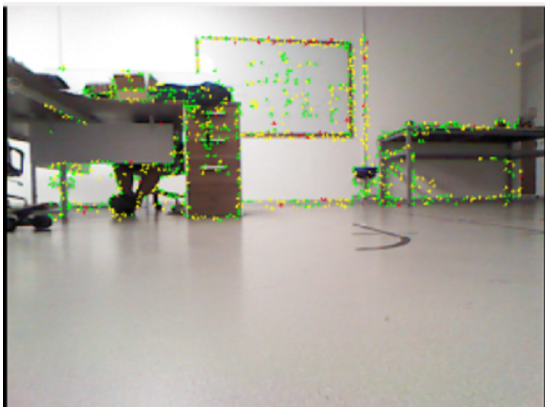


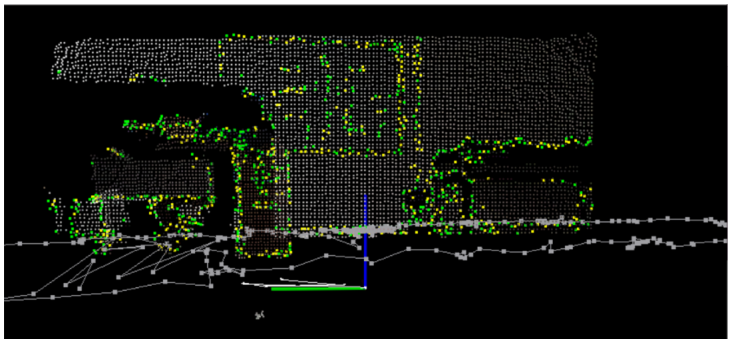

Figure 3.14: ROS project structure

## 3.5 Mapping result with visual odometry

After having satisfying result for controlling AGV, the map could be build up with the help of RTAB-Map ROS package. Figure. 3.15a is the RGB iamge with extracted feature points shown, while Figure. 3.15b shows the point cloud in depth image. These points are usually some points of objects boundary which has significant color level changes between adjacent pixels, which is key factors for map building and localization.



(a) Extracted features



(b) Point cloud and odometry

Figure 3.15: result1

After remote controlling AGV around the lab environment, a 3-D indoor point cloud map is build up successfully, as shown in Figure. 3.16 and Fuigure. 3.17, the blue lines on the ground consisted of points are odometry trajectories when moving AGV around. By the way, the ground is also configured as black, which indicates without obstacles.

