Hybrid Aerial and Ground-based Mobile Robot for Retail Inventory

by

Yibo Lyu

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 4, 2018

Keywords: autonomous mobile robot. RFID based inventory, SLAM, UAV, Parallel Tracking and Mapping (PTAM)

Approved by

Thaddeus A. Roppel, Chair, Associate Professor of Electrical and Computer Engineering
Shiwen Mao, Samuel Ginn Endowed Professor of Electrical and Computer Engineering
John Y. Hung, Professor of Electrical and Computer Engineering
Robert Dean, Professor of Electrical and Computer Engineering

Abstract

Radio frequency identification (RFID) technology is increasingly used in retail stores and warehouses because it can help to improve the inventory process, enable automatic checkout, and reduce shoplifting. This dissertation introduces two related robotic systems that are designed to support RFID-based inventory counting; the first is a ground-based mobile robot, and the second is an unmanned aerial vehicle (UAV). The UAV is intended to supplement the vertical reach of the ground vehicle.

The ground-based robot uses two novel methods to create a map and plan its path- (1) automapping, a semiautonomous algorithm that guides the robot through the space to be inventoried, and (2) multi-layer mapping, which synthesizes a structured-light sensor (e.g. Kinect) with a light detection and ranging (LIDAR). The experimental results show that the map made by the new automapping method is as good as one made manually. The multilayer map is more accurate than the map made by traditional simultaneous location and mapping (SLAM), compared with the ground truth of the map.

In this dissertation, a control algorithm for the UAV guides the UAV to a landing pad on the ground-based robot. This algorithm is named PTAM, for parallel tracking and mapping. PTAM is a camera tracking system which can work without pre-made map and landmarks. Also, it uses a 2-D camera to do a 3-D tracking. Therefore, the UAV can track its position and scale by using this algorithm. Then, it can generate a 3-D path according to inventory needs

and the UAV will move according to this path point by point. The experimental result shows that the UAV can finish a pre-set flight path with tolerable error. The flight path includes taking off from a landmark, moving from point to point as pre-set, and returning to the landmark.

Acknowledgments

I would first like to thank my advisor Dr. Thaddeus Roppel for his support and guidance during my time at Auburn University. I would like to thank Justin Patton for taking me to the RFID world and providing me with all the resources for my research. I would like to thank Dr. Senthilkumar CP for his instruction with RFID technology. I would like to thank Dr. Shiwen Mao, Dr. John Y. Hung and Dr. Robert N. Dean for serving my dissertation committee.

My sincere thanks also go to my wife for standing behind me throughout my life. She has been my inspiration and motivation for continuing to improve my knowledge. My dissertation could not have been finished without her help and support.

I would also like to thank my parents for their wise counsel and sympathetic ear. You are always there for me. Finally, there are my friends. We were not only able to support each other, but also happily by talking about things other than just our papers.

At last I would like to thank all my colleagues in our lab and at Auburn University. I enjoy working here for the pleasant atmosphere and the mind storm here also contributes to my dissertation. A special thank to Dr. Jian Zhang. He gives me many help and instructions with my research work.

Table of Contents

List of Tables

List of Illustrations

List of Abbreviations

ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
LIDAR	Light Detection and Ranging
RFID	radio-frequency identification
EPC	Electronic Product Code
UPC	Universal Product Code
LF	low-frequency
HF	high-frequency
UHF	ultra-high frequency
VHF	very-high frequency
UAV	Unmanned aerial vehicle
PTAM	Parallel Tracking and Mapping
FPV	first-person vision
EKF	extended Kalman filter

Chapter 1 Introduction

1.1 Introduction

Because of the appearance of low cost and stable RFID tags, the retail world has begun a rapid conversion from barcode to RFID technology [1]. Even if there are already many applications of RFID technology working well in retail stores, how to perform an inventory efficiently and quickly is still a big problem. For the RFID-based inventory [2], which is usually called “cycle counting”, people need to walk around the sales floor or warehouse carrying a hand-held RFID reader to collect RFID tag information from the merchandise. The whole process usually takes a very long time and there are no definite standards for cycle counting. People usually walk by the shelves and racks and wave the reader. If the reader finds new tags nearby, it will give an audible feedback so that people can hear it and know it is reading new tags. This is a time-consuming and labor-intensive process. After people perform the inventory work, there is only an inventory result from the merchandise, but they still do not know whether they get all the merchandise or the definite location of all the items.

During the past ten years, the autonomous robot has rapidly developed, and huge progress has been made. Autonomous robots are increasingly being used in daily life. Since more and more robots come to people’s lives, such as the vacuum cleaner, more and more people have begun to accept and use robots [3]. In addition to using robots in families, tour guidance and service robots are widely used in banks, hospitals, and exhibition centers [4]. These examples demonstrate that autonomous robots have the capability to deal with more complicated tasks than ever before, and we believe that autonomous robots will serve us better in the near future.

The robots are used for some dangerous tasks, such as detecting objects underground, in the deep sea, or removing bombs which may explode at any time [5], [6], [7]. Robots performing these tasks can greatly reduce the risks of injuring people. Also, robots can finish some time-consuming and dull tasks which humans do not want to do inventory work is typically time-consuming, boring, and labor-intensive task. Therefore, using robots to perform inventory work is of interest to the retail inventory.

An autonomous inventory-taking robot has been previously developed in the RFID Lab at Auburn University [8]. Before the robot begins inventory, the user needs to build a map. Then, the robot navigates the sales floor or warehouse using the premade map to perform the inventory. After finishing the inventory process, the robot can localize the items with the information it read during the inventory. The work has localization errors around 0.5 m. However, this autonomous robot has two drawbacks. The first one is that the map needs to be composed manually and it is of poor quality because the map is somewhat disutorted. The other drawback is that robot cannot access some areas in the warehouse. For example, the robot cannot read tags from certain places and will miss some tags, which causes a low inventory rate.

1.2 Goals and Methods

This dissertation presents two methods to improve the performance of autonomous inventory. The first method is auto-mapping and the second one is multi-layer mapping. For the auto-mapping algorithm, we combined SLAM [9], frontier-edge detection [10] together in ROS [11]. The robot uses SLAM to make a map and frontier-edge detection to set the

destination where it is going. Control algorithms nodes in ROS help the robot move to its destination. If these steps are all repeated, eventually the robot will autonomously create a map. To improve the quality of the map, we utilize the multi-layer mapping algorithm, which is based on SLAM. We modified the original process of SLAM and ran two SLAM processes with Kinect [12] and LIDAR [13] synchronously. Using them separately can avoid the drawbacks of both sensors. The LIDAR accurately estimates pose during the mapping process but it misses most information in the environment because it gets a single 2-D horizontal slice. Kinect can get a 3-D point cloud from the environment. However, it has a very limited angle of view and an unsatisfactory pose accuracy. Therefore, multi-layer mapping uses the information from Kinect and the accurately estimated pose of LIDAR to make a better map. Because we run the two SLAM processes synchronously, they will constantly output the robot's estimated pose. Additionally, the Kinect SLAM process directly uses the position output provided by the LIDAR SLAM process. After the mapping process, we get two maps made with high consistency. These two maps make up the multi-layer mapping result. The mapping algorithms are discussed in Chapter 4.

For the tags which cannot be read by the robot because items are on high shelves or in inaccessible areas, an unmanned aerial vehicle (UAV) is used, which is also called a drone, instead of the gound-based mobile robot to finish the difficult parts of the inventory work. UAVs are rapidly developing and many companies and researchers are working on them. Many commercial UAVs are already in use [14]. Currently, most UAVs are for commercial purpose and entertainment, such as UAV photogrammetry [15] and so on. They have very powerful capacities. For example, a UAV can autonomously follow you while you are driving

down the road. Also, some UAVs have a first-person view (FPV) function so that you can operate the drone while looking at its front camera. This way, you can operate the drone and see scenery high in the sky or deep in a canyon. There are also some UAVs utilized in outdoor inventory [16] and in the agricultural industry (for example, applying pesticide on crops) [17].

Due to the powerful capability of UAV, we plan to use the drone to perform inventory when the robot is unable to do so. For the indoor inventory, self-location is always a big challenge due to the low accuracy of the GPS. Representing the environment with a 2-D camera is also a challenge. Finally, landing on a fixed point and obstacle avoidance prove to be difficult as well. This dissertation will introduce the methods to solve these challenges in the UAV project. We use an algorithm called Parallel Tracking and Mapping (PTAM) to localize the UAV by itself with a single 2-D camera. Meanwhile, we get 3-D map points which can represent the environment efficiently from PTAM. We present a landing algorithm similar to the grid search to make the UAV land on a fixed point with the help of its bottom camera. For the obstacle avoidance, we have designed an algorithm, but have not finished all the experiments yet. We can detect the nearest obstacle blocked in the way of the UAV, but the detour action has not been tested yet.

The principal contributions of this dissertation are: (1) A mapping method automatically executed by robots, making a map with high quality inside a specific area set by humans, (2) a novel mapping method integrating advantages of both Kinect and LIDAR, resulting in a map with higher quality compared with the original SLAM process, (3) a theoretical framework for the control of a UAV as it attempts to land and take off from an elevated

mobile platform, (4) a practical implementation of a UAV launched from a ground-based mobile robot, demonstrating that the theory can be applied to a real-world inventory collection task.

The outline of this dissertation is as follows. Chapter 2 presents a review of the pertinent literature. Chapter 3 presents the background of RFID technology. Chapter 4 presents the principles of SLAM, auto-mapping and multi-layer mapping algorithms, the experimental process and the results. Chapter 5 presents the models of UAV, PTAM algorithm, control methods for UAV, the exerimental process and the results. Chapter 6 presents conclusions and suggestions for future work.

Chapter 2 Literature Review

The topics of robot research contain industrial manufacturing robots [18], educational use of home robots [19], rescue and search robots [20], social service robots [4], home health-care robots [21], inventory robots[8], [22] and many other types. For all these different kinds of robot applications, autonomous navigation is the fundamental function. Before the robot can autonomously navigates, an accurate premade map is critical. In this chapter, we introduce some concepts and researches about mapping methods in robotics.

Also in the past ten years, unmanned aerial vehicles (UAV) have developed rapidly. More and more people accept UAV in their common lives, both for personal and commercial uses. Researchers apply UAV in various fields, such as civil engineering [23], disaster management [24], [25], [26], precision agriculture [27], inventory UAV [28], [29] etc. All these applications in UAVs show that the UAVs now have the capability to accomplish challenging tasks in complicated environments. In this chapter, we focus on some concepts and researches about indoor UAV navigation and UAV 3D mapping and tracking.

2.1 Mapping

Simultaneous localization and mapping (SLAM) is a popular and efficient way to solve the mapping problem. SLAM allows robots to autonomously map in an unknown environment and unknown location. Durrant and Bailey illustrated the SLAM algorithm in detail[9], [30]. The articles mainly introduce the structure of the SLAM problem and explain the evolution of SLAM process with two commonly used methods, extended Kalman filter

(EKF-SLAM) and Rao-Blackwellized particle filters (FastSLAM). The articles also show different kinds of real-world implementations of SLAM. Furthermore, they introduce some parts about the SLAM process which are mostly paid attention by researchers during the past decade, including computation, convergence and data association. Fig. 2.1 shows the essential problem for SLAM: simultaneously estimate both robot locations and landmarks. x represents the position, z represents the observation and u represents the motion update. As it shows, when moving to next point, the robot can use the previous position and motion update to get current position. However, the true locations are never known.

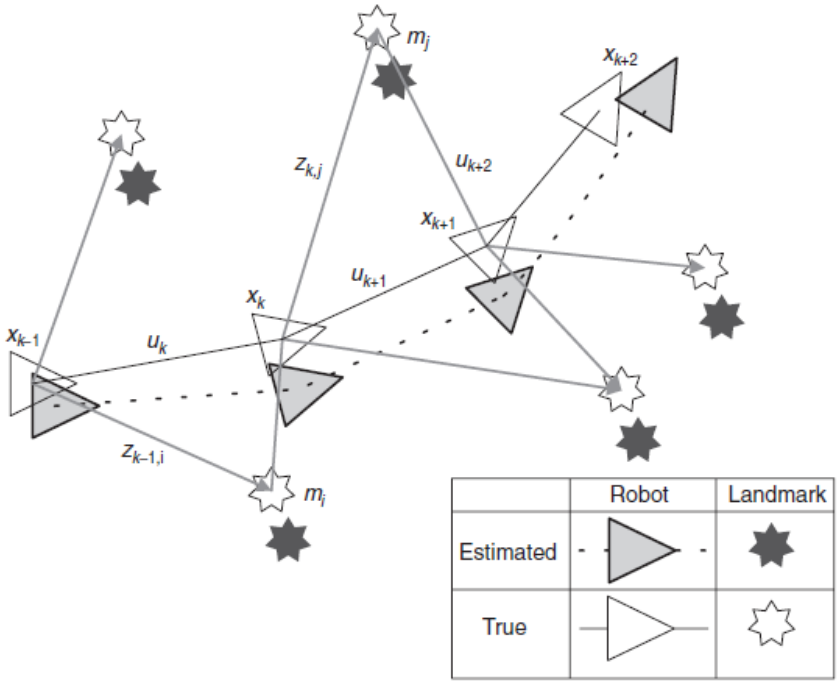


Figure 2.1 Essential SLAM problem

In the mapping process, the most important part is how the sensors represent the environment. The range measurement sensors only provide a contour of the environment because it only scans a pure 2D surface. Therefore, the quality of the map depends on how to represent the environment with an Red Green Blue – Depth (RGB-D) camera. Khoshelham

and Elberink discussed the calibration of the Kinect sensor and analyze the accuracy and resolution of the depth data of the Kinect [31]. Their experiment results show that the random error of depth measurement increases when the distance increases. The maximum random error is about 4cm at the maximum range. The low resolution of the depth measurement also influences the data quality. Their conclusion is that the best mapping distance for the RGB-D camera should be 1-3 m from the sensor.

Henry and his colleagues investigate the method how to apply RGB-D cameras in robotics for building dense 3D maps of the indoor environment [32]. Their RGB-D mapping approach contains three key points: how to place the consecutive data frames; how to detect loop closures; how to consistently complete the data sequence in all. Their experiments are done as follows: a person walked through single-loop indoor environments carrying an RGB-D camera in both daylight and dark situations. The results show that their experiment map is pretty accurate and the quality of the map is very high. However, they only have experimented on single-loop indoor environments. They do not show how their detecting loop closures algorithm works in multi-loop indoor environments.

Endres has done some research on RGB-D SLAM system [33], [34]. He and his colleagues apply SLAM algorithm using RGB-D sensors. Their work shows that they can simultaneously estimate the trajectory of an RGB-D sensor and build a dense 3D model of the environment. They use the RGB-D benchmark for experiments and evaluating their work. The results show that their map has an accuracy of 9.7 cm in average. The results are shown in Fig. 2.2 and Fig. 2.3.

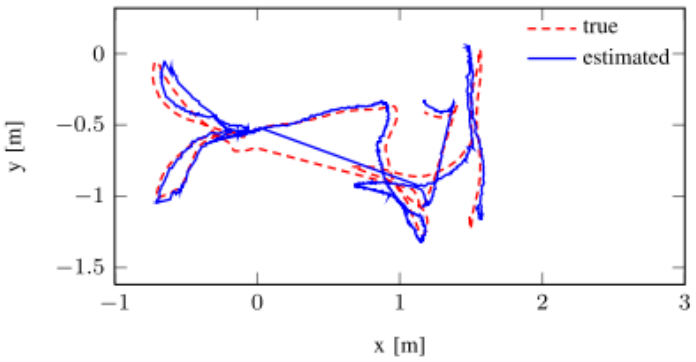


Figure 2.2 Ground truth and estimated camera trajectory in 2D [33]

Figure 2.3 Output voxel grid (in 1cm resolution) [34]

2.2 Mapping algorithms applied in robotics

Arabi and colleagues present a 2D autonomous mapping method for robots [35]. Their robot uses motor encoder, IMU and range measurement sensors as their main sensors. They claim that their robot can autonomously navigate in an unexplored area, avoid the obstacles, collect data and transmit the data to a host computer to make a 2D map. They state that their experiment results show that their robot can complete the process successfully. However, they do not provide enough experiment data or map to support their conclusion. Furthermore, only

using one range measurement sensor to observe the environment is not enough. They may miss a lot of environment information without a camera.

Similarly, Gong also presents an indoor low-cost 2D mapping method for robots based on tiny SLAM [36]. He applies the “tiny SLAM” algorithm on their robot which is made up with a LIDAR sensor and a NVIDIA Jetson tk1 platform. The experiment results show that his map positioning accuracy is about 10 mm-30 mm. This is a very accurate result in mapping although the robot has some problems avoiding obstacles and cannot recover from localization failure.

Basu brings up an autonomous navigation and mapping algorithm mainly based on SONAR [37]. The system is built with a standalone robot and an application used for mapping. The robot can autonomous navigate with the onboard controller and transmit data to the application. He uses Kalman filter to make the sensor data more accurate. The map is made up with 3 SONAR readings. The digital compass in the robot can help with the direction of the robot. The experiment results show that the robot can successfully get a simple map in an uncomplicated indoor environment. They state that their robot can only work in an uncomplicated indoor environment for now because they do not have many sensors data to represent the environment.

Both Huo and Luo apply sensor fusion method for indoor robot mapping [38], [39]. Huo focuses on scan matching for the SLAM process. He uses Kalman filter to improve the pose estimation. The results he gets are closet to the real environment although the effect of the map is not good enough due to accumulated errors. While Luo focuses on detecting moving objects besides SLAM because his algorithm is applied in intelligent service robots. Luo

presents an improved method of graph-based SLAM. The results show that the simultaneous SLAM and moving object detection work well in the office building environment.

A key algorithm for low-cost UAV is the parallel tracking and mapping (PTAM) algorithm which enables the low-cost UAV to do the navigation autonomously [40]. Klein and Murray present a method of estimating pose in an unknown situation. They divide this process into two parallel threads, one for tracking and the other one for mapping. They use a 2D camera to build a 3D map and track the pose in the map. The maximum number of landmarks can be thousands. Fig. 2.4 shows a typical operation in PTAM system. A table is tracked in this photograph. The color dots shown in the graph are the successful observations of the landmarks. This graph contains 660 dots and the frame is tracked in 18 ms. The tracking process contains six parts: image acquisition, camera pose, and projection, patch search, pose update, coarse-to-fine tracking, and tracking quality & failure recovery. The mapping process contains four parts: initialization, keyframe insertion, bundle adjustment and data association refinement. The results show that the system is capable of providing high-quality tracking and reasonably textured maps. The accuracy and robustness of tracking advances the state-of-the-art. However, this system still has two main drawbacks. One is that the system is difficult to operate by the untrained person because PTAM needs a complicated initialization which needs help from the operator. The other one is that it is not good enough to be used in arbitrary environments. It needs some objects to be detected by the PTAM algorithm and then the algorithm can get some keyframes and track the position using these keyframes.