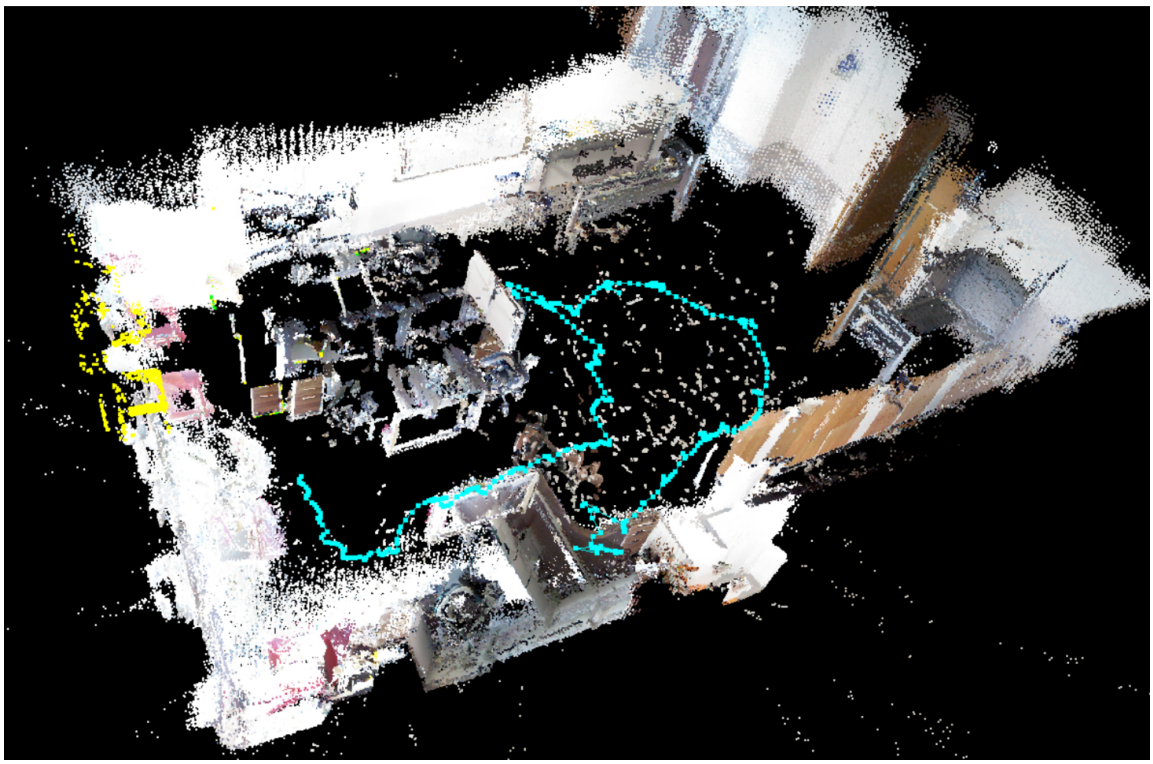


Figure 3.16: Map, look down from above

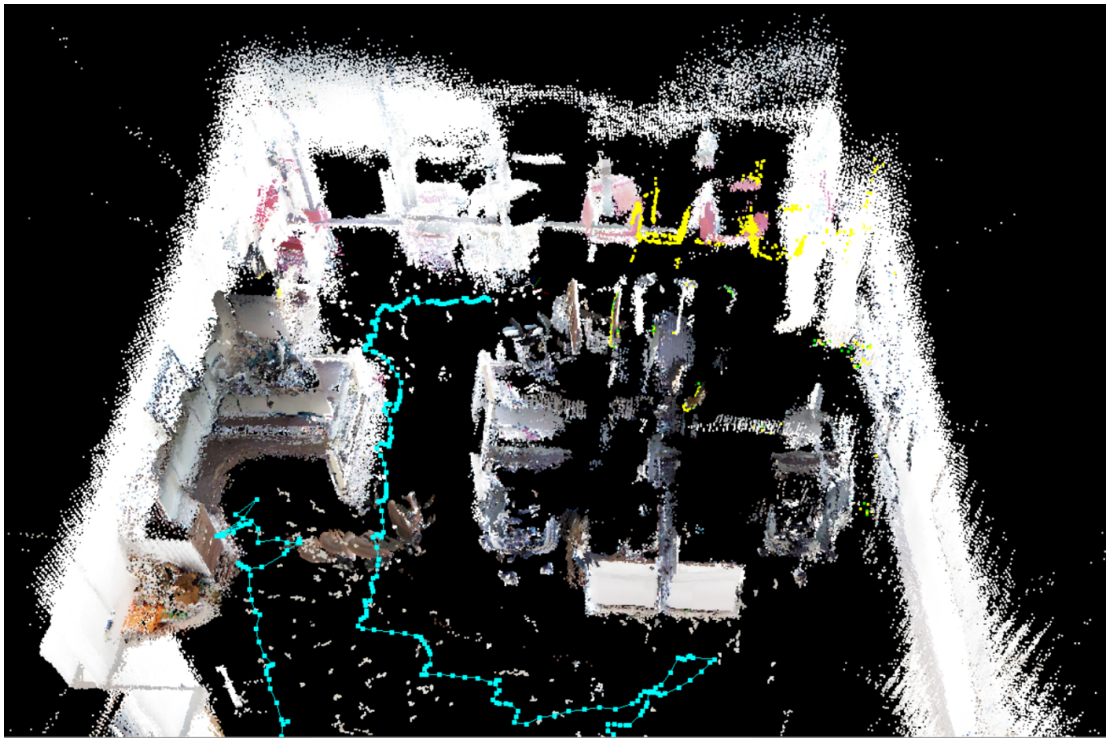


Figure 3.17: Map, look down from another view

## 3.6 Localization with Mapping result

For localization mode, the map dataset will not increase with new income data. Nevertheless, new coming graph will be used for comparison with map data, and system is trying to find a scene in map which matches input data best. An example is shown in Figure. 3.18, where the left middle window indicate the robot has localized itself in the map, then the position of robot will automatically change to right location with proper pose. Other localization results could be seen in Figure. 3.19 and Figure. 3.20. The small frame with blue, red and green axis indicates the estimated location, as well as pose of AGV.
After measurement in real scene by roll rulers, the localization precision is within $10cm$, as long as robot could localize itself after observing surrounding environment when put in an unknown location and turned on.

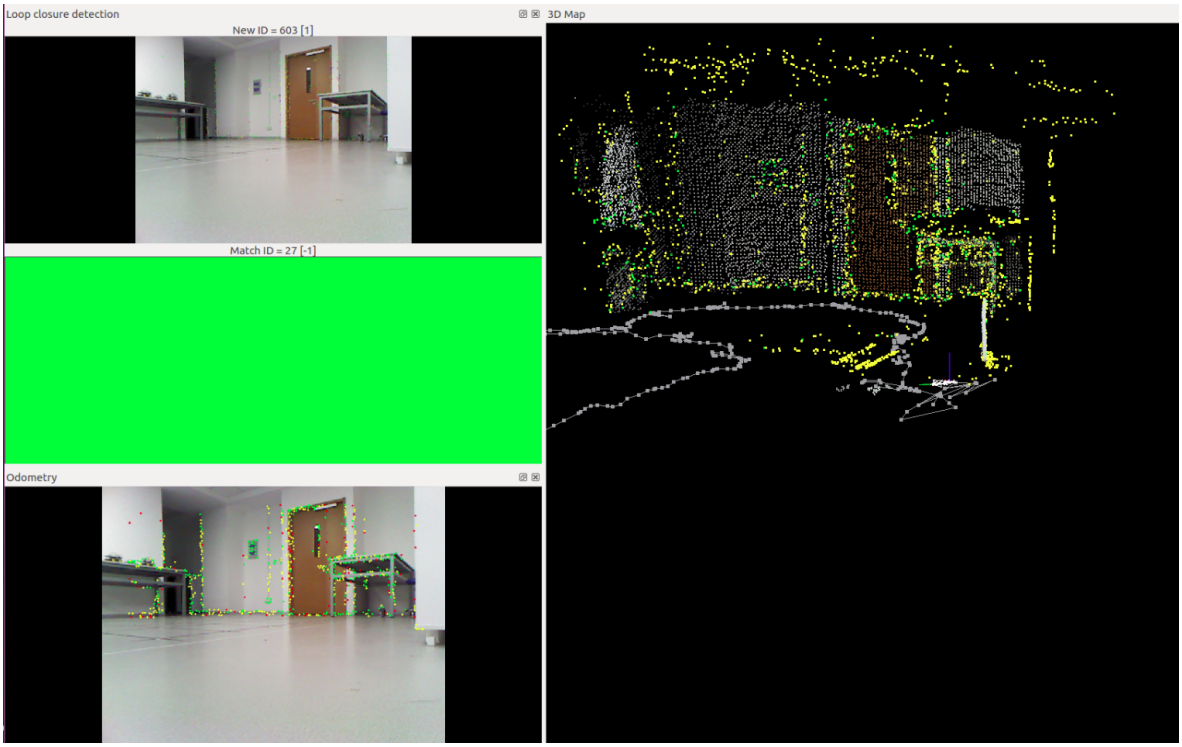
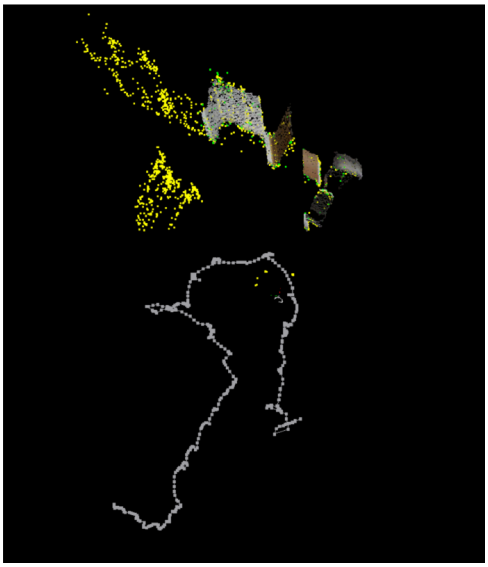
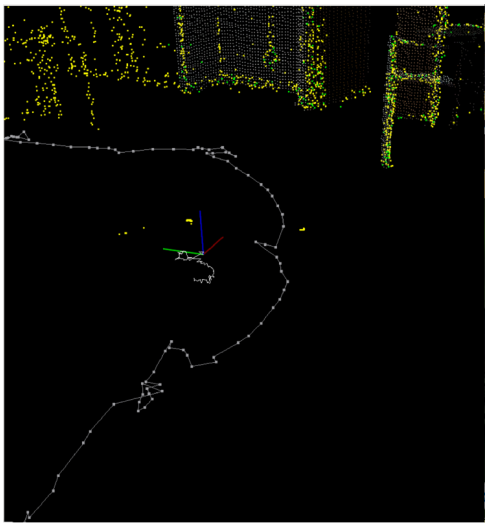