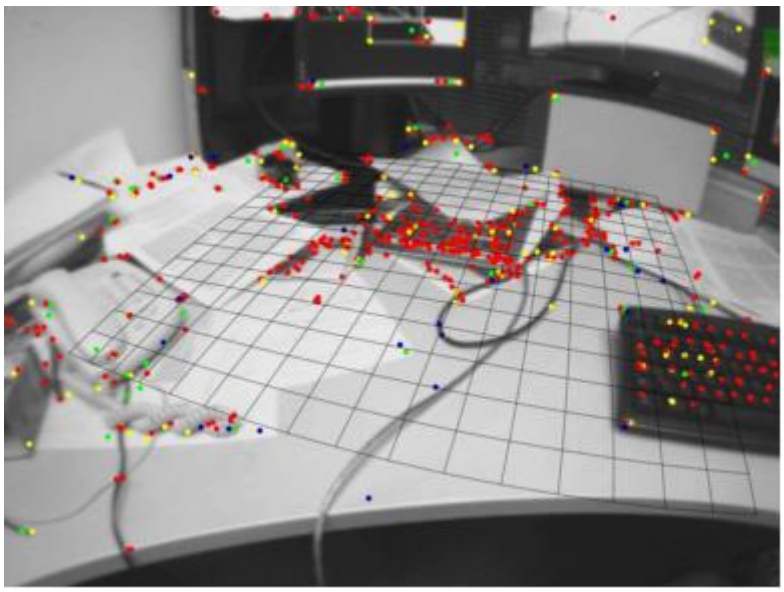


Figure 2.4 Typical keyframes identified by PTAM

Engel and his colleagues present a system that can make a low-cost quadcopter along with a control workstation navigate autonomously in an unknown environment without GPS signal [41]. Their system contains three main parts: a monocular SLAM system, an extended Kalman filter in data fusion and a PID controller for controlling commands. They use AR Drone 2.0 as their quadcopter because it is low-cost and has a good capability. They apply PTAM as their key algorithm for monocular SLAM system. The extended Kalman filter is used for fusing all available data and compensating for wireless network delay. The experiments take place in a series of real-world environments. The results show that the scale estimation accuracy will fluctuate from 1.7% to 5%. The position accuracy varies due to different environments. The root-mean-square error (RMSE) for the kitchen, large indoor room, and outdoor environments are 4.9 cm, 7.8 cm and 18.0 cm, respectively. Their experimental results show that their accurate and robust virtual navigation works well and is reliable.

Xuyu Wang demonstrates an AOA based method, which uses cooperative APs with antenna arrays for accurate indoor localization [42]. This method first estimates the arriving angles for all multipath components using the MUSIC (Multiple Signal Classification) algorithm. Then it exploits the geometric relationship among the angles to identify LOS (Line of Sight) angles. The user location can be computed with the LOS angles and the known distance between the two APs.

In addition to the traditional approaches, machine learning based methods became a hot topic in wireless localization. Multiple researchers [43] [44] [45] proposed a deep learning based indoor fingerprinting system. The system consists of an off-line training phase and an on-line localization phase. In the off-line training phase, the deep learning is utilized to train all the weights of a deep network as fingerprints. Then in the on-line localization phase, a probabilistic method based on the radial basis function is used to obtain the estimated location.

2.3 UAV

Other researchers have published work about indoor UAV navigation. Sa and Corke present a system identification, estimation, and control for MikroKopter [46]. Wang et al report on a UAV equipped with a monocular 2D camera and a range laser sensor. They provide a navigation control algorithm that can make the UAV autonomously navigate in an unknown indoor environment [47]. Teuliere and his colleagues developed a novel 3D model-based tracking for indoor UAV [48]. They use particle filters to deal with the potential poses of the images from the camera. Their experiment result shows the standard deviation of the pose estimation is about 16 cm. Tiemann applies ultra-wideband positioning (UWB)

system in indoor UAV navigation. The RMSE of results is under 10cm with a probability of 95% [49]. Bipin develops an autonomous navigation algorithm for generic monocular quadcopter in natural environments [50]. Ciftler presents a solution to navigating a UAV to a Rayleigh fading source using Q-learning algorithm [51]. Kumar presents an indoor navigation system based on UAVs equipped with LiDAR and IMU in real-time pipeline classification [52]. There are also some outdoor UAV navigation applications [53], [54], [55], [56].

From all this research work about the mapping algorithms, we can tell that SLAM is already a mature technology and applying SLAM in robotics is also a popular method. However, getting high quality maps using SLAM in indoor obstacle-rich environments still needs improvement. The research work about UAV increases rapidly in recent years and many research about indoor and outdoor navigation for UAV are represented. However, these research work does not have many practical results. Research work about indoor navigation is even less. Although PTAM is already presented and is a key algorithm, autonomous indoor UAV navigation still needs more research work to become practical.

My research is based on PTAM and I have improved the performance of its initialization and made the UAV navigating autonomously in sales floor environment. Furthermore, I developed an algorithm to make the UAV land in a specific point with less than 15 cm. Chapter 3 will introduce some essential backgrounds for RFID technology.

Chapter 3 RFID Technology Background

The adoption of RFID technology is rapidly increasing in supply chain management. The motivation of rapid transition to RFID is due to the drawbacks of existing barcode technology [57]. It is not convenient to use the reader to read the barcode because they need to be put in strict requirements of LOS (Line of Sight), especially when a barcode must be read in a giant warehouse environment. In a warehouse, there is a great volume of barcodes, which need to be read manually on a one by one basis. Therefore, it uses an unacceptable amount of time and resources to perform a full inventory in the warehouse. Sometimes, the barcodes on the commodities are folded, soiled, or even ruined. In these situations, the barcodes have very low readability. Another serious drawback is that a barcode does not provide product category information for every item. The products in the same category share the same Universal Product Code (UPC) [58]. In this case, it is difficult to track items or perform inventory quickly, which is currently very important.

To overcome these drawbacks in barcode technology, the retail world turns to an RFID technology. It has been in existence for several decades, but not widely popular due to its high cost and low capability. Recently, the cost has been greatly reduced and the capability has been improved, which has motivated retail employees to widely accept and begin to apply this technology in the retail world. RFID technology is of significant assistance in performing inventory, commodities tracking, and many other applications in supply chain management field. RFID tagging provides item-level Electronic Product Code (EPC) for every single item [59]. RFID technology provides great conveniences in retail inventory and tremendously reduces the amount and cost of manual labor. In this chapter, we summarize the

classification of RFID tags and show the basic model for RFID system.

3.1 Model & Type of RFID System

The basic concept of how an RFID system works is illustrated in Fig. 3.1,

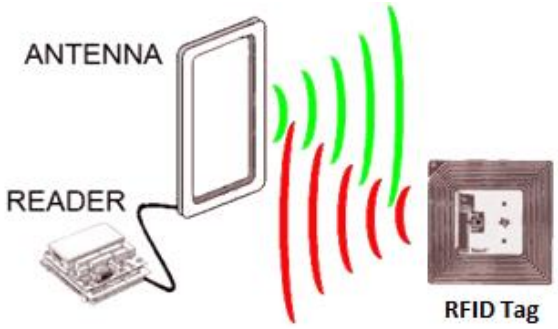


Figure 3.1 A basic RFID system [56]

The reader will send out the radio wave through the Antenna. The antenna may be connected to the reader with a cable, or it may be built into the reader. After the tag receives the radio wave, it collects energy from the interrogating radio waves and acts as a passive transponder. Then the RFID tag will feed back stored information, such as the EPC code.

The RFID system can be classified by frequency or whether the tag has a power source. First, we will talk about the frequency classification. The valid frequency ranges from 100 kHz to 5.8 GHz. These frequency bands set by regulators are available for unlicensed industrial applications. According to the values of frequency, the RFID system can be classified as a low-frequency (LF) system, medium-frequency (MF) system, high-frequency (HF) system, very-high-frequency (VHF) system, or ultra-high-frequency (UHF) system. The most widely used systems are the LF, HF, and UHF systems. The system with a frequency

from 120 kHz to 134 kHz is called the LF system. The system that works on 13.56 MHz is referred to HF system. The UHF system works on a frequency band from 850 MHz to 960 MHz. Currently, most of the warehouse and sales floor retail RFID technology utilizes a UHF system due to its low cost and high capability.

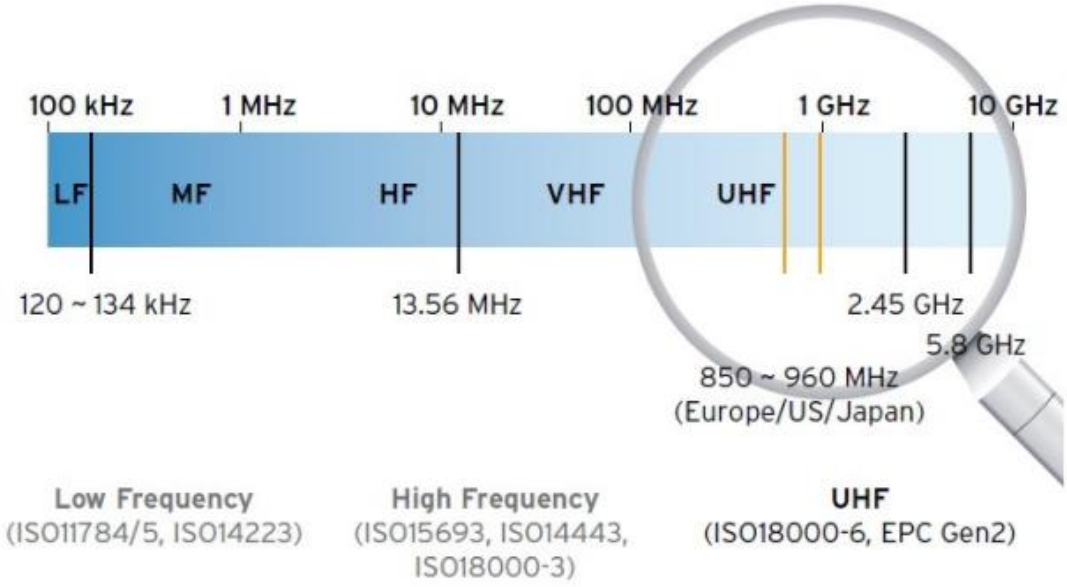


Figure 3.2 Frequency spectrum for RFID [56]

We can also classify the RFID system according to whether the tag has a power supply. Passive RFID systems, semi-passive RFID systems, and active RFID systems are three areas of categorization.

In a passive RFID system, there is no power supply in the tag. Therefore, the tag needs to extract power from the signal sent by the reader. This type of system is the most commonly used because of its low cost and small size. In an Active RFID system, there is an onboard battery in the tag. This battery provides power for the tag transmission signal. This type of system has a very long range and performs well in the obstacle-rich environment. Its

drawback is that it is high in cost and large in size. The semi-passive RFID system is essentially a passive RFID system because it still needs to extract power from signals to give a reply. However, it contains a small battery which provides a power supply for the onboard circuit. Fig. 3.3 shows the basic model for the passive RFID system, semi-passive RFID system, and active RFID system.

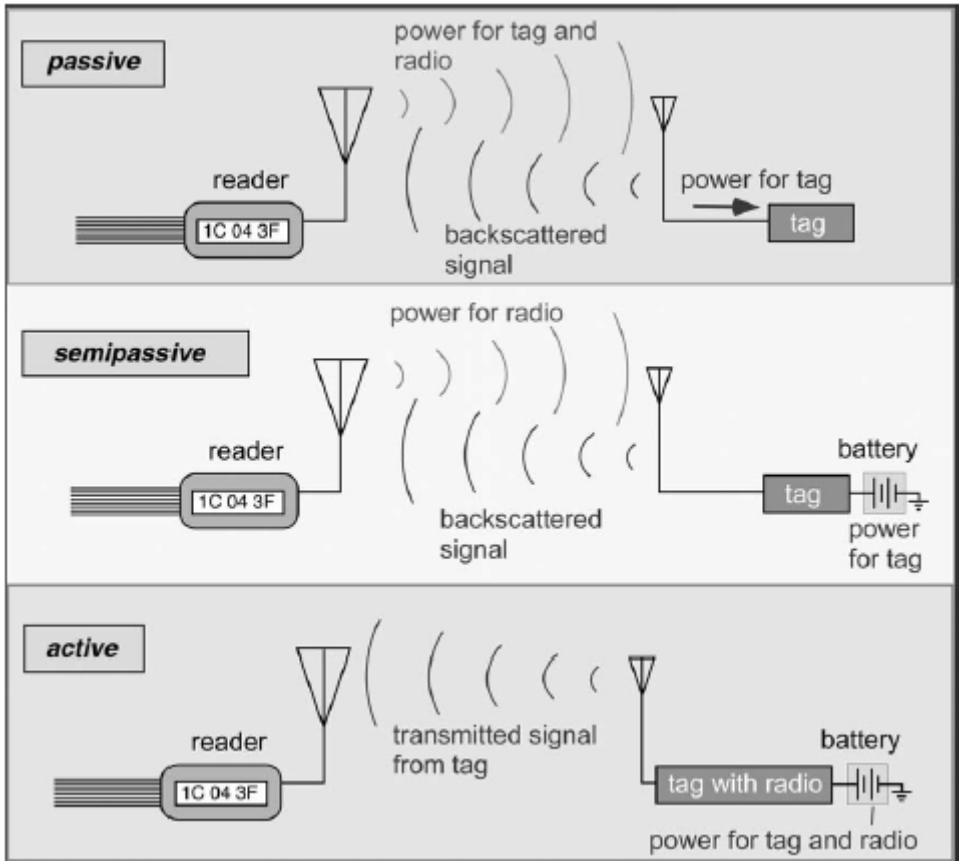


Figure 3.3 Passive, semi-passive, and active RFID system [56]

In this dissertation, we use the UHF frequency passive RFID system [60], which is also the most widely used in the retail world. This is due to its low cost and high reliability. In the rest of this chapter, we will introduce more details about the UHF passive RFID system.

3.2 UHF Passive RFID System

The most significant character for the passive RFID system is that the tag needs to extract power from the signal sent by the reader. Therefore, whether a reader can read a tag depends on whether the value of power received by the tag is enough for the tag to reply. In free space, the signal power will go down as the distance to the destination becomes longer. We can calculate the received power according to the Friis transmission equation shown as equation (1).

$$P_r = P_t \frac{G_t G_r \lambda^2}{16 L \pi^2 d^2} \quad (1)$$

In this equation, P_r is the power received by the tag and P_t is the power sent by the reader. G_t and G_r are the antenna gain of reader and tag, respectively. λ is the wavelength of the signal. L is the loss factor and d is the distance between the reader and the tag. When a reader and tag are determined, the only variable is the distance between them. As the distance increases, the received power decreases rapidly. Therefore, when we need to perform inventory, we need the robot and drone move closer to the tags so they can read more.

However, the calculation discussed above is only an ideal situation in free space. In a real environment, especially on a sales floor and in a warehouse, there are many obstacles, including shelves, racks, tables, and mannequins. All these things can reflect the signal sent by the reader, resulting in waves that propagate and interfere with each other. All of these waves will build a very complex power intensity distribution in the environment. This is called multipath fading. A simple example of a simulation environment is shown in Fig. 3.4 [61].

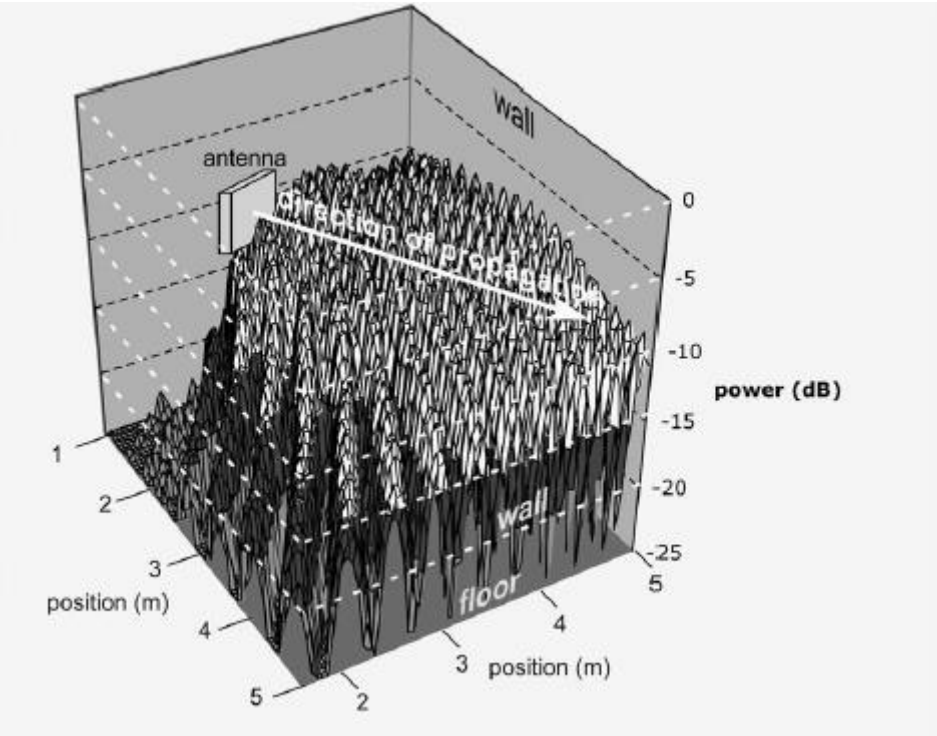


Figure 3.4 Simulation of received power distribution [56]

In this simulation, there is a 5-meter-long cubic room and inside this room, only the floor and walls can reflect the signal. We can see an antenna and its direction of propagation with a white arrow. If there is no multi-path fading, the power intensity distribution should be decreasing continuously while the distance to the antenna increases. However, the practical situation is very different. With only half of the wavelength long, the power intensity could change from maximum to minimum. If we consider the UHF RFID system, we assume the half wavelength is 13 cm. That means if a tag is successfully read in one place, an arbitrary position 13 cm from the tag is unreadable. Then, a tag placed in 13 cm from this unreadable position is read by the reader again.

This simulation is a very simple environment compared with the obstacle-rich environment. Even in this simulation, the power intensity distribution is very complex. Additionally, the power received by the tag is hard to calculate or predict. Therefore, when

we try to localize the tags, multi-path fading proves to be a practical phenomenon which cannot be avoided.

RFID technology are developed rapidly, and my research works are all related to this. Due to its model and specification, an autonomous inventory ground robot equipped with RFID reader is made and applied in sales floor and warehouse. Furthermore, a UAV is also made to offset the incapability of the ground robot; due to the characters of passive RFID system, a tag localization algorithm is made. Chapter 4 introduces the projects related to robotics mapping project, including Auto-mapping and multi-layer mapping.

Chapter 4 Mapping

In order for a robot to navigate, a map is required. A map usually contains most of the environmental information, such as the sizes and localizations of objects in the environment, and the scale and shape of the contour of the whole environment. With this information, the map can help the robot build a global path and localize itself while navigating. Therefore, the quality of the map is a measure of environmental reality. The more accurate the map is, the more information the environment reflects, and the better will the robot perform during navigation. In this chapter, I will first introduce a robust and most popular mapping method in robotics, SLAM. Then two improved methods are introduced: auto-mapping, and multi-layer mapping.

4.1 SLAM

In robotic mapping, Simultaneous Localization and Mapping (SLAM) is a key method used to update a map in an unknown environment while keeping track of the robot's localization [9] [30]. Because the map and robot's localization rely on each other, there is the question of determining which problem to solve first. The key problem in SLAM is to estimate the current pose (position and orientation) for the robot and the map is generated according to all the past control commands and observations. The pose and map probability can be expressed as equation (1):

$$P(x_k, \mathrm{m} | z_{0:k}, u_{0:k}, x_0) \quad (1)$$

In equation (1), x is the pose of the robot, m is the map, z represents the observations, and u refers to the odometry measurements. The x_k means the current pose and $z_{0:k}$ and $u_{0:k}$ means

all observations and commands from time 0 to current time *k*.

The observation is decided by the current pose and the map it sees so the observation model can be expressed as:

$$P(z_k, |x_k, m) \tag{2}$$

Similarly, the motion model can be expressed as:

$$P(x_k|x_{k-1}, u_k) \tag{3}$$

According to the two models, we can get two-step recursive updates:

$$P(x_k, \mathrm{m}|z_{0:k}, u_{0:k}, x_0) = \int P(x_k|x_{k-1}, u_k) \times P(x_k, \mathrm{m}|z_{0:k-1}, u_{0:k-1}, x_0) dx_{k-1} \tag{4}$$

$$P(x_k, \mathrm{m}|z_{0:k}, u_{0:k}, x_0) = \frac{P(z_k,|x_k, m)P(x_k, \mathrm{m}|z_{0:k-1}, u_{0:k}, x_0)}{P(z_k,|z_{0:k-1}, u_{0:k})} \tag{5}$$

If we can solve the conditional probability equation, we can estimate the robot pose and map correctly. There are several known algorithms for solving this. The most popular solutions are using extended Kalman filter (EKF) or particle filter.