Figure 3.18: rtabmapviz: loop closure and localization success

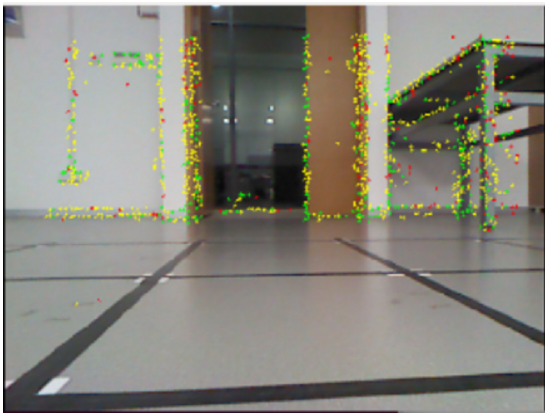


(a) Localization result 1

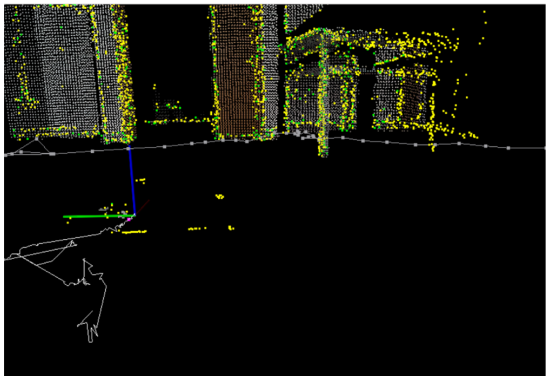


(b) Zoomed result

Figure 3.19: A figure with two subfigures



(a) RGB scene



(b) AGVA localize itself

Figure 3.20: A figure with two subfigures

## 3.7 Visual odometry testing

The visual odometry is also tested with a preliminary result shown in Figure. 3.21, with trajectory as white line plotted on the graph. There is a problem with this AGV is that when ground is not flat enough the car would bounce up and down, as a result the visual odometry is not very stable comparing to wheel odometry.

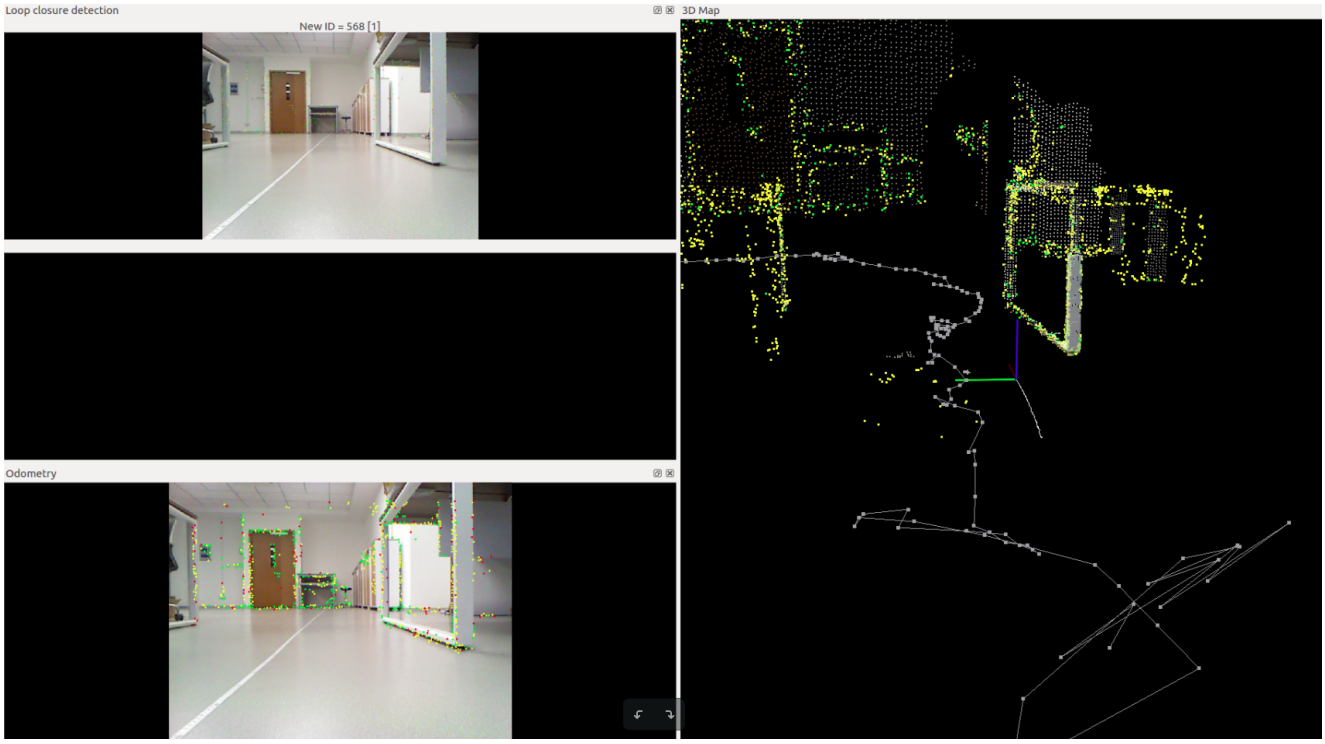



Figure 3.21: rtabmapviz: visual odometry with trajectory

# Chapter 4

# Improvement

This project has been introduced roughly with results discussed, there are some achievements while much more parts with deficiency could be improved in the future.

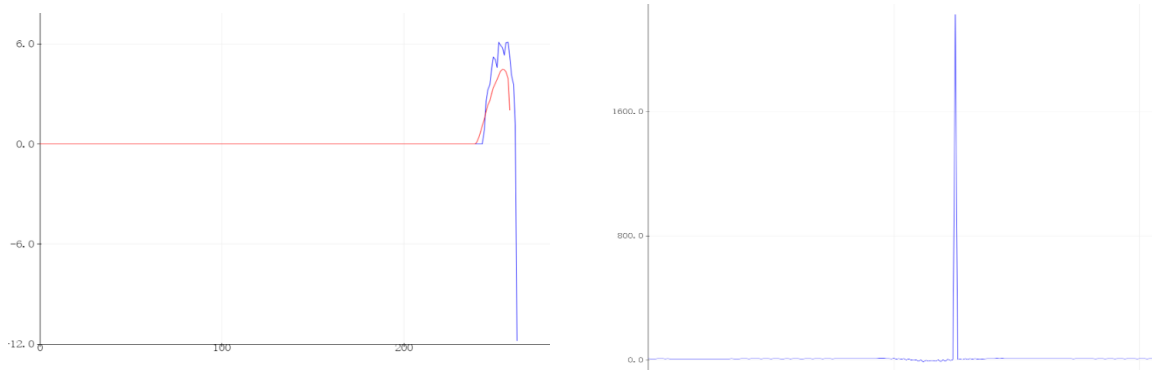
## 4.1 Discussion

### 4.1.1 Limited calculation power of Arduino board

For precise counting of pulses in encoders, the Arduino function called "Interruption" is implemented, and its work principle is every time a sensor data change in light sensor A would trigger interruption and stop present program and execute interruption function which realizes the encoder's speed calculation algorithm as in Algorithm. 1. While in low speed of motor as well as interruption the potential hazard would not erupt as MCU is capable of handling all the tasks, it would be a trouble when wheel's speed reach a threshold value, like $10cm/s$, and with wheel radius of $2.8cm$ the angular speed could be calculated as $3.571rad/s$ and the frequency of pulses are around $30KHz$, it seems acceptable for Arduino, but with the burden of three encoders (as speed are calculated separately for three wheels) and odometry the wave output immediately collapse like the Figure. 4.1a and Figure. 4.1b show.
To illustrate in detail of this phenomenon, in Figure. 4.1a the blue line the calculated speed and red line is the filtered speed with KF, when rotating the wheel very fast by hand in one direction, the plotting program will froze (or stop) when speed surpass threshold value. In Figure. 4.1b, when setting desired value beyond threshold value it will continuously trigger interruption function and the inner program of Arduino would stop (including wave plotting), then after a long period by chance it returns to inner program and output a large cumulated velocity and then trigger interruption again.
To solve this problem while keeping precise accord of encoders, hardware Interruption, like there are hardware Interruption support in Arduini Due board, and gear motor with less gear ratio is recommended, sometimes cause a normal gear motor is enough with high precision.

### 4.1.2 Sensitive Kinect

Even with power supply of $12V$ voltage, the current through kinect should keep stable, therefore filters in circuit is recommended. Of course the easiest way is to supply kinect with separate battery.




(a) wave output when rotate wheels by hand

(b) wave output when set a high desired speed.

Figure 4.1: Problems when using encoders on Arduino

### 4.1.3 Transfer speed

A trade-off problem is the way of transmitting video streaming and there are two ways available, which are by USB cable connection and Wi-Fi connection. Concerning transmit speed and video quality direction connection has better performance; while for Wi-Fi transmission it's more convenient to move AGV but speed is limited, moreover, streaming need to be compressed first and decompressed at receiver side, which is technically more difficult. For manipulation convenience this project choose direct USB connection, but the long USB wire add wights as well as friction to AGV car, therefore, it's not a permanent arrangement.

## 4.2 Future work

### 4.2.1 Fusing odometry

The odometry of both wheel encoder and visual part are tested and proved to be valid, but for better performance they could be fused using filters in development.

### 4.2.2 Replace camera with higher precision

While there is a higher version of Kinect v2 with higher accuracy, but v2 needs many pre-processing on the depth images before using them to compensates random noise and multi-path interference, besides, it need USB 3.0 interface for streaming. Although these pre-processings are not necessary for Kinect v1 [18], which makes it suitable for fast prototyping. IN terms of performance and depth precision Kinect V2 is a better choice in future work.