For EKF-SLAM [62], the basic idea is shown in equation (6) and (7):

$$P(x_k|x_{k-1}, u_k) \leftrightarrow x_k = l(x_{k-1}, u_k) + m_k \tag{6}$$

$$P(z_k, |x_k, m) \leftrightarrow z_k = g(x_k, m) + n_k \tag{7}$$

Here, *l* and *g* functions are robot and observation models, respectively, while *m* and *n* are additive zero-mean Gaussian errors, respectively. This method needs a huge amount of calculations in every iteration. So, it costs lots of time to deal with the SLAM process. To improve EKF-SLAM, FAST-SLAM comes out. FAST-SLAM [63] is based in recursive Monte Carlo sampling or particle filters. The particle filters cannot be applied directly to the SLAM process because of high dimensional state-space of the SLAM problem. Using

Rao-Blackwell theorem can reduce the sample-space so that Rao-Blackwell Particle Filtering (RBPF) [64] is a solution to solve the SLAM problem. The SLAM probability equation after applying Rao-Blackwell theorem is expressed below:

$$P(x_{0:k}, \mathrm{m}|z_{0:k}, u_{0:k}, x_0) = P(\mathrm{m}|x_{0:k}, z_{0:k})P(x_{0:k}|z_{0:k}, u_{0:k}, x_0) \quad (8)$$

In equation (8), The key idea of RBPF is to separate the SLAM problem into two sub-problems: the first one is pose estimation, and the second one is map updating with the estimated pose. In equation (8), $P(x_{0:k}|z_{0:k},u_{0:k})$ is called the robot path posterior, which represents the pose estimation. In this equation, x means the pose of the robot; m means the map generated by SLAM; z includes the information of observation and u includes the information of odometry; P represents the probability. This is a kind of localization problem and can be solved by the commonly used Monte Carlo localization algorithm. $P(m|x_{0:k}, z_{0:k})$ represents the solution to the problem of generating a map when the pose is already known.

Rao-Blackwell Particle Filtering constructs a collection of particles. Each particle has its own trajectory (a trajectory is represented by a sequence of poses $x_0, x_1, x_2, \ldots x_t$) and a related map. The updating step for Rao-Blackwell Particle Filtering is shown as follows:

- Pose estimating: Each particle estimates its current pose x_t from its last poses (x_0, x_1, x_2, … x_{k-1}) and observation information.
- Map updating: Each particle updates its own map with estimated pose x_t in step one and observation information.
- Weighting: Each particle survives with a weight (also called likelihood) of how well the observations match its own map.

After one updating step is done, the estimated pose and map can be selected from the best-matched particle.

The pose estimation process is based on Monte-Carlo Localization. The process is shown as follows:

- Movement updating: generate from each previous particle to a new particle according to the motion model: $x_k \leftarrow x_{k-1} + u_{k-1}$.
- Observation updating: weigh the particles with the observations likelihood: $w_k \leftarrow P(z_k|m, x_k)$.

The estimated pose can be selected from the particle with the highest weight. In the observation updating step, $P(z_k|m, x_k)$ represents the measurement probability and tells how well the observation z_k matches with the map m when the robot pose is x_k.

The SLAM process loop is shown in Figure 4.1. The first step is the "control update": according to the data from a control command, SLAM can estimate the distance and orientation the robot moves from its last pose. Then it gets information from sensors, such as LIDAR and Kinect. Next, comes the observation update. It will estimate a new pose according to the old pose and the data from the sensors. Finally, it updates the map using the new estimated pose. This completes one iteration.

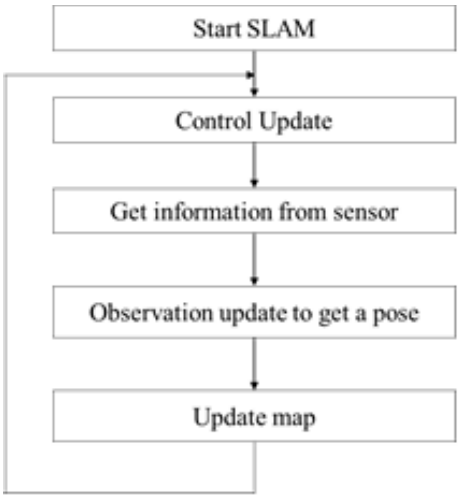


Figure 4.1 Process of SLAM

4.2 Auto-mapping

During the mapping process, the robot is usually controlled by a human with a keyboard or joystick. People drive the robot to cover the whole environment. However, there are some drawbacks of humans driving. First, people only use their eyes to adjust whether the map is missing or completed. Sometimes the map looks good to human's eyes, but people do not know how the robot views the map. Then, people need to avoid their own obstacles when operating the robot. It takes a long time, and during this period, people need to be cautious.

In this case, a new mapping algorithm, called auto-mapping, is proposed. Just as the name indicates, the robot will do automatic mapping using this algorithm. The map in the robot's view is made up of pixels so that the robot knows the missing parts of the complete map. Meanwhile, the robot can avoid obstacles automatically using dynamic window avoidance (DWA) of navigation stack in ROS. Automatically avoiding obstacles is very important, especially when running the robot in obstacle-rich environments such as sales floors and warehouses. Thus, the auto-mapping algorithm solves the problems caused by humans.

The auto mapping algorithm contains three parts: automatically setting a goal, move toward a goal and making a map. The first step of the auto-mapping algorithm is setting goals for the robot. The goal is a location the robot will be traveling to until it reaches the last destination. A goal needs to satisfy two important requirements. One requirement is that the goal needs to be a point to where the robot can move. In an uncompleted map, there are some

unknown areas, which means the robot has not seen them yet and does not know whether or not they are occupied. The other important requirement is that the goal should help the robot to explore unknown areas. That is because the map is not completed until there are no unknown areas. Therefore, the goal should be to help robots detect unknown areas quickly and efficiently.

4.2.1 Frontier-based Exploration Algorithm

The frontier-based exploration algorithm uses grids to represent the evidence in the space. The grids are made up of cells which store the probability that the space the cell represents is occupied. When an observation update appears, which usually means new sensor information from a robot's sensors is read, the grid is updated with an observation model.

All cells in the grid are set to a prior probability of occupancy, which is estimated according to the probability that any position is occupied. When updating the grid, all the cells are classified into three groups according to their occupancy probabilities. The cell with an occupancy probability lower than prior probability is classified into open cells. The cell with an occupancy probability equal to prior probability is classified into unknown cells. The cell with an occupancy probability higher than prior probability is classified into occupied cells.

Just like edge detection [65] in computer vision, the process to detect frontiers finding the boundaries between open cells and unknown cells. In this case, any open cell which is adjacent to an unknown cell is marked as a frontier cell. Furthermore, adjacent frontier cells

are gathered into frontier regions using blob coloring, which is an image segmentation technique. Any region which is more than a prior minimum size is considered as a frontier [66].

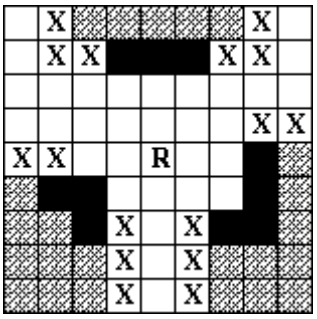


Figure 4.2 Example for frontier detection. An 8-neighbor world is assumed.

The figure above shows a very simple example of how frontier detection works. The location of the robot is marked with R in the grid. These black cells mean they are occupied; white cells mean they are unoccupied; gray cells mean their occupancy is still unknown. According to the frontier detection mentioned above, cells marked with X are the frontier cells.

This algorithm is very suitable for our project. The map in ROS is stored in 2-D point array. The values in the array represent the occupied probability of that position, which is very similar to the background of this algorithm. We only need to modify the prior probability to differentiate the occupied and unoccupied areas. Every time that the robot needs to set a new goal, it gets the current map and robot's pose. Then it applies this algorithm to detect the frontier. After that, it chooses a frontier to be its goal.

4.2.2 Auto-mapping Algorithm

Just as mentioned above, the auto-mapping algorithm contains three parts: automatically setting a goal, moving toward a goal and making a map. The whole algorithm is working in ROS environment. The algorithm keeps getting readings from the robot's sensors (rangefinder and RGB-D camera). Also, it gets odometry from the measurement of encoders that are adhered to the wheels via ROS and a raw map generated by SLAM process.

(1) Automatically setting a goal

First, the raw map needs to be generated from SLAM process. However, we do not directly use the map generated by SLAM. This map is dynamically generated so it has some noise. We use a median filter to diminish the noise in the map and improve the map quality. Then, we set a goal in the new filtered map and calculate the frontier cells using the frontier-based exploration algorithm. After that, we calculate the distance from the frontier cell to the robot, and choose the one which has the shortest distance (so that the robot can explore the nearest unknown area first). When the goal is set, the robot will try to move toward goal. Meanwhile, the next goal will not be set until the first goal is reached. When there are no more unknown areas, we know the map is complete.

(2) Moving toward a goal

After the algorithm successfully sets a goal, the robot needs to move to the goal position. The goal is expressed in the coordinate format as (x, y, Θ) in the map coordinate system. In the moving part, we use the move_base ROS node, which is a major component of the navigation stack [67]. From the figure below, we can see how the move_base node works. The move_base node first inputs a map for navigation. Then it translates this map into a global_costmap and combines this map with the goal and robot pose to obtain a global path.

Meanwhile, it also gets information from sensors and gets a local_costmap . Global path and the local path will eventually result in velocity commands for the motors.

We send the goal generated by the frontier-based exploration algorithm and the map generated by SLAM process to the move_base node. Then move_base will drive the robot to the goal. Furthermore, the robot will avoid obstacles during its movement toward the goal because of the Dynamic Window Approach (DWA) [68]. In ROS, DWA is integrated into local_planner as dwa_local_planner. It works as follows:

1) Sample robot's control space *(dx, dy, dΘ)* discretely.
2) For each sampled velocity, use forward simulation to predict what would occur to the robot if this velocity was applied for a period of time.
3) Evaluate all trajectory results from the forward simulation applied in step 2 (considering the close level to obstacles), close level to the goal, and close level to the global path and speed.
4) Pick the highest-scoring trajectory and send the associated velocity to the mobile base. Discard the trajectory in which collision occurs and select the trajectory with the highest score. Then, send the velocity related to that trajectory to the base controller.
5) Repeat steps 1 to 4.

When the robot reaches the goal, we let it stay there and scan around several times in order to get as much information as possible. With this observation information, we can update and improve the map.

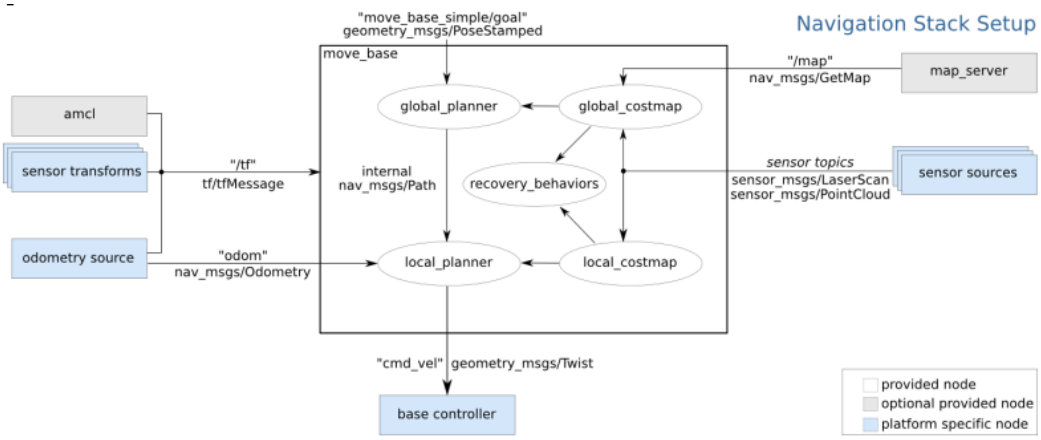


Figure 4.3 Navigation stack in ROS

(3) Making a map

There is an ROS node called slam_gmapping, which provides SLAM capability. With the observation update from a rangefinder and RGB-D camera, slam_gmapping can generate a map in real time. An example map is illustrated in the bottom figure. The map is shown in rviz, a software in ROS which can show the map and the position of the robot in the map. The white areas shown on the map represent the unoccupied areas; the black areas represent the occupied areas; the gray areas represent the unknown areas; and the blue areas represent the inflated space of the occupied areas, which are dangerous for a robot to navigate. The black dot in the left middle position represents the robot.

Also, we can set a specific valid area in rviz so that the robot will only map all the areas inside the valid area. The green polygon is a valid area boundary. We can see the robot only move around inside the boundary and after the robot covers all the areas inside the boundary, it stops and finishes the auto-mapping process.

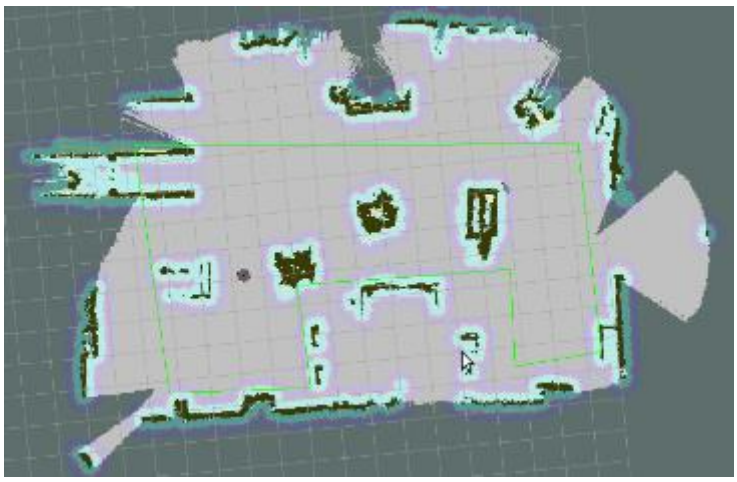


Figure 4.4 Map shown in rviz

4.3 Multi-layer Mapping

The original SLAM process only has one observation input. To combine the strengths of LIDAR and Kinect, merging the information from these two sensors as input is commonly used. With this established method, a map containing some useful information in the environment can be generated. However, this method has two main limitations. Firstly, the information update rate from the two sensors doesn't match well. Kinect has a frame rate of 30 Hz but LIDAR updates at less than 10 Hz. Thus, the map is generated from data obtained at different times. A more serious problem is that the established merging method may lose a great deal of useful information, especially in dynamic and obstacle-rich environments. This is because LIDAR can only scan a 2-D plane, missing obstacles which are not at its height. If a grid cell in the LIDAR map has been marked from "unknown" to "open" (meaning nothing is detected in that grid), this cell will not be changed to "occupied" immediately when the Kinect sees something in it. The SLAM process will probabilistically assume the Kinect observation is noise or a temporary obstacle (such as a person walking across the robot's path)

and still leave that grid as open. Only if the Kinect sees an obstacle for longer than a given threshold of time, will the corresponding map cells be marked as “occupied”. The Kinect can only see a narrow-angle in front of the robot, so it is difficult for the robot to face obstacles from all directions for a long enough time to build a consistent merged map.

To overcome the above limitations, we present a new algorithm named multilayer mapping. This algorithm can effectively use the information from both LIDAR and Kinect, and avoid most problems occur in the merged method. A much more precise and complete map was obtained in a dynamic and obstacle-rich environment, such as a retail store.

4.3.1 Features for Kinect & Lidar

Two of the most common vision-type sensors used for indoor mobile robots are light detection and ranging (LIDAR), and the Kinect. A 3D LIDAR can provide much observation information representing the environment. Due to its vertical FOV, 3D LIDAR gets 3D points from the environment instead of a 2D surface generated by 2D LIDAR. Therefore, a robot with a 3D LIDAR will perform well in navigation and mapping. However, a 3D LIDAR usually costs thousand dollars. Thus, the LIDAR sensors used for cost-constrained indoor robots are 2D – they provide range vs. angle while sweeping at a constant height above the floor. Range and precision are reasonably good, but the single vertical slice limits the utility in a geometrically intricate environment such as a home or store. The Kinect can yield depth information using a structured light (SL) sensor, together with a conventional RGB video image. The SL sensor in the Kinect provides a three-dimensional image of the space in front of the sensor. Each pixel is tagged with depth information, but the range and resolution are

limited. We want to combine the strengths of these readily available, inexpensive sensors to generate a useful map containing information at multiple heights so that mobile robots can perform better in everyday life.

4.3.2 Optimized Process for SLAM

The process of multilayer mapping is shown in Fig. 4.5. The algorithm contains two SLAM processes. The first one is conventional SLAM with inputs from the control commands and LIDAR. Herein, we call this the LIDAR process. This process estimates the robot's pose and produces what we call the LIDAR-layer map. The second process, which we call the Kinect process, will wait until the LIDAR process estimates the pose. After inputting the pose from the LIDAR process, it receives RGB-depth input from the Kinect and registers it to global coordinates to update the Kinect-layer map. The Kinect-layer map and LIDAR-layer map use the same pose to update their maps so they will have high consistency

The LIDAR-layer map is essential for estimating the pose of the robot; the Kinect-layer map is used for navigation. Because the Kinect-layer map is highly detailed, it can help navigate quickly and precisely.

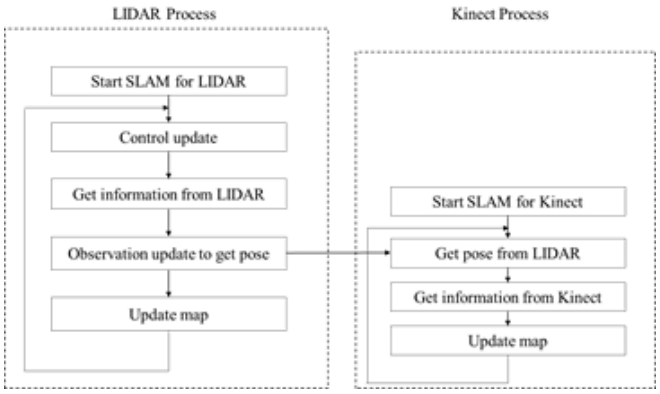


Figure 4.5 Process of multilayer mapping

4.3.3 Multi-layer Mapping Algorithm

As the above description of pose estimating step explained in 4.1 & 4.3.2, observation information is a very important factor in the SLAM process. The more information the observation can provide, the more accurately the pose can be estimated and consequently the more realistic the map can be.

Mapping with the single sensor alone has its own limitation. In our experiment, we use a 360-degree Laser Scanner Development Kit (RPLIDAR). It only provides observation information in a 2D horizontal plane and misses all useful information which is not at its height. As a result, during navigation, the possibility of the robot colliding with certain objects, such as tables which are non-uniform in the vertical dimension, increases greatly. Still, LIDAR can provide relatively accurate pose estimation owing to its long range and nearly full angle of view.

The Kinect can generate depth information using a structured light (SL) sensor and a conventional RGB sensor. What the Kinect gets is a purely 3D image, which is very powerful. Fig. 4.6a is a photo of shoe rack and Fig. 4.6b shows its corresponding depth image read by Kinect. The depth image contains nearly all the information of the shoe rack and is very realistic. However, the 3D image has to be converted to a 2D scan because of limited calculation capability in a cost-constrained indoor robot. Consequently, the observation information from Kinect is ultimately limited in a small range with a narrow-angle. Therefore, the pose estimation from the Kinect has higher uncertainty than LIDAR. In this case, Kinect cannot provide an accurate pose by itself. If we combine the LIDAR and odometry together,

we can get an accurate pose estimation for the robot. However, if we combine the Kinect and odometry together, we still only have an approximated pose estimation for the robot.