### 4.2.3 Implement of IMU and combine with camera

Although we like Big Clean Problems, and usually camera plays an important role in perception and navigation. But only one camera may be insufficient in industrial indoor environment where there tend to lack enough texture information, and camera-only technology would highly limit the speed of AGV, as feature mapping step of SLAM is easy to fail when the overlapping of two frames is too small.

Fortunately, with the help of IMU data, we could get a feasible estimation of pose even obtaining invalid visual input for a small period. IMU is widely used in the field of navigation system. The recommended solution is utilizing a nine-axis IMU module, which combines accelerometers, gyroscopes and magnetometers. With the high quality microprocessor and advanced motion calculation, as well as dynamic Kalman filter algorithm, the IMU module is able to calculate its own motion state simultaneously. But only IMU is only wise, as IMU could only provide accurate angular speed and acceleration speed for short period while there exists obvious drift that leads to large error when accumulating.
Comparing to the drift problem of IMU, camera data tend to stay stable. For example, if we put one camera on ground, the pose estimation would also be stable in static situation. That is to say, camera data is sufficient to adjust the drift problem of IMU, and they have complementary advantages.

# Chapter 5

# Conclusion

This project of "a visual RGB-D SLAM application in AGV with omni wheels" which has last for near one year has periodically coming to an end. There have been many work that are accomplished very well while others remain to be improved.
Personally, I have benefited much from this final year project. I become skilled in using a powerful Linux operating system called Ubuntu. I successfully applied mathematical knowledge obtained from university in this project, including linear algebra and calculus. I also enriched my skills in engineering mathematics including the application of PID controller and kalman filter. From the usage of encoders, motors, welding circuits and Arduino boards, I have seen my advantages as a bachelor majoring in Electrical Engineering. Finally, I have practiced my programming ability while dealing and writing codes in C, C++, and XML.
In a broader view, a completed AGV system is constructed from scratch that could accomplish complicated tasks. Accurate speed control of AGV is achieved with the help of encoders, PID controller and kalman filter algorithms ,and it could move in every direction according to instructions. A motion analysis model is established specially for this 3-axis omni wheels AGV. A wheel encoder odometry is written from scratch and tested to have gratifying performance. Moreover, a 3-D point cloud map of FYP's laboratory in IR-613 is build up. Finally, the AGV could localize itself in the lab environment with location precision within $10cm$ and wonderful pose estimation. This system has been proved to be well-performed and could be further developed by other interested fellows.

# Bibliography


[1] S. J. Julier and J. K. Uhlmann, "A counter example to the theory of simultaneous localization and map building," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 4, pp. 4238–4243, IEEE, 2001.

[2] K. S. Kim, R. Wang, Z. Zhong, Z. Tan, H. Song, J. Cha, and S. Lee, "Large-scale location-aware services in access: Hierarchical building/floor classification and location estimation using wi-fi fingerprinting based on deep neural networks," in *2017 International Workshop on Fiber Optics in Access Network (FOAN)*, pp. 1–5, Nov 2017.

[3] Z. Zhong, Z. Tang, X. Li, T. Yuan, Y. Yang, M. Wei, Y. Zhang, R. Sheng, N. Grant, C. Ling, X. Huan, K. S. Kim, and S. Lee, "Xjtluindoorloc: A new fingerprinting database for indoor localization and trajectory estimation based on wi-fi rss and geomagnetic field," in *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, pp. 228–234, Nov 2018.

[4] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pp. 1271–1278, IEEE, 2016.

[5] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[6] T. Stoyanov, J. Saarinen, H. Andreasson, and A. J. Lilienthal, "Normal distributions transform occupancy map fusion: Simultaneous mapping and tracking in large scale dynamic environments," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 4702–4708, IEEE, 2013.

[7] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[8] S. Vignesh, R. K. SK, and N. Lingaraj, "Intelligent automated guided vehicle using visual servoing,"

[9] D. S. Schueftan, M. J. Colorado, and I. F. M. Bernal, "Indoor mapping using slam for applications in flexible manufacturing systems," in *Automatic Control (CCAC), 2015 IEEE 2nd Colombian Conference on*, pp. 1–6, IEEE, 2015.

[10] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conference on Computer Vision*, pp. 834–849, Springer, 2014.

[11] Z. Cai, J. Han, L. Liu, and L. Shao, "Rgb-d datasets using microsoft kinect or similar sensors: a survey," *Multimedia Tools and Applications*, vol. 76, no. 3, pp. 4313–4355, 2017.

[12] E. S. Unipersonale, "What is the encoder code wheel?,"

[13] A. Mahtani, L. Sanchez, E. Fernandez, and A. Martinez, "Effective robotics programming with ros - third edition," Packt Publishing, 2016.

[14] J. Stuelpnagel, "On the parametrization of the three-dimensional rotation group," *SIAM review*, vol. 6, no. 4, pp. 422–430, 1964.

[15] E. S. Unipersonale, "pid-motor-control,"

[16] G. Welch and G. Bishop, "An introduction to the kalman filter," (Chapel Hill, NC, USA), University of North Carolina at Chapel Hill, 1995.

[17] M. Labb and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation: Labb and michaud," *Journal of Field Robotics*, vol. 36, 10 2018.

[18] O. Wasenmüller and D. Stricker, "Comparison of kinect v1 and v2 depth images in terms of accuracy and precision," in *Asian Conference on Computer Vision*, pp. 34–45, Springer, 2016.

# Appendices

```
//File "PID_ros_Motor1.ino"Created by Zhenghang Zhong
//email: 729170049@qq.com
//This is an Arduino file used in ROS system
//Cntrol of motor1, with encoder, PID and Kalman filter used

#include "PID_v1.h"
#include <ros.h>   //ros library
#include <geometry_msgs/Twist.h>  //message type, geometry
#include <std_msgs/Float64.h>     //ros standard float type
#include <Arduino.h>

#define Interrupt1 0            //Interruption pin 0, in Arduino Uno it's
    pin 2

#define pi 3.1415926f

ros::NodeHandle nh;             //Define ros node handler
geometry_msgs::Twist odom_msg;  //Initialize odometry message

int DIRA = 6;                    //M1 Direction Control positive
int DIRB = 7;                   //M1 Direction Control positive
int PWM = 9;                    //PWM control pin
int encoderA = 2;               //encoder light sensor A pin
int encoderB = 8;               //encoder light sensor B pin

int stateB = 0;                 //store light sensor B's reading at present
float Speed;                    //store Speed value
double duration;                //the number of the pulses of Moter1 in a
    short time
byte encoderALast;
boolean Direction;              //the rotation Direction1, False means
    Positive

double sampleTime = 10;         //The time interval of detecting speed as
    well as refresh PID and KF parameters

unsigned long beginTime;        //For time counting
unsigned long endTime;

//PID variables
double Kp=29.8,Ki=83.12, Kd=1.286;       //Adjust the PID to fit the motor
double Setpoint,Input,Output;            //required speed
double set = 0;                          // fixed setPoint, will be changed
    by receiving data

PID myPID(&Input,&Output,&Setpoint,Kp,Ki,Kd,DIRECT);     //Initialize the
    PID controller

//KBlman filter added
// Initialization of KBlman Variables
float R = 6e-3, Q = 1e-4;         //Q = process noise covariance, R =
    measurement noise covariance
double Xpe0 = 0.0;              // Xpe0 = prior estimation of signal X at
    time t=0 (current state)
```

```
double Xe1 = 0.0;                 //Xe1 = estimation of X at time t=1 (
    previous state)
double Ppe0 = 0.0;                //Ppe0 = prior estimation of "error
    covariance" at t=0,
double P1 = 1, P0 = 0;            //P1 = error covariance at t=1, P0 = error
    covariance at t=0
double  K = 0.0, Xe0 = 0.0, Z = 0.0; //K = KBlman gain, Xe0 = estimation of
     signal at t=0, Z = measured signal at t=0

//To obtian the desired speed info from messages
void desiredSpeed( const std_msgs::Float64& cmd_msg){
 set = cmd_msg.data;
}

std_msgs::Float64 realSpeed;      //actual speed value
ros::Subscriber<std_msgs::Float64> sub("hope_speed_1", &desiredSpeed );  //
    Initialize a ros subscriber which would subscribe message of desired
    speed of motor1
ros::Publisher statePub("real_speed_1", &realSpeed);                     //
    Initialize a ros state publisher which would publish message of actual
    speed of motor1


void setup(){
  EncoderInit();                  //Initialize encoder
  digitalWrite(encoderA,INPUT); //Set the working mode of encoder A as
      input
  digitalWrite(encoderB,INPUT);
  digitalWrite(DIRA,OUTPUT);      //Set the working mode of motor direction
      controller A as output
  digitalWrite(DIRB,OUTPUT);
  digitalWrite(PWM,OUTPUT);

  digitalWrite(PWM,LOW);
  myPID.SetMode(AUTOMATIC);   //Set the PID mode as automatic updating
  myPID.SetOutputLimits(-125,125);       //limiting the PWM outputs withim
      (-125,125), as large speed would make Arduino collapse, stated in
      report
  myPID.SetSampleTime(sampleTime);       //set PID's sampling time, or
      parameter refresh period
  myPID.SetTunings(Kp,Ki,Kd);            //Tuning of PID parameters

  beginTime = millis();                  //return present time in
      millisecoond

  nh.initNode();                         // Initialize ros node handler
  nh.subscribe(sub);                     //regist subscriber
  nh.advertise(statePub);                //regist publisher
}

void loop(){
  //Movement(testSpeed); //test motor motion

  PIDMovement (set);  //feed required speed value to PID
  endTime = millis();
```

```
  if(endTime - beginTime >=sampleTime)
  {
    Speed= float( duration/53735*5.8*pi*1000/(endTime - beginTime));
                // every rotation circle around 53735 pulses generated,
       wheel diameter is 5.8cm, all units are in cm/s
    beginTime = endTime;

        //Kalman Filter parameters refreshment
    Z = Speed;
    //Serial.print(duration2 , 5);Serial.print("\t"); //for testing, plot
       output
    Xpe0 = Xe1; //Assumption or prediction 1
    Ppe0 = P1 + Q; //Assumption or prediction 2
    K = Ppe0/(Ppe0 + R); // Measurement update or correction of "KBlman
       gain"
    Xe0 = Xpe0 + K * (Z - Xpe0); // Measurement update or correction of "
       estimated signal"
    P0 = (1 - K) * Ppe0; // Measurement update or correction of "error
       covariance"
    //Serial.println(Xe0 , 5);
    Xe1 = Xe0; //Update: current t=0 becomes t=1 in the next step
    P1 = P0; //Update: current t=0 becomes t=1 in the next step
    Input = Xe0; //represh Input value to PID from KF output

    realSpeed.data = Xe0;  // feed to publishser
    if(abs(realSpeed.data) < 0.01)  //prevent noise
    {
      realSpeed.data=0;
    }
    statePub.publish( &realSpeed ); //publishing

    duration = 0;        //reset
    nh.spinOnce();       //node handler working once