In the following section, we describe the novel approach that overcomes the limitations inherent to LIDAR and Kinect sensors and makes use of the strengths of each.

a b

Figure 4.6 (a) photo of shoe rack, (b) depth image of shoe rack

SLAM with LIDAR provides an accurate pose in a fixed external reference frame, sometimes referred to as “global coordinates.” This accurate pose is used for registration of the Kinect image because we cannot get an accurate pose using only Kinect. With the depth information from Kinect and the accurate pose generated by LIDAR, a multi-layer map is obtained. In this map, we will have the accurate pose and all useful information from the environment. In this way, two separate maps with high consistency in coordinates are obtained, which makes up a multi-layer map.

4.4 Experiment & Result

4.4.1 Robot Platform Overview

Our experimental robot system contains three main parts: an autonomous robot, a Wi-Fi

network, and a remote workstation. These are displayed in Fig. 4.7.

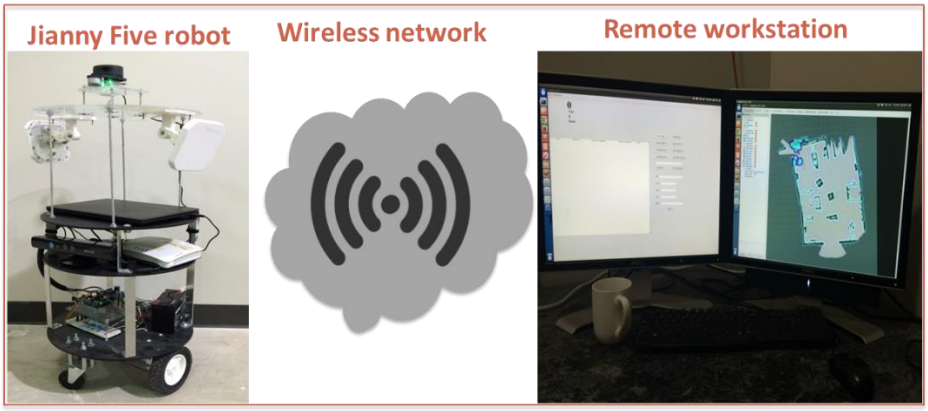


Figure 4.7 Robot System Components

Our autonomous robot is built on a REX-16D Round Robot Base from Zagros Robotics, which consists of two drive motors, two free rotating caster wheels, and three 14-inch diameter ABS plastic shelves for electronics and payload. One additional disk is added to the top of the chassis to mount RFID antennas. The description for the entire robot is shown in Fig. 4.8.

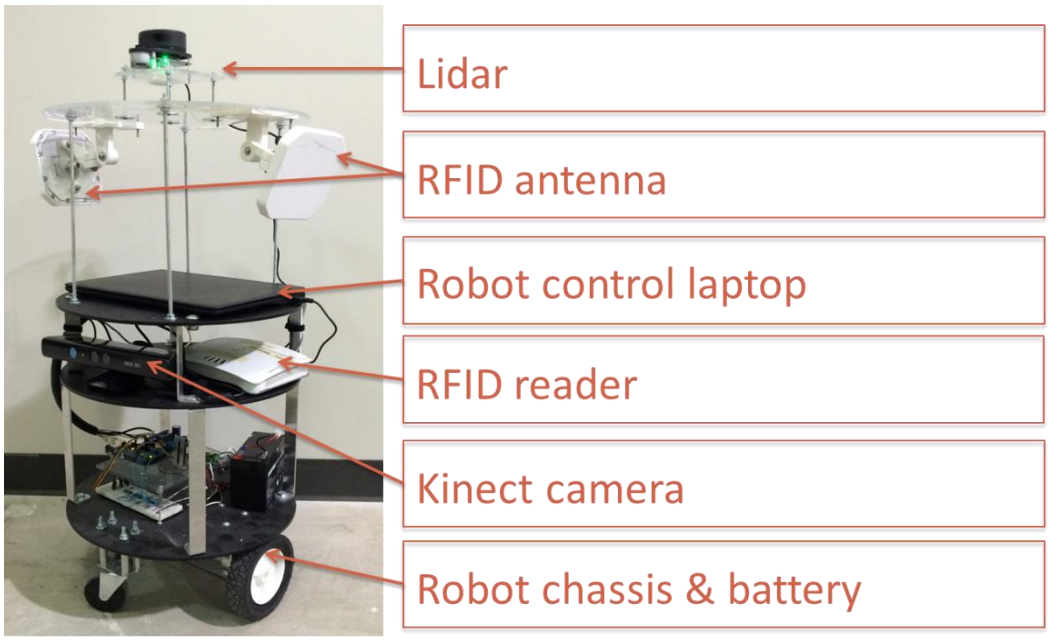


Figure 4.8 Robot Description

a. Raspberry Pi

We use two Raspberry Pi 3 as our robot control center. One is working on sensors input and data processing and the other is working on the robot's navigation. Each one has a 1.2 GHz quad-core ARM Cortex-A53 processor and 1G memory. Additionally, they have a USB interface to an Arduino Mega 2560, which performs low-level motor interface tasks, such as PWM output and wheel encoder input. Due to the built-in Wi-Fi LAN module, they have full capabilities in communication with Wi-Fi. Both Raspberry Pi 3 use Ubuntu as their operating system and the software we use is Robot Operating System (ROS). ROS provides open-source libraries and tools to help software developers create robot applications.

b. LIDAR

For high environment scans, we use the RPLidar A1M1-R3 laser scanner. This sensor can perform a 360-degree scan with a maximum range of 6 meters. The frequency we are

using is 5.5 Hz (although it can be configured to a maximum of 10 Hz).

c. Kinect

For capturing information in a 3-D environment, we use Microsoft Kinect™. It has a very special field of view: 43.5 degrees in the vertical direction, 54 degrees in the horizontal direction, and a range from 0.4 m to 5 m, which is shown in Fig. 4.9. Its frequency is up to 30 frames per second for colorful depth image.

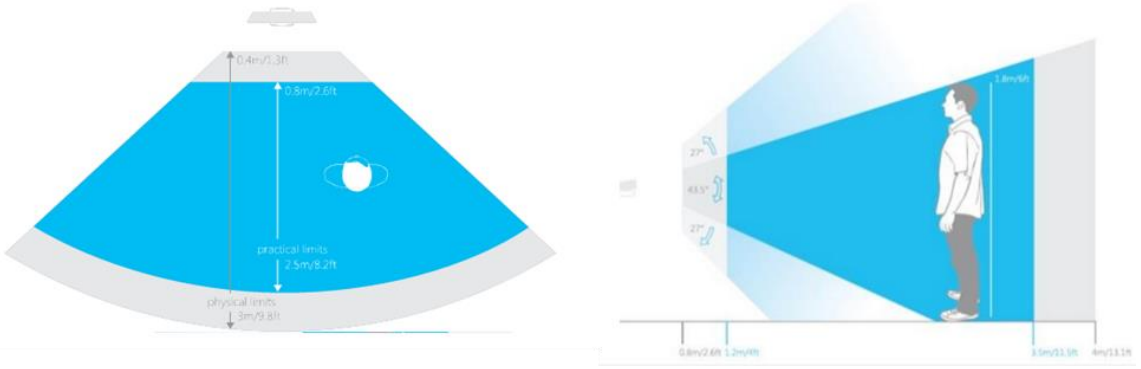


Figure 4.9 Kinect Field of View

d. RFID reader and Antennas

We use a Zebra FX7500 RFID reader and four Zebra AN720 Antennas to read tags. A Zebra FX7500 RFID reader can work on frequency bandwidth from 902 MHz to 928 MHz and from 865 MHz to 868 MHz. It can be connected to a maximum of 8 antennas at the same time. A Zebra AN720 Antenna has a frequency from 902 MHz to 928 MHz and 6 dBi antenna gain. The maximum range of the antenna is about 13 m and the beam width is 100 degrees.

To demonstrate the performance of our mapping algorithm, experiments were conducted using the mock sales floor at the RFID Laboratory at Auburn University. This is an enclosed

region laid out with fixtures and products to closely emulate a retail clothing store. The size is 17 m ×12 m (204 square meters). The sales floor contains several metal shelves, tables, racks, and a seating area as shown in Fig. 4. There is a total of 674 items with RFID tags on the sales floor, including 415 jeans and khakis, 26 pairs of pants, 149 T-shirts, 21 dresses and tops, 30 pieces of lingerie, and 33 pairs of shoes. The shoes are placed on a shoe rack which is No. 8 shown in Fig. 4. 10b. The lingerie is placed on a lingerie table, which is No. 2 in Fig. 4. 10b. Jeans are placed on two shelves, a metal shelf near the wall and a wooden shelf in the middle, which is No. 1 and No. 5 in Fig. 4. 10b, respectively. T-shirts are hung in racks in No. 2 and No. 3 in Fig. 4.10b. Pants and tops are placed in racks in No. 6 and No. 7 in Fig. 4. 10b. The dresses are hung in racks too in No. 9 and No. 10 in Fig. 4. 10b.

a b

Figure 4.10 (a) experiment sales floor, (b) blue print map

4.4.2 Auto-mapping

For the auto-mapping experiment, we put the robot in an arbitrary position on the sales floor. Then, we start all the programs we need and open the rviz in ROS to observe the map.

The initial map is shown in Fig. 4.11. The black circle in the middle represents the robot. The gray area represents the unoccupied area; the black area represents the occupied area and others are the unknown areas.

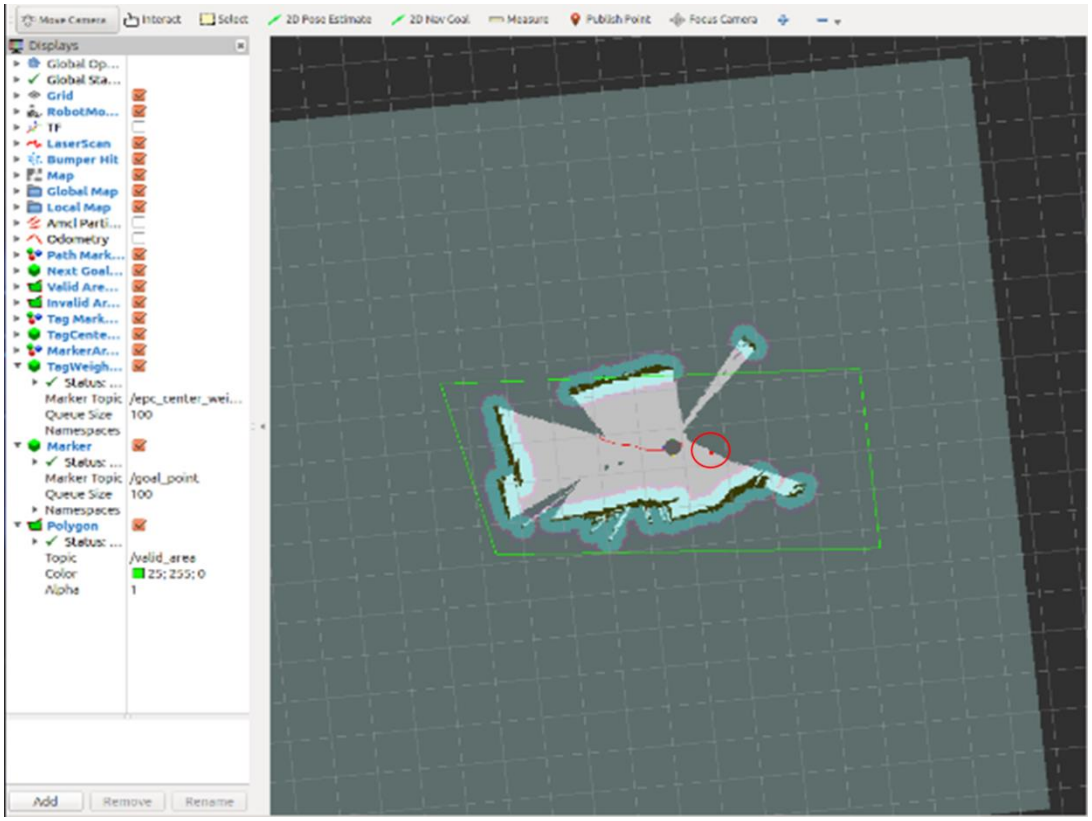


Figure 4.11 Initial map for auto-mapping

For initialization, we will first need to set an area for the robot to map. We can click several vertex points and they will make up a polygon, such as the green rectangle in Fig. 4.11. The number of clicked vertex points is an arbitrary number greater than two. The red point in Fig. 4.11 represents the next goal point set by the auto-mapping algorithm. The red line represents the moving trace of the robot. The position of the current robot is the first goal and the other side of the red line represents the robot's initial position. It moves from the initial point to the current position and then moves to next goal, which is represented by the

red point position.

The entire process involves repeating the above steps. First, set a goal and then move toward it. After updating the map, set the next goal, and move toward it until there is no unknown area inside the map of the pre-set polygon.

The result is shown in Fig. 4.12. The red circle is the initial position for the robot. The green polygon is the pre-set mapping area. We can see that the robot has finished almost all of the area inside the polygon excluding a small area in the top-left side. The robot finishes auto-mapping because, according to the algorithm, it cannot find any goal point. The robot cannot go into the unknown area in the top-left side because it coincides with the occupied area in the pre-set polygon.

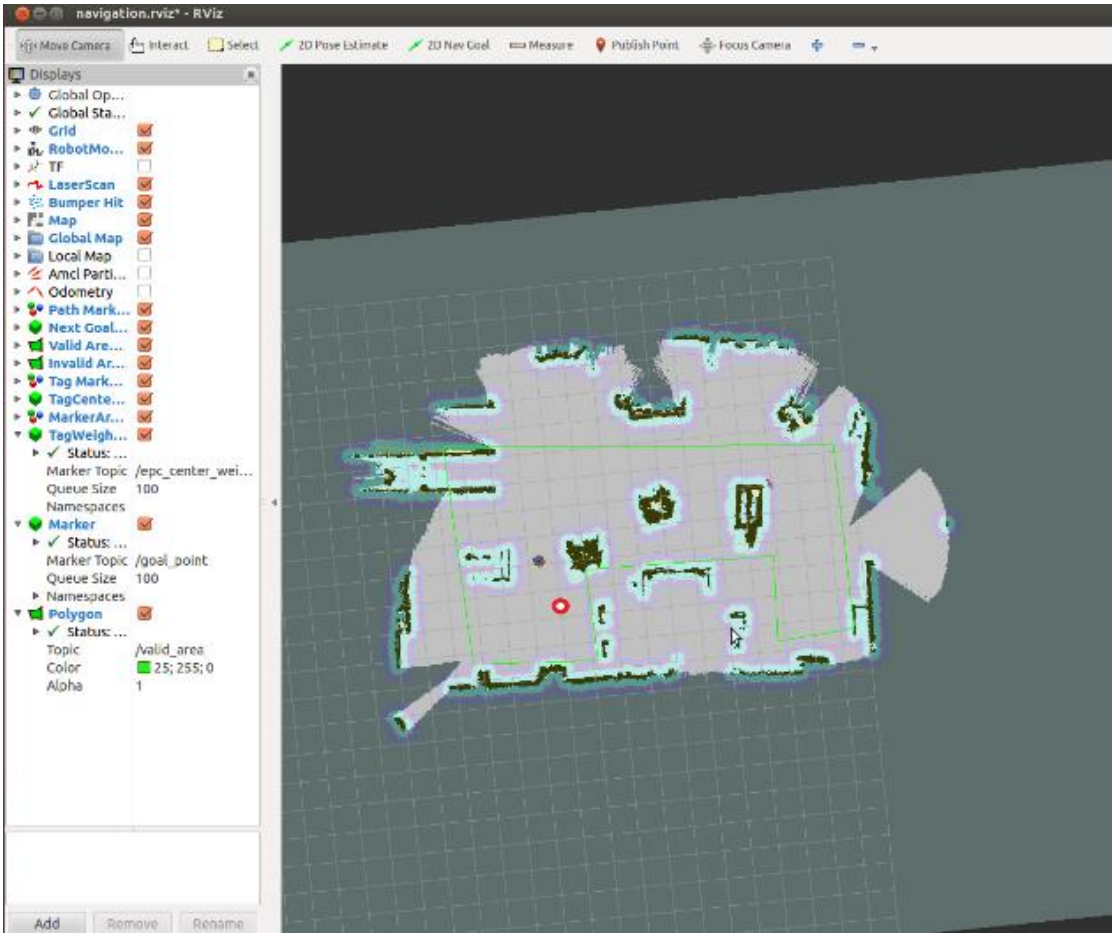


Figure 4.12 Finished auto-mapping map

We can see the accuracy of this map when compared with the blueprint map of the sales floor. To display it more clearly, the map has been rotated 90 degrees, which is shown in Fig. 4.13.

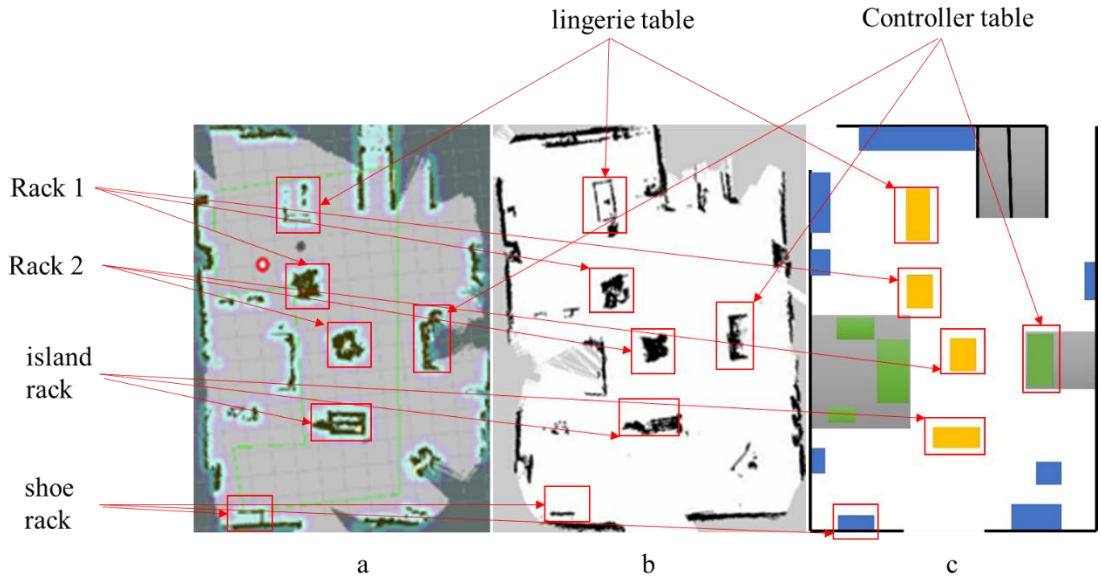


Figure 4.13 (a) map made by auto-mapping, (b) map made by human, (c) blueprint map

Compared with the blueprint map, the two result maps are slightly bent. We can see that the left and top walls are not straight in comparison to the other two walls. Then, we can compare the position and size of several important obstacles in the environment. They are the lingerie table, the controller table, racks 1 & 2, the island rack and the shoe rack. The red rectangles for the same obstacle are the same size within all the maps. After this, we can determine whether the size is correct. Except for the lingerie table, the sizes of all other obstacles are very similar to the real sizes.