  }
}


void Movement(int a)
{
  if (a>0)
  {
    analogWrite (PWM,a);       //PWM Speed Control
    digitalWrite(DIRA,HIGH);
    digitalWrite(DIRB,LOW);
  }
  else if(a<0)
  {
    analogWrite (PWM,-a);       //PWM Speed Control
    digitalWrite(DIRA,LOW);
    digitalWrite(DIRB,HIGH);
  }
  else if (a==0)     //stop
  {
    digitalWrite(DIRA,LOW);
    digitalWrite(DIRB,LOW);
```

```
    analogWrite(PWM,0);
  }
}

//PID modules
void PIDMovement(int a)
{
    Setpoint=a;
    Input=Speed;
    myPID.Compute();    //compute and execute PID
    Movement (Output);  //Move according to PID output which refresh itself
        automatically
}

//Encoder modules, it is illustrated in report
void EncoderInit() //Initialize encoder interruption
{
  Direction = true;//default -> Forward
  pinMode(encoderB,INPUT);
  attachInterrupt(Interrupt1, wheelSpeed, CHANGE);
}

void wheelSpeed()  //motor1 speed count
{
  int Lstate = digitalRead(encoderA);
  if((encoderALast == LOW) && Lstate==HIGH)
  {
    int stateB = digitalRead(encoderB);
    if(stateB == LOW && Direction)
    {
      Direction = false;  //when Direction keeps false, duration++
    }
    else if(stateB == HIGH && !Direction)
    {
      Direction = true;
    }
  }
  encoderALast = Lstate;

  if(!Direction)  duration++;
  else  duration--;
}
```

```
//File "odom.cpp" Created by Zhenghang Zhong
//email: 729170049@qq.com
//This is an C++ file used in ROS system
//odometry

#include <iostream>
#include <ros/ros.h>            //ros library
#include <tf/transform_broadcaster.h>
#include <std_msgs/Int16MultiArray.h>   //sandard ros variable type: int
    array
#include <std_msgs/Float64.h>           //sandard ros variable type: 64 bit
```

```
    long float
#include <nav_msgs/Odometry.h>          //sandard ros variable type:
    odometry type
#include <geometry_msgs/Twist.h>
#include <std_msgs/Float32MultiArray.h>
#include<math.h>
using namespace std;

struct ActThreeVell                     //define a structure type to store
    speed in  three wheels
{
    float v1;
    float v2;
    float v3;
};

//matrix for inverse speed calculation
const float A_inv[3][3] ={-0.3333333333, 0.66666666667, -0.333333333333,
                                        -0.57735027, 0, 0.57735027,
                                    0.0336700337, 0.0336700337,
                                        0.0336700337
};
//Rotation matrix
const float RotMatrixInv[3][3] = {
                                                    1.0, 0.0, 0.0,
                                                    0.0, 1.0, 0.0,
                                                    0.0, 0.0, 1.0
};


ActThreeVell real_Speed;
ActThreeVell hope_Speed;

//ros publisher initialization
ros::Publisher pub_hope_speed_1;
ros::Publisher pub_hope_speed_2;
ros::Publisher pub_hope_speed_3;

ActThreeVell ThreeWheelVellControl2(float Vx, float Vy, float angularVell)
{
#define AFA 60  //alpha angle
#define L 9.9  // L is the distance from center of car to center of wheel,
    in cm
#define pi 3.1415926f
ActThreeVell vell;

float theta = 0;
vell.v1 = (float)(-cos((AFA + theta) / 180.0f*pi) * Vx - sin((theta + AFA)
    / 180.0f*pi) * Vy + L * angularVell);

vell.v2 = (float)(cos(theta / 180.0f*pi) * Vx + sin(theta /180.0f*pi) * Vy
    + L * angularVell);

vell.v3 = (float)(-cos((AFA - theta) / 180.0f * pi) * Vx + sin((AFA - theta
    ) / 180.0f*pi) * Vy + L * angularVell);
```

```
return vell;
}

void real_speed_1_setter(const std_msgs::Float64& speed){
        real_Speed.v1 = speed.data;
}
void real_speed_2_setter(const std_msgs::Float64& speed){
        real_Speed.v2 = speed.data;
}
void real_speed_3_setter(const std_msgs::Float64& speed){
        real_Speed.v3 = speed.data;
}

//transform from speed in x-y to wheels' speed
void cmd_vel_setter(const geometry_msgs::Twist& cmd_vel){
        std_msgs::Float64 v1;
        std_msgs::Float64 v2;
        std_msgs::Float64 v3;
        ActThreeVell vell = ThreeWheelVellControl2(cmd_vel.linear.x*3,
            cmd_vel.linear.y*3, cmd_vel.angular.z/3);
        v1.data = vell.v1;
        v2.data = vell.v2;
        v3.data = vell.v3;
        pub_hope_speed_1.publish(v1);
        pub_hope_speed_2.publish(v2);
        pub_hope_speed_3.publish(v3);

        ROS_INFO("v1: [%f], v2: [%f], v3: [%f]", v1.data, v2.data, v3.data)
            ;
}

int main(int argc, char** argv)
{
        ros::init(argc, argv, "odometry_publisher");
        ros::NodeHandle nh;
        ros::Publisher odom_pub = nh.advertise<nav_msgs::Odometry>("/
            base_controller/odom", 100);
        tf::TransformBroadcaster odom_broadcaster;
        ros::Subscriber real_speed_1_sub = nh.subscribe("real_speed_1",50,
            real_speed_1_setter);
        ros::Subscriber real_speed_2_sub = nh.subscribe("real_speed_2",50,
            real_speed_2_setter);
        ros::Subscriber real_speed_3_sub = nh.subscribe("real_speed_3",50,
            real_speed_3_setter);

        pub_hope_speed_1 = nh.advertise<std_msgs::Float64>("hope_speed_1",
            1);
        pub_hope_speed_2 = nh.advertise<std_msgs::Float64>("hope_speed_2",
            1);
        pub_hope_speed_3 = nh.advertise<std_msgs::Float64>("hope_speed_3",
            1);


        float A_times_Rot[3][3];
```

```
        float R_Speed_wheel[3];
        float R_Speed_world[3];
        double x=0.0;
        double y=0.0;
        double th=0.0;
        double vx = 0;
        double vy = 0;
        double vth = 0;