Table 1 The comparison of errors of maps made by human & auto-mapping

Obstacle in blueprint map	**Lingerie table**	**Rack 1**	**Rack 2**	**Island rack**
Error of map made by human(m)	0.623	0.093	0.401	0.419
Error of map made by auto-mapping (m)	0.428	0.091	0.393	0.298

For the position accuracy, we calculate four main obstacles in the environment. The

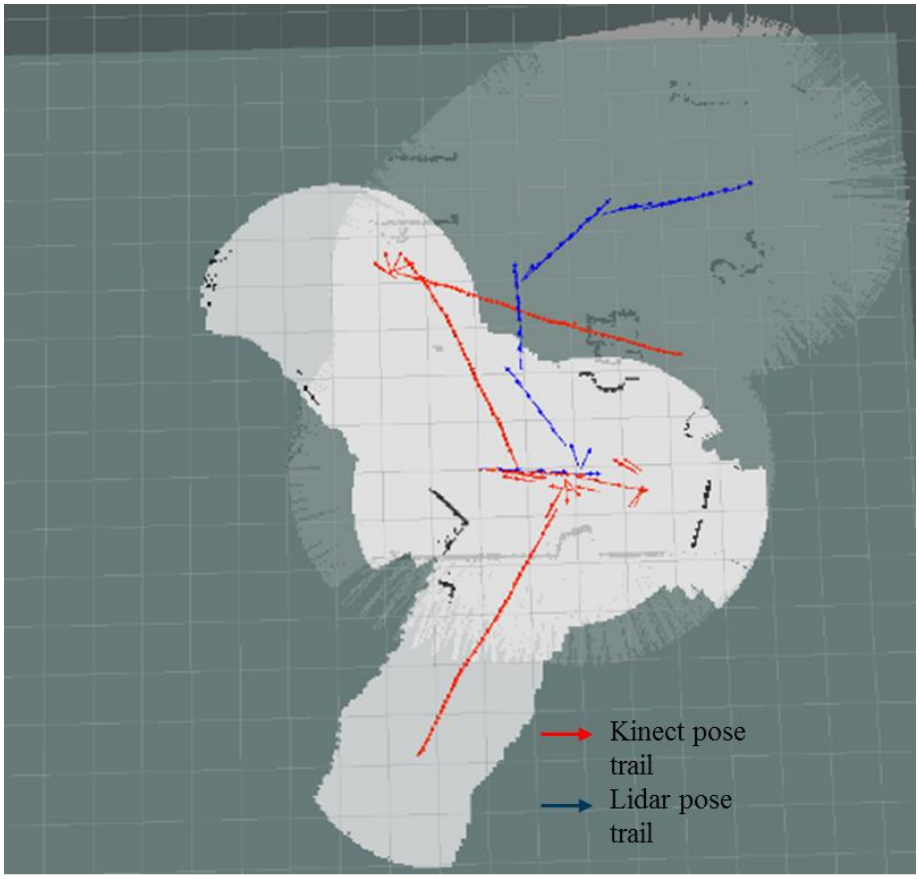
position errors of the human-made and auto-composed map are compared with the blueprint map. The result is shown in table 1. We can see that, in general, the position of the map made by auto-mapping is more accurate than the human-made map made. The position of the table is the most inaccurate one of all. The positions of rack 1 for both maps are the most accurate. We have tested both maps using them for the robot navigate, and the robot successfully finished the navigation using both maps. Each of them has a certain accuracy and are useful for navigation.

4.4.3 Multi-layer mapping

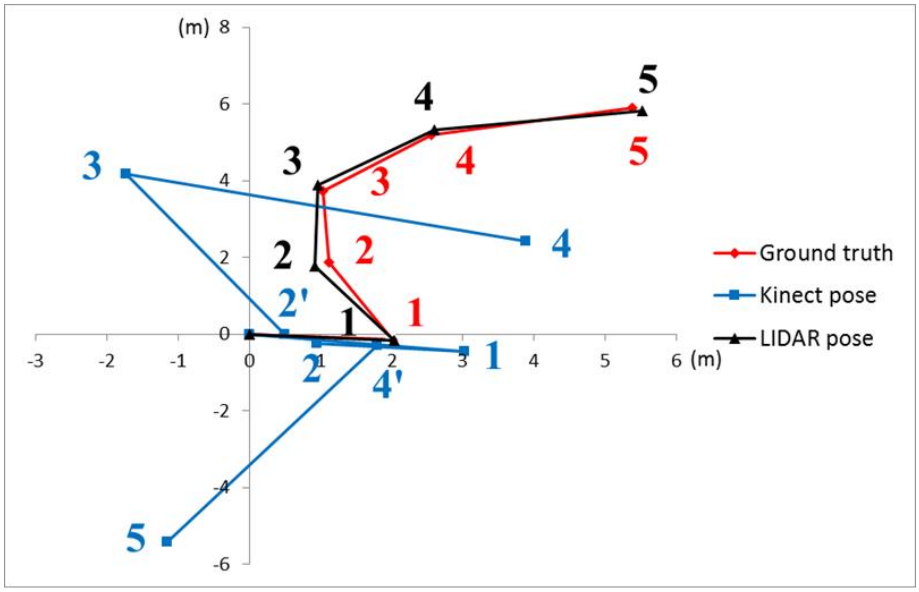
The experimentation process is more complicated for multi-layer mapping. We first start all the programs and map the sales floor with only the Lidar. After finishing this map, we repeat the mapping process with only Kinect. For the third time, the mapping process is repeated with both sensors with original SLAM method using the merged-sensor algorithm. For the last time, we repeat the mapping process with our multi-layer mapping algorithm.

Fig. 4.14a is a screenshot of SLAM being utilized in ROS. Fig. 4.14b shows the sampling points of pose estimation trails of Fig. 14(a) in sequence. As mentioned above, the Kinect cannot provide a very accurate pose due to its limited observation information, while LIDAR can provide relatively accurate pose estimation due to its long range and nearly full angle of view. According to Fig. 4.14b, the pose estimated by LIDAR is notable because, when the robot changes directions, the Kinect may give a wrong pose. When the robot turns around, the observation information changes significantly. In such cases, it is difficult for particle filters to match the right part in the map, which results in sudden pose changes in the

Kinect pose trail of Fig. 4.14b, such as point 2 (2') and point 4 (4'). Point 2' and Point 4' represent the sudden changes in point 2 and point 4, respectively. As for LIDAR, much smaller errors are made in turns, and can be controlled in a certain range due to the good field view of the LIDAR. The errors calculated for all five sampling points are listed in Table 1. We can directly see the major difference in pose estimation accuracies of LIDAR and Kinect.



a



b

Figure 4.14 (a) Lidar and Kinect pose estimation trail while mapping, (b) positions comparison of Lidar, Kinect, and ground truth

Table 2 Pose estimation errors of LIDAR and Kinect

Sampling point	**1**	**2**	**3**	**4**	**5**
Kinect pose error (m)	1.066	2.131	2.813	3.088	13.06
LIDAR pose error (m)	0.044	0.223	0.174	0.134	0.154

Fig. 4.15 shows the maps generated by SLAM with LIDAR (LIDAR-map, Fig. 4.15a) and by SLAM with Kinect (Kinect-map, Fig. 4.15b), respectively. Each map has its advantages and disadvantages. The LIDAR-map is precise but sparse. The LIDAR used for the cost-constrained indoor robot is purely 2D, such as a RPLIDAR 360-degree Laser Scanner in our experiment. This LIDAR can provide long-range 2D plane information. It is very precise and covers a large area. However, with limited vertical thickness, it can only give a very simple profile of the environment and misses a large amount of helpful information. In comparison, the Kinect-map is somewhat distorted. Much less observational information was provided by Kinect due to its limited view field and its inability to match well with the corresponding parts in the map. This may result in the wrong self-location of the robot. Therefore, the map updated by Kinect is highly distorted. However, the Kinect-map still contains a lot great deal of useful information. Most of the detailed information contained in the Kinect-map, such as the lingerie table, the island rack and the shoe rack, does not appear in the LIDAR-map in Fig. 4.15.

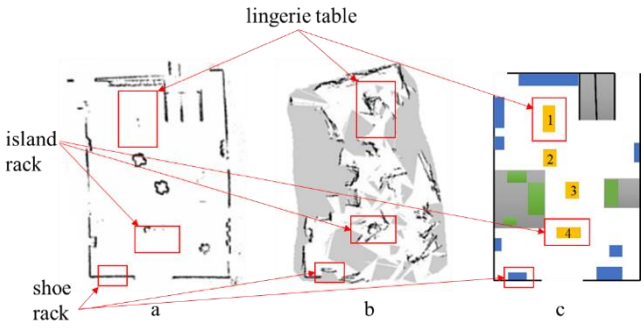


Figure 4.15 (a) LIDAR-map, (b) Kinect-map(c) Blue print map

After using the information from both sensors, better maps are composed, as shown in Fig. 4.16. Fig. 4.16a shows the map resulting from the conventional merged-sensor approach and Fig. 4.16b shows the map resulting from our new multilayer method. The latter is referred to here as the Kinect-layer map. There are four main obstacles shown in Fig. 4.16c, labeled 1 to 4. We measured the actual positions of these obstacles and compared them with the positions obtained in the merged-sensor map and in the Kinect-layer map. The results are shown in Table 2. For most obstacles, the position in the Kinect-layer map is better than that of the merged-sensor map. The exception is obstacle 2. For obstacle 2, both errors are very small. For other obstacles, the error in the Kinect-layer map is around 0.3 m and is greater than 0.4 m in the merged-sensor map. The accuracy of obstacle position is highly important for a map and represents the accuracy of the entire map. Generally speaking, the Kinect-layer map is better than the merged-sensor map. Position accuracy is only one map feature that is improved by our multilayer mapping process. Next, we will discuss the detailed advantages and disadvantages of multilayer versus merged-sensor mapping.

Table 3 The errors of obstacle positions in Merged-sensor map and Kinect-layer map

Obstacle in blueprint map	**1**	**2**	**3**	**4**

Merged-sensor map error (m)	0.623	0.093	0.401	0.419
Kinect-layer map error (m)	0.322	0.150	0.350	0.257

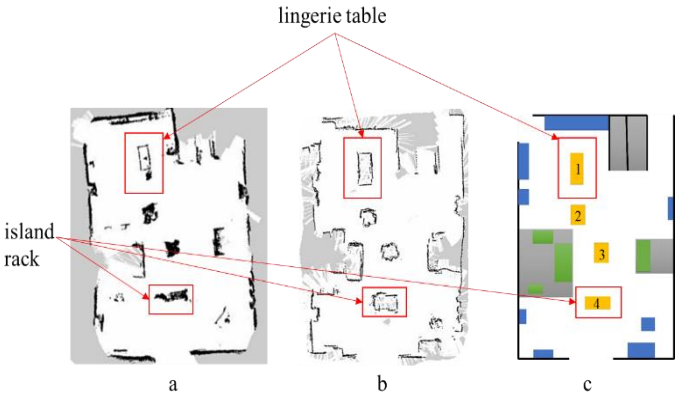


Figure 4.16 (a) Merged-sensor map, (b) Kinect-layer map, (c) Blue print map

Although the merged-sensor map contains roughly all the information in the environment, several problems remain. Firstly, the walls are skewed (especially the left and right walls). Also, the color of contours in the merged-sensor map is very dark. That is caused by the inaccurate self-location. When SLAM updates a map, it first locates the robot itself. Particle filters run to estimate the robot pose by using a genetic type mutation-selection particle algorithm to implement the prediction. However, because of the merging method, much useful information is lost, and the particle filter will give an inaccurate pose. Based on this inaccurate pose, the updated map is somewhat distorted. As for the lingerie table, due to the inaccurate estimation of the robot pose, the contour of the table is wrongly placed. Multiple slightly wrong contours will make the lingerie table, the island rack and the walls dark.

Secondly, the shapes of obstacles (i.e., the lingerie table above) can only be seen if the

Kinect views the table repeatedly and can mark the corresponding grid cell on the map as “occupied”. Moreover, it is difficult for the robot to face the table from all directions on a continual basis. As a result, the lingerie table and island rack can only be seen in part.

Compared with the merged-sensor map, the multilayer map has a higher consistency (well-registered) with the blueprint map. The walls (boundaries of the whole room) of the multilayer map are very straight, and the profiles of obstacles are much more accurate and clear. The use of LIDAR pose reduces the distortion of the map, and Kinect provides the information of multiple heights, preserving all the useful information. When updating the map, the robot updates the observation information surrounding the pose and can compose a precise and complete map. The robot will perform much better using the multilayer algorithm because it will be more precise while navigating.

Fig. 4.17a shows the Kinect-layer map made in another sales floor. There are a table and a rack on the top and three big shelves on the left side and in the middle of the map. The bottom side has some debris piles, which are marked in gray. These areas are inaccessible to the robot. We can find most of these obstacles in Kinect-layer map. Then, we compare the positions and sizes of three shelves and a table & rack. For the resolution, we think shelf 1 is the most accurate obstacle in the map so that we calculate the resolution according to its length in map and in real world. For the position, we mark the left wall and top wall as the standard lines. Calculate the distance of obstacles to both lines and then calculate for the errors. For the size error, we measure the length and width of the obstacles and calculate for the area. After that, compare the area error in percentage. The results are shown in Table 4.

Table 4 The errors of obstacle positions and sizes

Obstacle	**Shelf 1**	**Shelf 2**	**Shelf 3**	**Table & rack**
Position error (m)	0.149	0.235	0.645	0.129
Size error (area)	2.2%	4.3%	1.6%	17.6%

We can find that the obstacle positions are very accurate except for the shelf 3 and the obstacle sizes are also pretty accurate except for the table & rack. Both inaccurate parts are influenced by the wall which is a little bit distorted. Therefore, when comparing with the blueprint map, the distortion makes the distance to the standard lines smaller, so the results are not very accurate. But the general obstacles positions and sizes error are pretty good and we can say that the multilayer algorithm works well in the different environment, and the Kinect-layer map is highly accurate compared with the blueprint map.

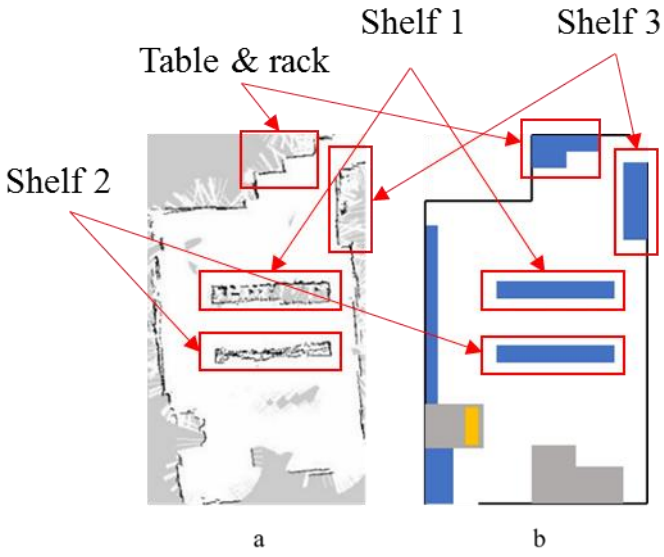


Figure 4.17 (a) Kinect-layer map, (b) Blue-print map

Chapter 5 Unmanned Aerial Vehicle (UAV)

In RFID inventory, utilizing the robot is an effective method, which has already been done. However, it has two fatal limitations. One is that robot cannot make sure it can cover 100% of the area in the working environment, such as a retail floor or warehouse. Sometimes, there may be some obstacles in such environments. In this case, the robot cannot move through the obstacles while there are some commodities which are needed to be read beyond the obstacles. At this situation, what the robot can do is only trying to read them out of the obstacles' range which will result in missing many RFID tag readings. Another limitation is that the robot cannot read much information when the tags are placed too high because the energy received by those tags is too low. Meanwhile, the position of the antenna is fixed so it cannot rise. Therefore, our robot cannot rise and only move on the ground.

Due to the limitations described above, we propose an innovative inventory method using UAV to autonomously perform inventory and passive RFID tagged items in complex environments, such as warehouses, retail stores, and factories. The drone takes off from a specific position and moves in a preset routine to investigate all passive RFID tagged items. During this process, the drone will move in different altitudes trying to cover the whole space which contains passive RFID tagged items. After the reading process, the drone will move back to the given take off position.

5.1 UAV Model

The quadcopter is the example type of UAV model we are discussing in this section. Quadcopters are popular and useful tools which allow researchers to attempt new ideas in

different fields, especially flight control and robotics. There are many advantages of using quadcopters as test platforms. Their prices are affordable and they are available in a variety of sizes and models. Also, the quadcopters can be easily built and maintained by amateurs due to their simple mechanical design [69].

The quadcopter is modeled with four rotors which are configured in a cross. The cross is linked mechanically with the motors so that the whole structure is robust. The four propellers are connected to four motors through reduction gears. The four propeller axes of rotation are parallel and fixed. The pitch blades are also fixed. The air flows downward of two propellers and gets an upward lift of the other two propellers. We can find that most of the parameters in the drone are fixed so the only thing which is flexible is the speed of the propellers. The method to control the drone to perform different actions involves controlling the speeds of the four motors.

Figure 5.1 shows the structure model in hovering condition, where all of the propellers have the same speed [70]. Each rotor generates a thrust and a torque around its center of rotation, along with a drag force going in the opposite direction of the drone. If all four rotors are spinning at the same angular velocity, just as Figure 5.1 shows, with rotor one and three spinning counter clockwise while rotor two and four spinning clockwise, the angular acceleration of the z axis is exactly zero. This is why the quadcopter does not need a tail rotor.

We can see a quadcopter model in Figure 5.1. The black lines represent the contour of the quadcopter. The bending blue lines represent the spinning directions of the propellers, and the straight blue lines represent the thrusts generated by the spinning propeller. The blue sign

Ω represents the angular velocity of the propeller. The green coordinate represents the drone frame coordinate. As the coordinate shows, the front of the quadcopter is the positive x axis; the left of the quadcopter is the positive y axis; the top of the quadcopter is the positive z axis.

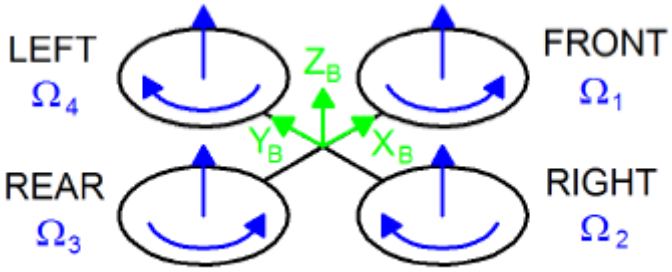


Figure 5.1 Simplified quadrotor motor in hovering

The four basic quadrotor movements are shown as follows:

- Throttle

The throttle is executed by increasing or decreasing the four propellers' speeds with the same values. This will result in a vertical force, which will raise or lower the quadrotor. Figure 5.2 shows the throttle execution in a quadcopter model. We find that all four angular velocities increase with the same amount Δ_A so that the change of force which the four propellers generate is marked with red sign $\ddot{Z}$. This means if the velocities of all four propellers increase by the same amount, the quadcopter will rise. On the contrary, if the velocities of all four propellers are decreased by the same amount, the quadcopter will fall.

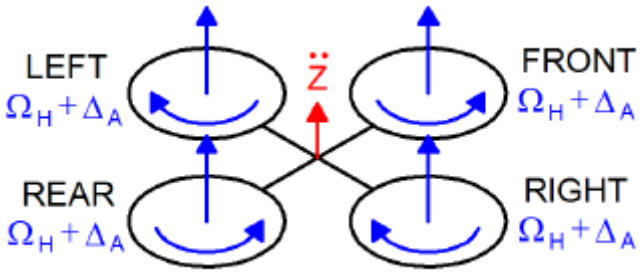


Figure 5.2 Throttle movement

- Roll

Roll is executed by increasing or decreasing the left propeller speed and by decreasing or increasing the right one. It will result in a torque in the *x* axis which makes the quadcopter turn. The entire vertical thrust is still the same as hovering because roll only results in a roll angle acceleration. Figure 5.3 shows the roll execution in quadcopter model. The velocity of the left propeller is increased by amount Δ_A and the velocity of the right propeller is decreased by the same amount Δ_A. The entire thrust is still the same, but the left thrust is increased and the right thrust is decreased. This will result in a turn action in the *x* axis of the quadcopter.

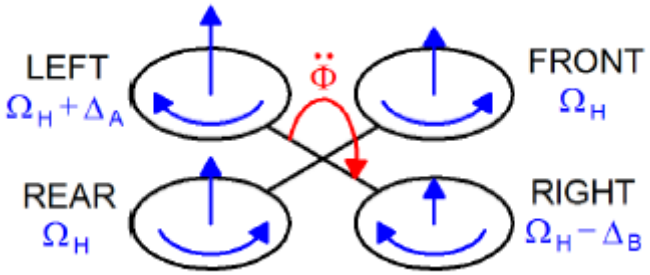


Figure 5.3 Roll movement

- Pitch

Pitch is very similar to roll and is executed by increasing or decreasing the rear propeller speed and by decreasing or increasing the front one. It will result in a torque in the y axis,

which also makes the quadrotor turn. The entire vertical thrust is still not changing because pitch only results in a pitch angle acceleration. Figure 5.4 shows the pitch execution in the quadcopter model. The velocity of the rear propeller is increased by amount Δ_A and the velocity of the front propeller is decreased by the same amount Δ_A. The entire thrust is still the same but the rear thrust is increased and the front thrust is decreased. This will result in a turn action in the *y* axis of the quadcopter.

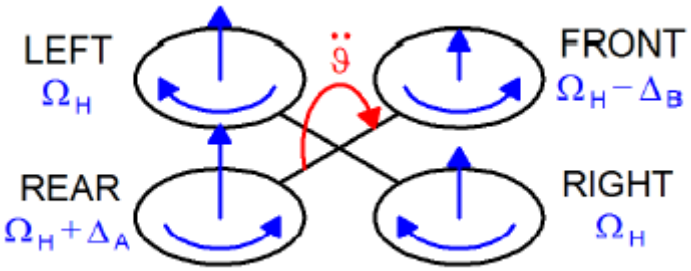


Figure 5.4 Pitch movement

- Yaw

Yaw is executed by increasing or decreasing the speeds of the front and rear propellers and by decreasing or increasing the speeds of the left and right propellers. It will result in a torque in the z axis, which also makes the quadrotor turn. Because the left and right propellers rotate clockwise while the front and rear propellers rotate counterclockwise when the entire torque is not balanced, the quadcopter will turn around the z axis. The entire vertical thrust still does not change because yaw only results in a yaw angle acceleration. Figure 5.5 shows the yaw execution in a quadcopter model. The speeds of the left and right propellers are increased by the same amount Δ_A and the speeds of the rear and front propellers are decreased by the same amount Δ_B. The quadcopter will turn around the z axis, according to the unbalanced torque.

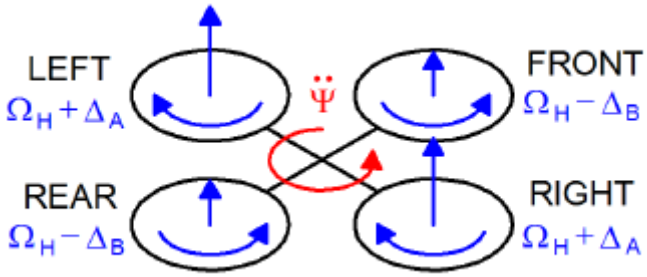


Figure 5.5 Yaw movement

5.2 PTAM

Parallel Tracking and Mapping (PTAM) is an algorithm to estimate a 3D pose with a 2D camera in an unknown environment [40]. The whole process is divided into two separate and parallel parts: the tracking process and mapping process. The tracking process will continuously track the position of the camera and the mapping process will build a map with the information representing the environment captured by the 2D camera.

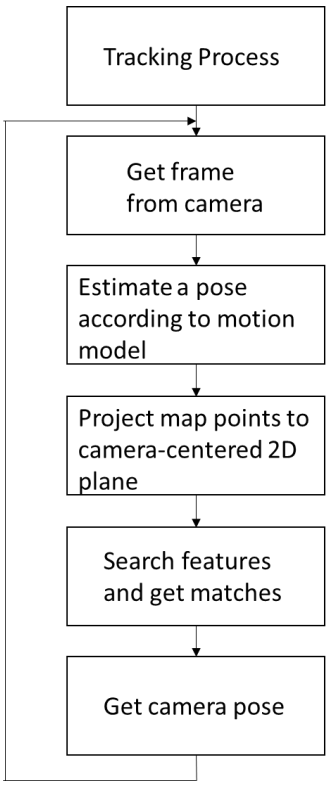


Figure 5.6 Flow Chart for Tracking Process

-Tracking Process

The tracking process is a point-based system with a pre-made 3D map. A 2D image is the input for this process and the process keeps a real-time estimate of the camera pose relative to the 3D map. The detailed process involves first getting a frame from the camera in order to estimate a pose from the motion model (which uses the velocity from the last pose). Then, project the map points on the image with the estimated pose. Next, search for the features in the image and finally update the map according to the

matches. The process is shown in Fig. 5.6.

To project the map points in the image, one first needs to transform from the world coordinate frame to the camera-centered coordinate frame shown in equation (1). *W* means the world coordinate frame and *C* means the camera-centered coordinate frame. *P* means a point and p_{jw} can transform to p_{jc} by left-multiplied by E_{CW}. *E* represents a 4×4 matrix, which represents the camera pose. *CW* means frame *C* from frame *W*. It contains both the rotation and translation components. The essential matrix helps to determine the rotation and translation from one point to the next point by comparing the different positions of a same 3D point in different camera-centered 2D planes.

$$p_{jC} = E_{CW} p_{jW} \tag{1}$$

To project camera points to the image, a calibrated camera projection model is used.

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \text{CamProj}(E_{CW} p_{iW}) \tag{2}$$

The pin-hole camera projection function is used to support lenses exhibiting barrel radial distortion. We transform from r to r' to reduce the radial distortion using equation (3), (4), (5). (v_0, u_0) is the principle point, (f_u, f_v) is the focal length and (ω) is the distortion. All these camera related parameters are assumed to be known.

$$\text{CamProj}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \frac{r'}{r} \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix} \tag{3}$$

$$r = \sqrt{\frac{x^2+y^2}{z^2}} \quad (4)$$

$$r' = \frac{1}{\omega}\arctan\left(2r\tan\frac{\omega}{2}\right) \quad (5)$$

Patch search is used to find a single map point in the current frame. After successful patch observations, a camera pose update can be applied. The pose update is computed iteratively by minimizing a robust objective function of the re-projection error.

-Mapping

The map consists of M points, which are located in the world coordinate frame and N key frames, which are snapshots taken at various points in time with the camera-centered coordinate frame. Each key frame also stores a four-level pyramid of gray scale 8bpp images. Each map point stores a single source key frame (typically the first key frame in which this point was observed), a single source pyramid level within this key frame, and pixel location within this level.

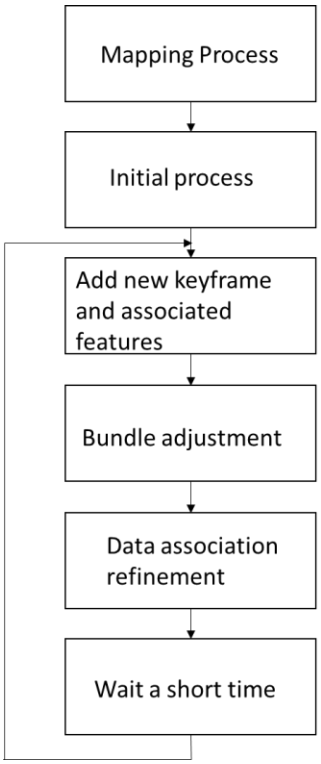


Figure 5.7 Flow Chart for Mapping Process

When the PTAM starts, an initial map is built by using the five-point stereo algorithm. The initializing process needs a user's cooperation. First, put the camera in a location so that the system will generate the first key frame and stores 1,000 2D patch-tracks. Then, the user needs to move this camera to a slightly offset position. After this, the system will generate the second key frame, and the 2D patches will be tracked through this motion.

Using the five-point algorithm can estimate an essential matrix and triangulate the base map. The whole mapping process is shown in Fig. 5.7.

As the camera moves away, the map needs more key frames. The new key frames to be added must satisfy several conditions. The tracking quality must be good; Time must be more than twenty frames from the last key frame is added; and the camera must move at least a certain distance. The key frame first assumes the tracking camera's pose estimates and measures the visible features in the frame. The tracking system calculates the features from the accelerated segment test (FAST) corners for the pyramid of the key frame. Then it uses non-maximal suppression and thresholding to make a set of the most salient points. After discarding the salient points near the observation of existing features, the remaining salient points are about to become map points. New map points need depth information, so they can form triangulation with another view. The nearest key frame is selected as the second view.

Correspondence between the two views are established using epipolar search: the pixel patches around corner points, which lie along the epipolar line in the second view, are compared to the candidate map points using zero-mean sum square difference (SSD).

The map is refined using bundle adjustment. Bundle adjustment iteratively adjusts the map to minimize the robust objective function:

$$\{\{\mu_2..\mu_n\},\{p'_1..p'_M\}\} = \underset{\{\{\mu\},\{p\}\}}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j\epsilon S_i} \mathrm{Obj}\left(\frac{|e_{ji}|}{\sigma_{ji}}, \sigma_T\right) \quad (6)$$

We assume that the jth map point in keyframe i should be in (u_{ji}, v_{ji}) with a standard deviation of σ_{ji} pixels. The state of the map can be E_{KW} and p. The reprojection error is represented by e_{ji}.

Data association refinement is executed when the process is idle. This part is used to

improve the map quality. For example, when a new feature is added by epipolar search, measurements initially exist only in two key frames. However, it is possible that this feature is visible in other key frames as well. If this is the case, then measurements are made, and they are successfully added to the map.

5.3 Monocular UAV control platform

For RFID inventory, the most important capability for the UAV is to navigate in an unknown GPS-denied environment. The challenges of this topic include how to localize the UAV with its sensor and how to navigate in an environment with limited sensor information, even sometimes sensor information loss, due to high network latency. This problem has a popular solution, entitled SLAM, for a robot and a robust control method. For applying SLAM on UAV, choose which type of sensors have the most influence a lot. Nowadays, monocular cameras, laser range scanners, RGB-D sensors, and stereo cameras have all been undergone experimentation. Comparing all these sensors, we think the monocular camera is the most suitable for our UAV. It has several advantages include:

(1) Obtaining rich information from environments with very light weights and small sizes, which cannot be detected by other sensors.

(2) The long range of a monocular camera (as compared with an RGB-D or stereo camera), which can work well in both small, obstacle-rich, and large, open environments. However, it cannot get the scale of the environment directly. It needs to apply some algorithms and other sensors to help with the scale.

According to the requirements discussed above, we chose the Parrot AR Drone 2.0 as our

experiment UAV. It has some advantages. First, it is low in cost. You can get a good drone for less than $300. It has several sensors which are necessary for navigation. It has two cameras, a front camera and a bottom camera, a sonar placed on the bottom, and an IMU. With all these parts, the drone is estimated to weigh approximately 400g. Although it only provides a Wi-Fi communication tunnel, it is enough for controlling and transferring data.

There are two packages in ROS which can help control the drone. One is called the ardrone package and the other is called the tum_ardrone package. The ardrone package is an interface between the UAV and the controller. It can receive different kinds of data sent by UAV via Wi-Fi. These data include information from sensors (both front and bottom cameras and IMU), odometry data, and settings for the drone, such as tag detection and hovering mode.

The other package, tum_ardrone, is a basic controller for the drone. It mainly includes PTAM algorithm and PID control [41]. PTAM can help the monocular camera track itself while mapping. It also helps the monocular camera scale its depth. It uses extended Kalman filter (EKF) to compensate for different time delay in the system.

Meanwhile, PID control helps the drone move smoothly and accurately. When controlling the drone, a command will first be modified according to the PID algorithm before being sent. Then, EKF will calculate the transfer delay with the modified command to get the final command sent to the drone. In the PID control method, we assume Gaussian noise in the sensor measurement.

$$x_i \sim N(\lambda\mu_i, \sigma_x^2 I_{3\times3}) \quad (7)$$

$$y_i \sim N(\mu_i, \sigma_y^2 I_{3\times 3}) \tag{8}$$

Equation (7) and (8) show the Gaussian noise model for x and y. μ_i denote the real distance traveled by the UAV. σ_x and σ_y mean the variances of the measurement errors.

For the state space in EKF, we use ten state variables in all.

$$x_t \coloneqq \left(x_t, y_t, z_t, \dot{x}_t, \dot{y}_t, \dot{z}_t, \Phi_t, \Theta_t, \Psi_t, \dot{\Psi}_t\right)^T \tag{9}$$

Where (x_t, y_t, z_t) denote the position of the UAV; $(\dot{x}_t, \dot{y}_t, \dot{z}_t)$ denote the velocity of world coordinate in m/s; $(\Phi_t, \Theta_t, \Psi_t)$ represents the roll, pitch and yaw; $\dot{\Psi}_t$ means the yaw speed. We define $h(x_t)$ as the observation function for the UAV. $h(x_t)$ is based in camera-centered coordinate. Therefore, (x,y) can be transformed from world coordinates. z, pitch and roll are the same.

$$h_l(x_t) \coloneqq \begin{pmatrix} \dot{x}_t \cos\Psi_t - \dot{y}_t \sin\Psi_t \\ \dot{x}_t \sin\Psi_t + \dot{y}_t \cos\Psi_t \\ \dot{z}_t \\ \Phi_t \\ \Theta_t \\ \dot{\Psi}_t \end{pmatrix} \tag{10}$$

Then, the PID process can predict the command. We can get the acceleration for x and y according to the current state x_t. We can calculate the vertical and yaw acceleration and the speed of roll and pitch according the current state and control command u_t. So, the state from x_t to x_{t+1} can be shown as follow:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ z_{t+1} \\ \dot{x}_{t+1} \\ \dot{y}_{t+1} \\ \dot{z}_{t+1} \\ \Phi_{t+1} \\ \Theta_{t+1} \\ \Psi_{t+1} \\ \dot{\Psi}_{t+1} \end{pmatrix} \leftarrow \begin{pmatrix} x_t \\ y_t \\ z_t \\ \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \\ \Phi_t \\ \Theta_t \\ \Psi_t \\ \dot{\Psi}_t \end{pmatrix} + \delta_t \begin{pmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \\ \ddot{x}(x_t) \\ \ddot{y}(x_t) \\ \ddot{z}(x_t, u_t) \\ \dot{\Phi}(x_t, u_t) \\ \dot{\Theta}(x_t, u_t) \\ \dot{\Psi}_t \\ \ddot{\Psi}(x_t, u_t) \end{pmatrix} \tag{11}$$

5.4 Control Algorithm

5.4.1 High-level Control Algorithm

The drone is planned to be used for inventory and tag localization, so our control algorithm will focus on how to let the drone fly the shortest routine and read the most tags. Inventory often happens on the sales floor or in the warehouse, which are obstacle-rich environments. Because the drone does not work on a premade map and the environment is too large for to fly over during one trip, our control strategy combines the drone and the robot together in order to assist the drone in completing inventory work. We plan to install a platform on top of the robot so that the drone can take off and land on the platform. Additionally, we can apply charging equipment to the drone. The robot has a pre-made map, so it knows its own position and where the shelves and racks with tagged merchandise are. The most important thing is that the robot can take the drone to locations near high shelves and other area which the robot cannot reach on its own.

For the inventory algorithm, we mainly consider the shelves to be inventory objects because of their heights and lengths, which makes the drone necessary. The drone obtains information about the altitude and length of the shelf from the robot. The shelf has many

layers, so the drone cannot read tags from all of them at one time. Therefore, our strategy is to make the drone move one layer at a time. It will first take off at about 1m high. After initialization, it will move from one side of the shelf to another, as shown in Fig. 5.8. During this process, the drone's altitude will stay constant at 1m. After finishing the inventory for this layer, it flies up to 1.5m. From there, it moves from side to side to perform the inventory. It will repeat this process until the drone's altitude reaches the altitude of the shelf. When the drone finishes all the inventory work on the shelf, it will move back to the robot and land on its top platform.

In conclusion, our high-level control algorithm determines that the drone will take off, land, and even charge on the robot's top platform. The robot will carry the drone and move to high shelves. Then, the drone will take off and perform the inventory for a single shelf. After finishing the inventory, the drone will return to the robot and land on the platform again. The robot will begin to move to the next shelf when it knows the drone has already landed on the top. Fig. 5.8 shows the final goal for our design. Beside the drone and robot, the box in the top-right corner is an example of the result of an inventory report and item location.

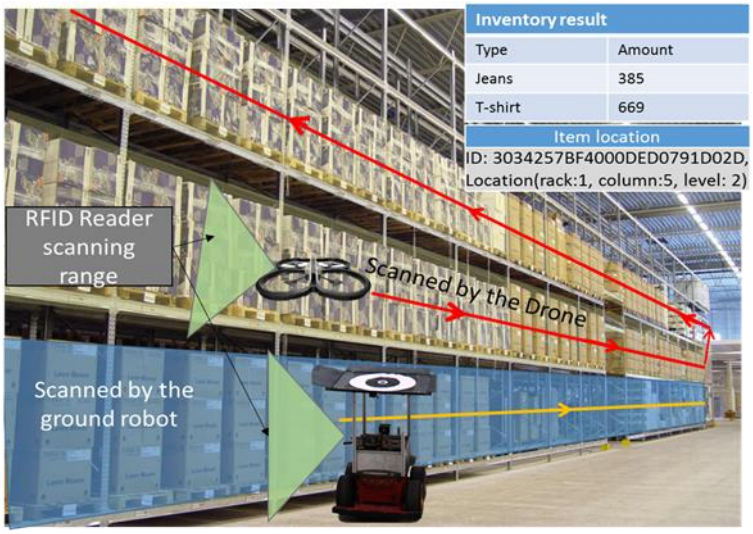


Figure 5.8 A symbiotic autonomous ground robot and drone system operating in a warehouse

5.4.2 Low-level Control Method

Since all the processes are autonomous, we need to completely control the drone by using the control program in order to receive sensor data from the drone. The ardrone package in the ROS provides an interface for that purpose. With this interface, we can control the movement of the drone by sending commands in ROS, such as “takeoff”, or “go to 0 0 1 0”, which means to let the drone fly up 1m in altitude. Because the drone works in 3-D coordinates, its position is usually expressed in the “x y z yaw” form. In this way, the control program is able to affect the drone’s movement as much as we want. The drone will send back data from different sensors. We can obtain data flow, the status of the drone, “move to 0 0 1 0”, etc. with the information provided by both cameras. With this information, we know the status of the drone, the position of the drone according to the PTAM algorithm, and are able to detect an obstacle.

While doing inventory, the drone only relies on the PTAM due to the absence of a pre-made map (which means there is no global map). Therefore, we then create a global

planner. The robot will take the drone to a location near the shelf where the drone will perform inventory. For our current design, we simplified the process by placing the landmark on the ground near a shelf, which served as the drone's dock. The next step involves the drone moving from one side to another, and then continuously rising 0.5 m until it reaches the altitude of the shelf. After all the inventory work is finished, the drone will return to the dock. The entire process is shown in Fig. 5.9.