        float temp =0;

        temp = 0.0;
        for(int i=0; i<3; i++)
        {
                for(int j=0; j<3; j++)
                {
                        for(int m=0; m<3; m++)
                        {
                                temp += RotMatrixInv[i][m]*A_inv[m][j];
                        }
                        A_times_Rot[i][j] = temp;
                        temp =0;
                }
        }

        //real speed need calcu to return speed in x,y, angular
        ros::Time current_time, last_time;
        current_time = ros::Time::now();
        last_time = ros::Time::now();
        ros::Rate r(1.0);

        ros::Subscriber cmd_vel_sub = nh.subscribe("/turtle1/cmd_vel", 50,
            cmd_vel_setter);


while(nh.ok()){
        //the size of the message queue.  If messages are arriving faster
                than they are being processed, this is the number of
           messages that       will be buffered up before beginning to throw
            away the oldest ones.
        ros::spinOnce();               // check for incoming messages
        current_time = ros::Time::now();

        //With given robot speed, compute odometry in a typical way
    double dt = (current_time - last_time).toSec();

        temp=0;
        //float R_Speed_wheel[3] = {real_Speed.v1, real_Speed.v2,
            real_Speed.v3};
        R_Speed_wheel[0] = real_Speed.v1;
        R_Speed_wheel[1] = real_Speed.v2;
        R_Speed_wheel[2] = real_Speed.v3;
        //To get real speed of AGV from wheels' speed
        for(int i=0; i<3; i++)
        {
                for(int m=0; m<3; m++)
```

```
            {
                    temp += A_times_Rot[i][m]*R_Speed_wheel[m];
            }
            R_Speed_world[i] = temp;
            temp =0;
    }
    //speed in x,y and yaw speed
    vx = R_Speed_world[0];
    vy = R_Speed_world[1];
    vth = R_Speed_world[2];

ROS_INFO("R_Speed_world_vx: [%f], vy: [%f], vth: [%f]", R_Speed_world
    [0], R_Speed_world[1], R_Speed_world[2]);

double Delta_x = vx * dt;
double Delta_y = vy * dt;
double Delta_th = vth * dt;
//ROS_INFO("Delta_x_y_th: [%f], vy: [%f], vth: [%f]", Delta_x, Delta_y,
     Delta_th);
x += Delta_x;
y += Delta_y;
th += Delta_th;

//since odometry is 6DOF, a quaternion is created from yaw
geometry_msgs::Quaternion Odom_quat = tf::createQuaternionMsgFromYaw(th
    );

//publish the transform over tf
geometry_msgs::TransformStamped odom_transform;
odom_transform.header.stamp = current_time;
odom_transform.header.frame_id = "odom";
odom_transform.child_frame_id = "base_link";

    odom_transform.transform.rotation = Odom_quat;
    odom_transform.transform.translation.z = 0.0;  //obtain yaw
odom_transform.transform.translation.x = x;
odom_transform.transform.translation.y = y;



    //to send the transform though broadcaster
odom_broadcaster.sendTransform(odom_transform);

    // publish the odometry message over ROS
nav_msgs::Odometry Odom;
Odom.header.stamp = current_time;
Odom.header.frame_id = "odom";

    //set the velocity in odometry
Odom.child_frame_id = "base_link";
Odom.twist.twist.linear.x = vx;
Odom.twist.twist.linear.y = vy;
Odom.twist.twist.angular.z = vth;

    //set the position in odometry
```

```
    Odom.pose.pose.position.x = x;
    Odom.pose.pose.position.y = y;
    Odom.pose.pose.position.z = 0.0;
    Odom.pose.pose.orientation = Odom_quat;

        //Publish the message
    odom_pub.publish(Odom);

        //refresh the last time variable
        last_time = ros::Time::now();
        //stop and wait for next loop
        r.sleep();
        }
}
```

```
        \\For calibration, file in C++
    \\fx : focal length x
    \\fy: focal length y
    \\cx: optical center x
    \\cy: optical center y
    \\ for the 16-bit PNG files
    factor = 5000
    \\To get the coordination of 3D depth points
    for v in range(depth_image.height):
      for u in range(depth_image.width):
        Z = depth_image[v,u] / factor;
        X = (u - cx) * Z / fx;
        Y = (v - cy) * Z / fy;
```

```
%camera intrinsic parameters
%Calibration output file for kinect depth camera:
    depth_A70773X07269241A.yaml
%Location: /home/<username>/.ros/camera_info
image_width: 640
image_height: 480
camera_name: depth_A70773X07269241A
camera_matrix:
  rows: 3
  cols: 3
  data: [582.7506440953495, 0, 318.0310701126388, 0,
    584.0312737377828, 251.8010085186393, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [-0.05827476044625381, 0.1388510360952216,
    0.002194165041538614, -0.002133718901370064, 0]
rectification_matrix:
  rows: 3
  cols: 3
```

```
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [584.8213500976562, 0, 316.8646212168132, 0, 0,
    586.8474731445312, 252.523176986766, 0, 0, 0, 1, 0]

%Calibration file for kinect rgb camera: rgb_A70773X07269241A.
   yaml
%Location: /home/<username>/.ros/camera_info
image_width: 640
image_height: 480
camera_name: rgb_A70773X07269241A
camera_matrix:
  rows: 3
  cols: 3
  data: [523.0091277522147, 0, 353.5812245432616, 0,
    521.6991914641592, 243.6861629744842, 0, 0, 1]
distortion_model: plumb_bob
distortion_coefficients:
  rows: 1
  cols: 5
  data: [0.02731296864632632, -0.05559154839761533,
    -0.001320097202715892, 0.007537711307951782, 0]
rectification_matrix:
  rows: 3
  cols: 3
  data: [1, 0, 0, 0, 1, 0, 0, 0, 1]
projection_matrix:
  rows: 3
  cols: 4
  data: [522.4449462890625, 0, 358.568872155287, 0, 0,
    526.1556396484375, 243.1694343603667, 0, 0, 0, 1, 0]
```