Figure 5.9 The drone autonomous takes off, navigates and lands back

During the navigation process, a local planner and an obstacle avoidance method are still necessary. We first build a local map and dynamically update a small part (1 m×1 m×1 m) in front of the camera. Capture the key points (with distance to the camera less than 1m) in the map from PTAM and mark them with value -1. According to the field of view in the front camera, shown in Fig. 5.11 & 5.12, detect the nearest obstacle points and mark these points with value 1. Then, the points between the front camera position and points with value 1 are 0.

When the drone moves, only allow it to move in the points of 0, not 1 or -1.

If one goal point is blocked by an obstacle, discard it and fly up or down to another altitude. Next, move to determine whether the destination is blocked. If it is not blocked, move to it from the top or bottom. Abandon the destination if it is blocked, and then move to the next destination point or find a way to detour.

We let PTAM process send out map points and current camera pose in coordinate format. We then select the map points which are in the camera's field of view, according to the model shown in Figs. 5.10 & 5.11. For these selected map points, we classified them into three altitudes: the current camera altitude ± 0.5 m, higher than the current camera altitude + 0.5 m and lower than the current camera altitude – 0.5 m. For each altitude, one must determine the nearest obstacle point (distance in y orientation), and save all of the obstacle points.

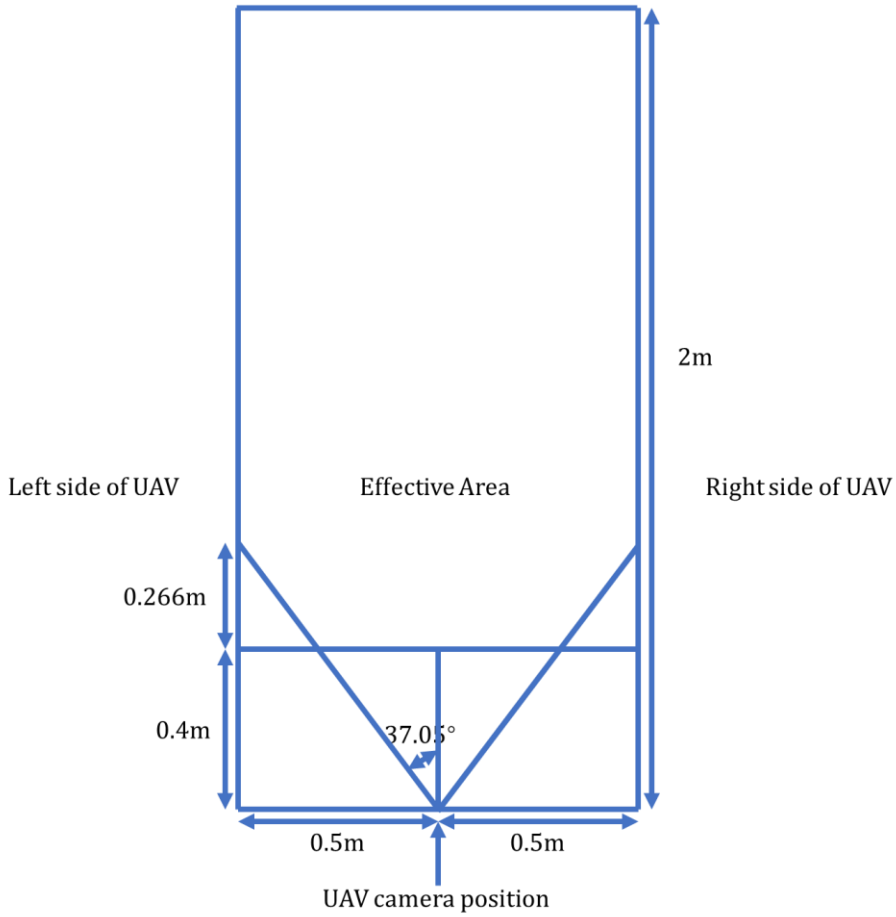


Figure 5.10 Top view of front camera model

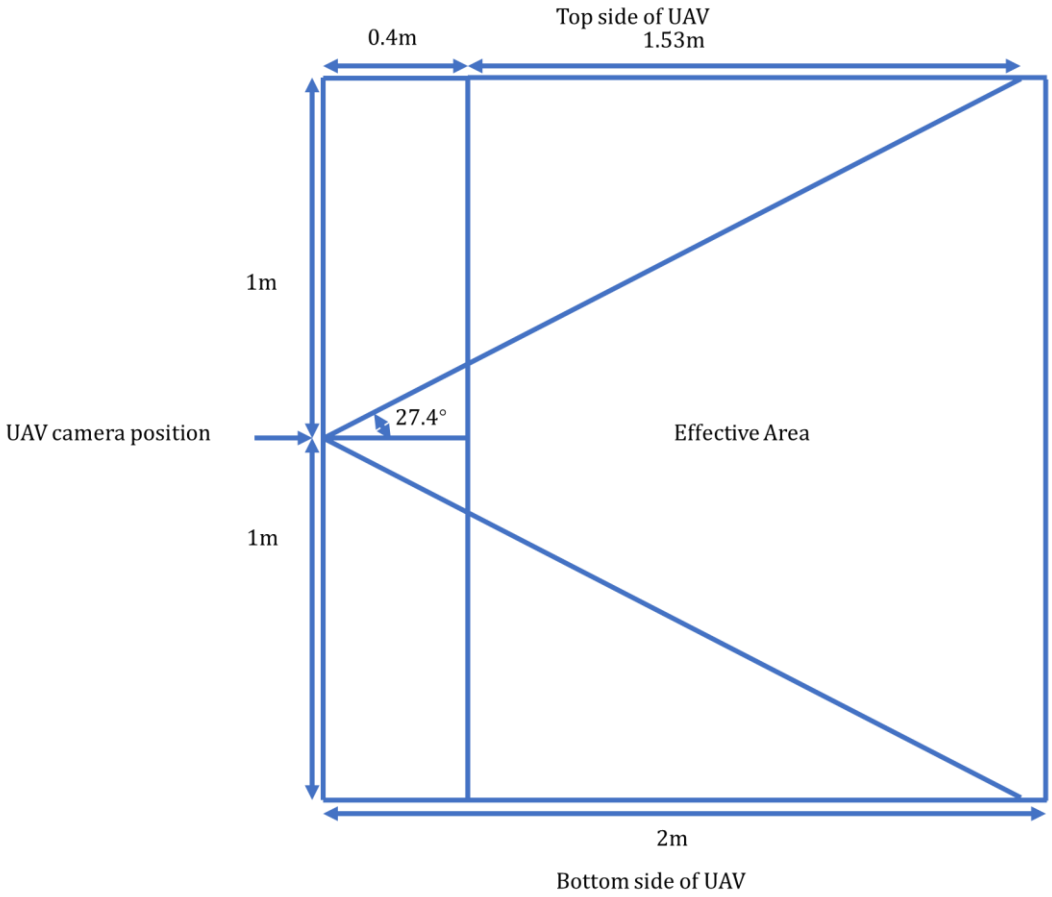


Figure 5.11 Side view of front camera model

When controlling the UAV, set the series of destinations by 1m. For example, if the drone is in “0 0 1 0” and plans to move to “0 0 3 0”, let the drone move to “0 0 1 0”, “0 1 1 0”, “0 2 1 0”, “0 3 1 0” in series.

The last step for the drone is to land on the landmark. The drone can recognize the landmark through its bottom camera. Our algorithm involves returning to the top of the robot after finishing all of the inventory work. The drone can directly use our landing algorithm if it recognizes the landmark. If not, it will search for the landmark. The drone will move as Fig. 5.12 shows. Each step is increased by 0.5 m, so the drone will move in different directions. If it recognizes the landmark, it will stop all of the movements and begin to land.

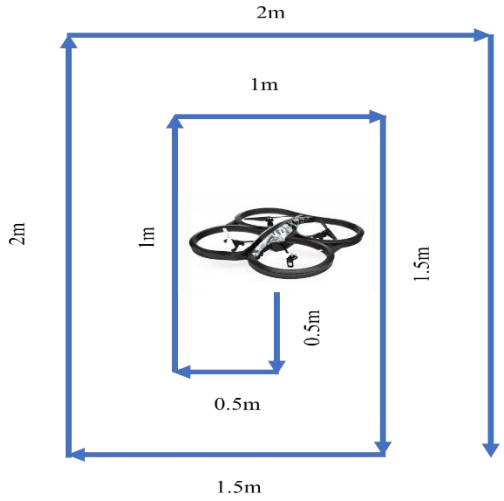


Figure 5.12 Search method for landmark

When the drone recognizes the landmark, it will begin to land according to the landing algorithm. The flow chart is shown as below:

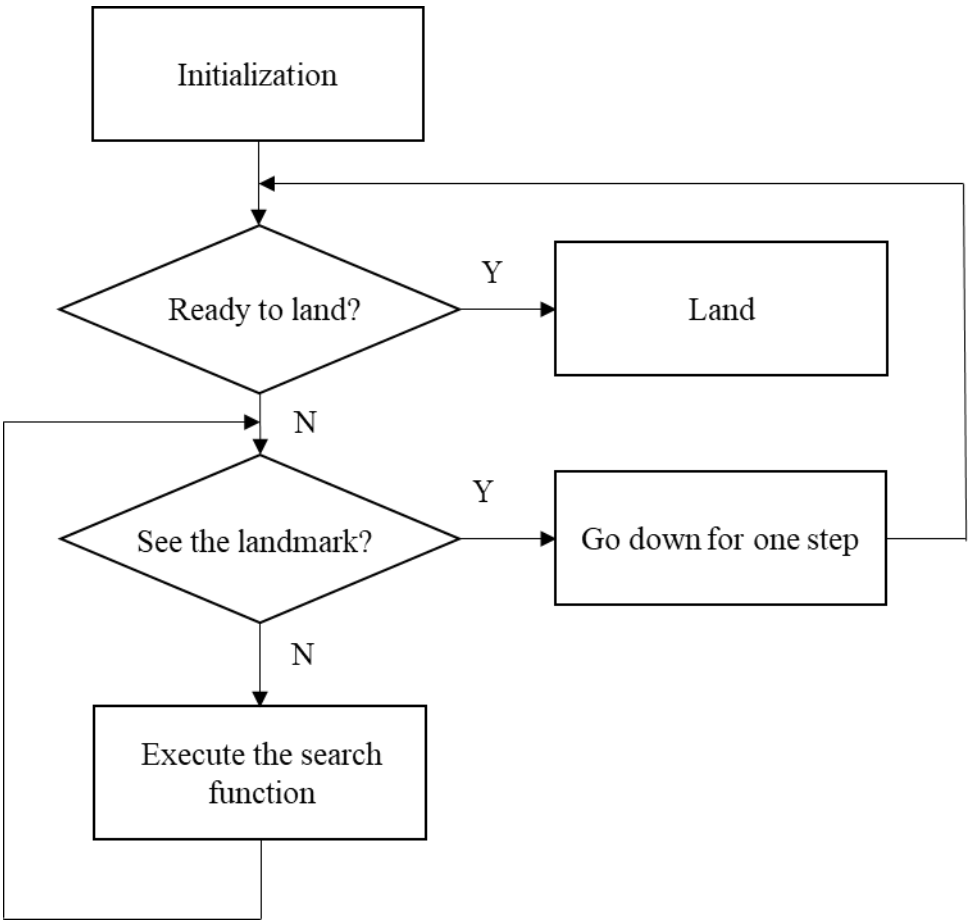


Figure 5.13 Landing algorithm

First initialize the landing process. Get the current height h. The diagonal angle of the bottom camera $\theta = 64°$. So, the radius of the detecting range for the bottom camera is:

$$r = h \times \tan\frac{\theta}{2} \quad (11)$$

After the initialization, we continue to check whether the drone is ready to land. If the current height h is less than 0.5 m, we directly let the drone land. We do this for two reasons. One is because if the drone can detect the landmark in 0.5 m height, the landmark will be in the range of 0.3 m. The radius of the drone itself is 0.3 m. Considering the size of the landmark (0.2 m×0.3 m), the maximum error between the centers of drone and landmark is 0.15m. The second reason is that the bottom camera can hardly recognize the landmark with a height of less than 0.5m.

If the height is still not ready to land, we will check whether the drone can see the landmark. If it can, continue to make the drone go down. It will go down for a pre-set percentage of current height. For example, if we set the percentage as 0.35 and the current height is 2 m, it will go down 0.7 m for this time. After going down, check whether the drone can see the landmark. If it cannot see the landmark this time, execute the search function. Repeat these steps until the height is less than 0.5m and the drone will directly land.

We design a search function to avoid the drone lose the track of the landmark. The function is called when a drone cannot see a landmark after going down and it can see the landmark before it goes down. The detailed process is shown in Fig. 5.14.

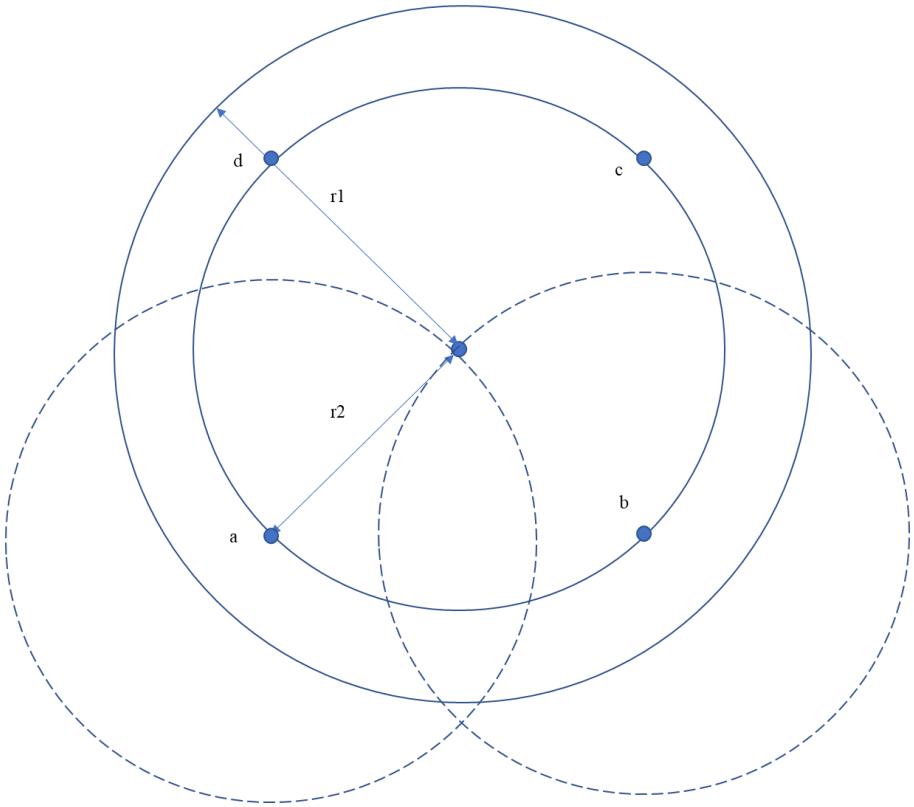


Figure 5.14 Search function

The circle with radius *r1* means the detecting range before the drone going down and the circle with radius *r2* means the detecting range after the drone going down. So, if the drone loses the track of the landmark after going down, the landmark must lay in somewhere between the two circles. So, we make the drone move to *a*, *b*, *c*, *d* in sequence to look for the landmark. Once the drone detects the landmark, it will stop going to other points and begin to go down again.

5.5 Experiment & results

5.5.1 Platform overview

The whole UAV experiment platform is made up of three components: Parrot AR Drone

2.0, Wi-Fi network, and a controlling workstation.

Parrot AR Drone 2.0 consists of a drone shell, a hull, and a battery. The hull only protects the drone from collision. The battery is a lithium-ion polymer 11.1 V battery with a capacity of 1000 mAh. One battery can last for 15 minutes if the drone keeps flying. The drone shell contains four brushless motors, four propellers, and sensors. Fig.5.15 shows the illustration of AR Drone 2.0 and its indoor hull. The brushless motors are 14.5 watts, run 28, 500 revolutions per minute, and have 8/72 gear reductions. Two of the four propellers rotate clockwise and the other two rotate counter-clockwise. The propellers are 18 cm long and about 1 cm wide. In regard to the sensors, the AR Drone has two cameras, including front and bottom cameras, a sonar, and an IMU. The front camera is a 720p sensor with the 93-degree lens. Its frequency can be up to 30 frames per second. For the bottom camera, it's a QVGA sensor with a 64-degree lens. Its maximum frequency is 60 frames per second. The IMU is placed inside the drone shell. The sonar is placed beside the drone's bottom camera. It is mainly used for measuring the distance from the drone to the ground.

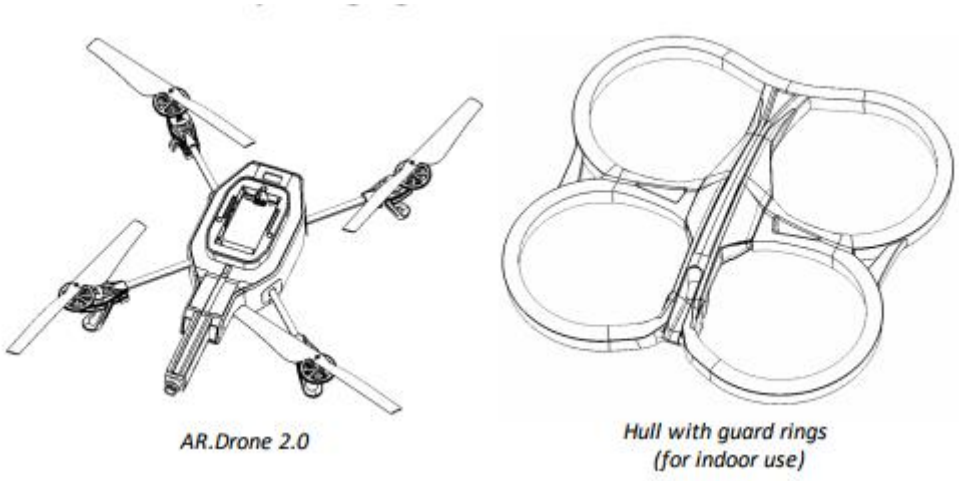


Figure 5.15 Parrot AR Drone 2.0 schematic illustration

The experiment area is shown in Fig. 5.16. It is a wall which mimics the real ware house shelves. The landmark in Fig. 5.16 represents the dock for the drone. The drone will take off from here and land back here. The shelves area is 6.3 m long and 3 m high.

Figure 5.16 Experiment area

5.5.2 Experiment & result

Before showing the experiment results, we will first introduce the simulation result for

the landing algorithm. In the simulation part, we have tested many times with different values for original height, searching angles, going down step length and Gaussian noise parameters. I will first show a simple result grid generated by the simulation program.

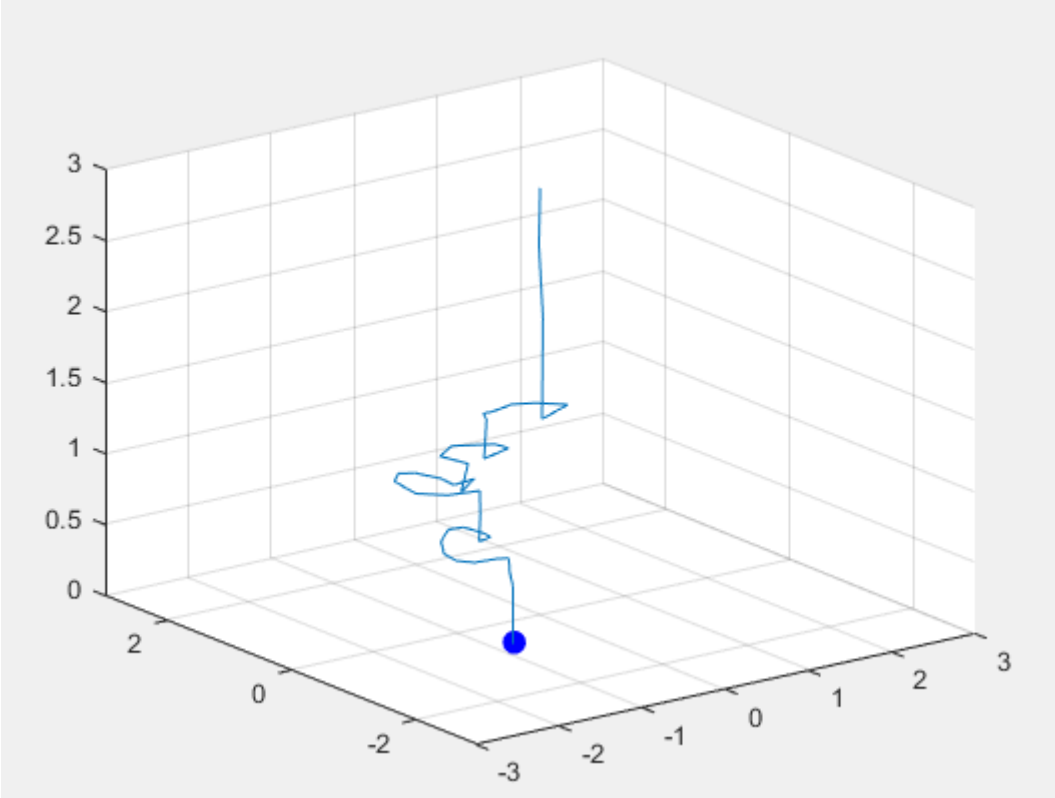


Figure 5.17 simulation result for single run

From Fig. 5.17, we can see a trajectory of the drone until it reaches the landmark which is represented by a blue point on the horizonal plane. We can see the landing strategy as we discussed above. The drone will go down step by step until it loses the sight of the landmark. Then, it will try to search for the landmark and once it finds the landmark, it will begin to go down again. As soon as its height is below 0.5 m, it will directly land. Then, we set the h as 3m which is our real experiment landing height. We set the Gaussian noise for x and y as 0.005 and for z as 0.001 because vertical measurement is more accurate than horizonal

measurement. Then, we use different searching angle from 10° to 50° and different down step length from 0.1 to 0.4 to get different parameter situations. For each situation, we run the simulation for 100 times. Among thses results, we look for the best parameters when $h = 3$. When down step length equals to 0.2 and the searching angle equals to 30°, the drone has the most times to land on the landmark. The success rate varies from 75% to 92%. Fig. 5.19 shows the result of 10 repeating experiment with the best parameter for the simulation with a success of 9/10. When there is no noise, the success rate is 100%. We have applied this algorithm in pratical experiments and it has a 5/5 success rate.

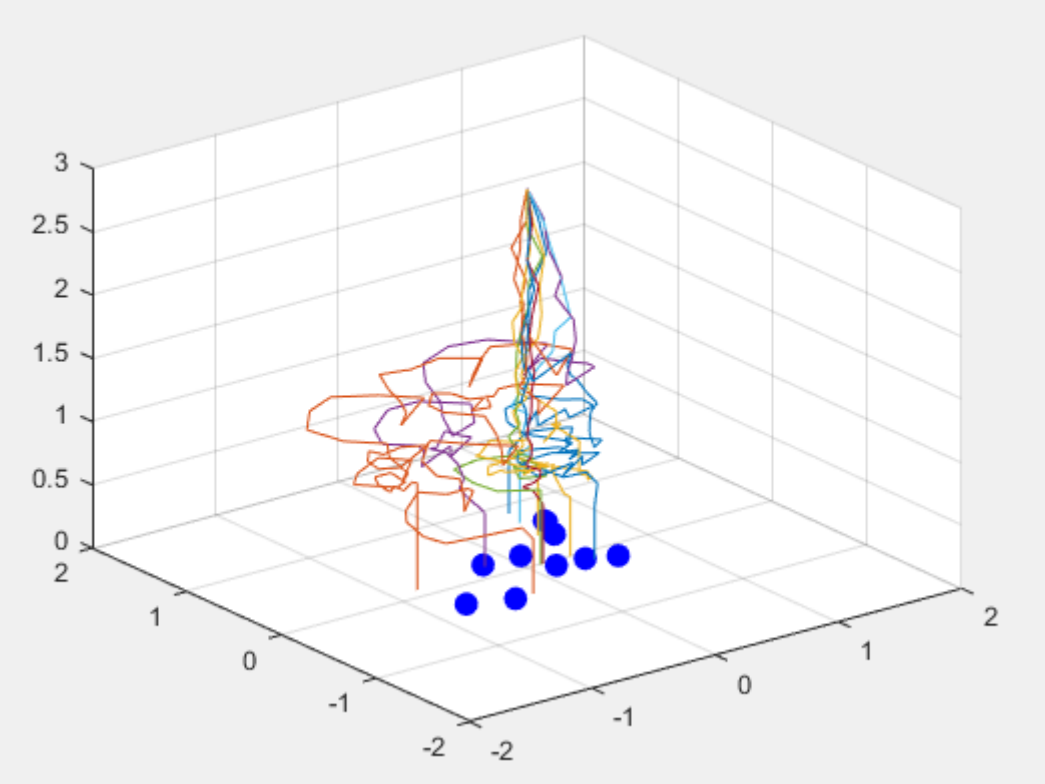


Figure 5.18 simulation results

For the experimentation process, we put the drone in the dock, connect the Wi-Fi of the drone in the workstation, and start the control program in the workstation in order to see the

drone’s performance. We also check the obstacle information during the experimentation process.

Fig. 5.19 shows the initial status of the experiment. The right side of Fig. 5.19 is the real environment captured by the camera. It is clear to see that the drone stays on the dock. The landmark is attached to the ground directly under the drone. The left side of Fig. 5.19 is the map made by the PTAM process. The green and red lines represent the y axis positive and x axis positive, respectively. The vertical white line represents the z axis positive.

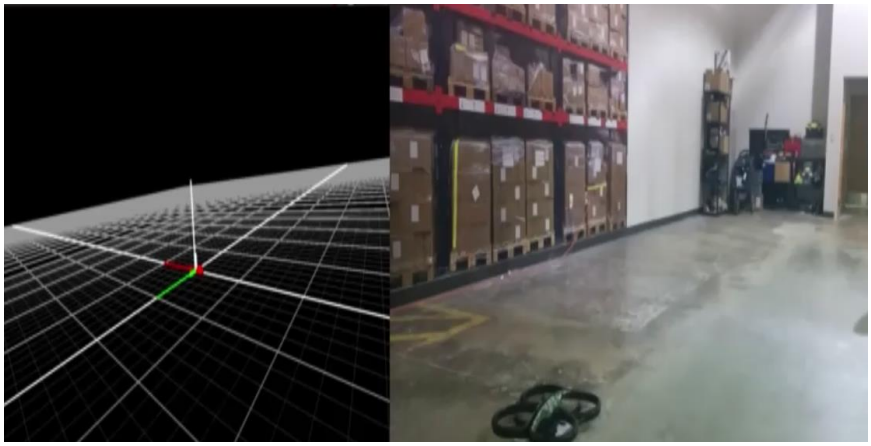


Figure 5.19 Initial status for drone experiment process

Fig. 5.20 displays the drone initialization process. We can see the drone first takes off and flies up and down before it begins to navigate. This process is the initialization process for PTAM. After moving up and down for three times, it will capture the first two key frames and calculate the image depth by applying the five-point stereo algorithm.

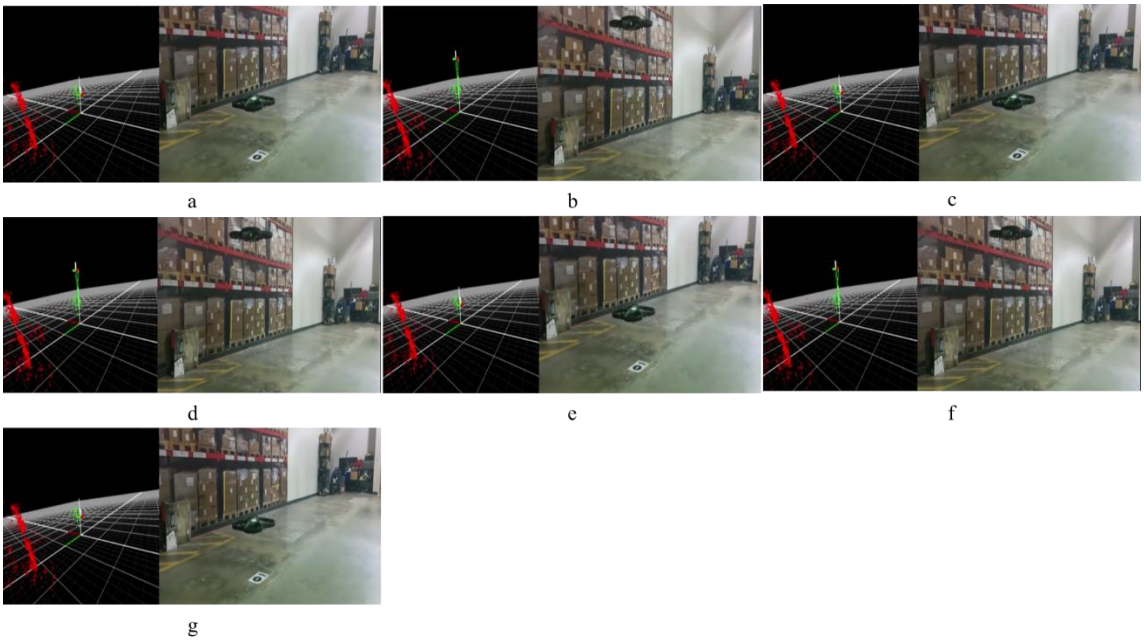


Figure 5.20 Process of drone initialization

When the initialization process is completed, the drone begins to move horizontally to complete the inventory work. The inventory process is shown in Fig. 5.21. Each step is 2 m. After reaching the side of the shelf, it moves 0.5m in the vertical direction. Then, repeat the horizontal move until the drone reaches its maximum height.

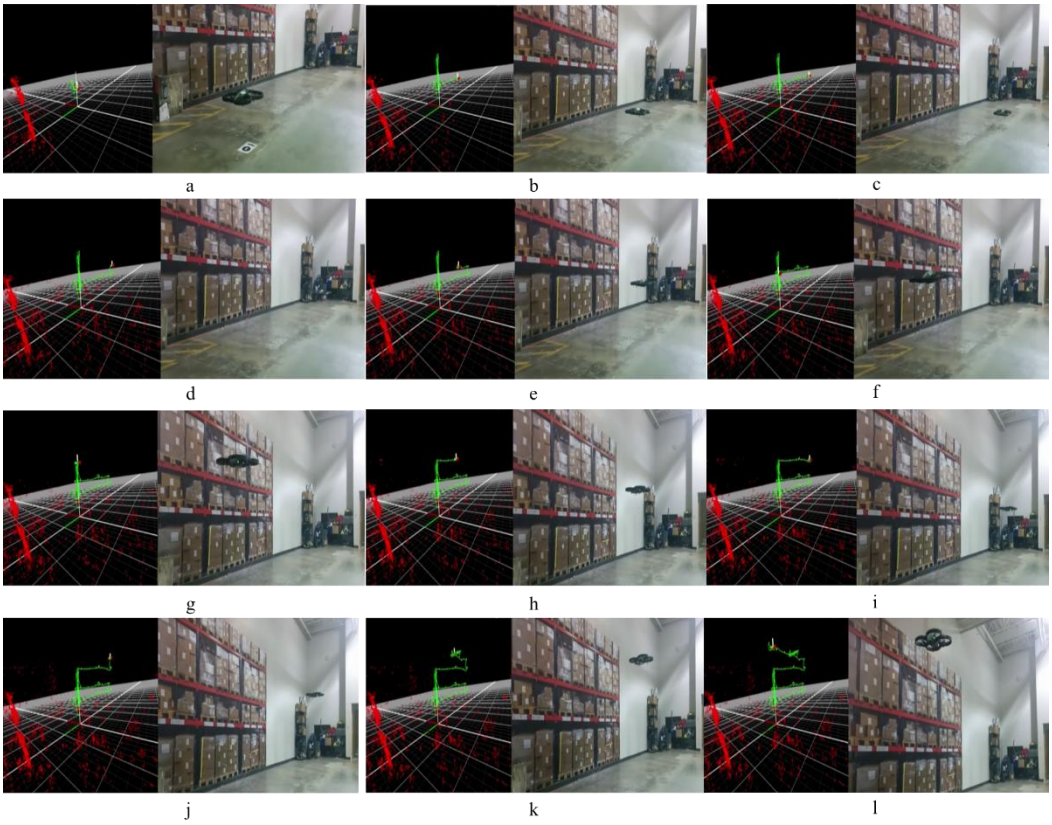


Figure 5.21 Process of drone inventory

After finishing the inventory at its maximum height, the drone begins to return to the landmark. It first moves back to the initial coordinates so that its position has the greatest probability to detect the landmark. The top graph in Fig. 5.22 shows the drone returning to the top of the landmark. Then, the landing algorithm, discussed above, is applied, so the drone begins to move toward the landmark. As long as it detects the tag, it will land on it. We can see the drone has already landed on the landmark in the bottom graph in Fig. 5.22.

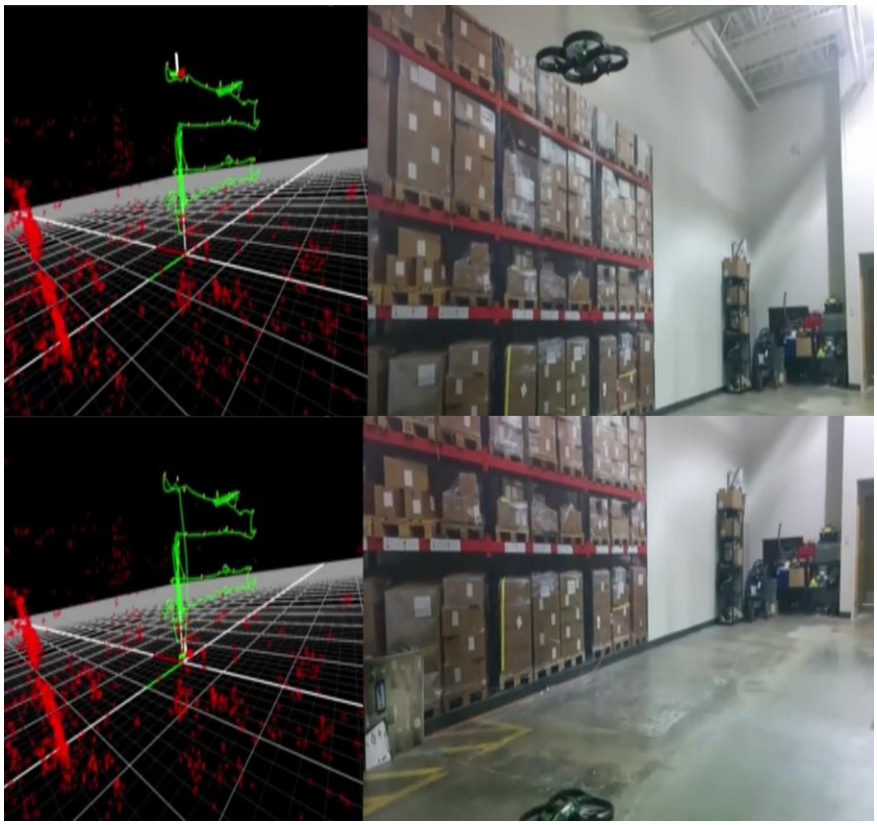


Figure 5.22 Land process

Chapter 6 Conclusion & Future Work

This dissertation first describes previous work using the autonomous robot to do inventory on a sales floor or warehouse. Before the robot can navigate and do the inventory work in an environment, it needs to build a premade map. This map is usually built manually. Furthermore, during the process of inventory, the robot may encounter some inaccessible areas, such as high shelves, which prevent the robot from reading tags. This dissertation presents novel methods to solve these problems. We first apply an algorithm called auto-mapping in the mapping process. This algorithm allows the robot to make a map autonomously in an environment with a setting range. The result shows that the robot can autonomously make a map and navigate with it. In order to make the map more accurate, we developed an algorithm called multi-layer mapping. It modifies the original SLAM process and uses Kinect and LIDAR to make two maps with the same pose. The result shows that the map with new algorithm is more accurate than the map made before. Beyond the mapping improvement, we also present an algorithm to control a drone to perform the inventory work. The drone can take off in a specific spot, navigate, return to the original spot, and land. The whole process is completed autonomously and successfully.

6.1 Conclusion

For the mapping algorithms, we have conducted several experiments in our mock apparel store to evaluate the quality of the maps made by auto-mapping and multi-layer mapping algorithms. The experiment results show that the map made by auto-mapping is even more

accurate than the map made manually. This approach enables the people who do not know how to draw a map manually to use our robots. The experiment results of multi-layer mapping show that the map is more accurate than the map made by an original algorithm, both on sizes of obstacles and positions of obstacles. This map can help the robot navigate more accurately in the environment so that the robot can move to more accurate positions and produce more inventory results.

For the drone controlling algorithm, we have tested the drone performance in a mimic warehouse environment. The result shows that the drone can autonomously execute the action from taking off, navigating according to the preset routine, and landing back at its original position. Although the drone takes off and lands in a specific position on the ground, it means that the drone can do ~~a~~ simple inventory work now. If we can prolong the time for the battery and make the drone take off and land on the robot, then we can definitely deploy our drone doing inventory on a real sales floor or in another warehouse.

Combining the algorithms that I have tested, the autonomous inventory robot now can autonomously make the map needed for navigation, and the quality of the map is better than before. Furthermore, the drone can help the robot do inventory on high shelves. Therefore, we have concluded that the algorithms we have created can greatly help the robot with inventory and tag localization work. It eventually will help to reduce the labor in inventory and meanwhile improve the effect of the inventory work.

6.2 Future work

The main drawback for the auto-mapping algorithm is that if the environment is too big,

the map will be more inaccurate. Our experiment apparel store is 204 square meters. However, a real warehouse or sales floor is much bigger than this. For instance, a Walmart Supercenter could be up to 24, 154 square meters. Apparently, a single map cannot cover the whole areas in this kind of warehouse. Therefore, we may need to do multi-robot mapping in the future. We may also set a specific size as the standard map size so every time the robot only builds a map with such a size. After finishing it, the map will be saved so that it can be used in the future.

For the drone project, there are many things we need to do in the future. First, the drone needs to have an obstacle avoidance function. Although we have already designed an algorithm for the obstacle avoidance function, we still need to do some experiments on it to see if it works. Also, we need to build a platform on the top of the robot and make the drone take off and land on the platform every time. Finally, it needs a more accurate landing algorithm, so we may need to update our version to meet the requirements to land on the top of the robot.

Reference

[1] R. Want, “An introduction to RFID technology,” *IEEE Pervasive Computing*, vol. 5, no. 1. pp. 25–33, 2006.

[2] T. J. Fan, X. Y. Chang, C. H. Gu, J. J. Yi, and S. Deng, “Benefits of RFID technology for reducing inventory shrinkage,” *Int. J. Prod. Econ.*, vol. 147, no. PART C, pp. 659–665, 2014.

[3] B. Hendriks, B. Meerbeek, S. Boess, S. Pauws, and M. Sonneveld, “Robot vacuum cleaner personality and behavior,” *Int. J. Soc. Robot.*, vol. 3, no. 2, pp. 187–195, 2011.

[4] S. Pieskä, M. Luimula, J. Jauhiainen, and V. Spitz, “Social service robots in public and private environments,” *Recent Res. ...*, no. August 2015, pp. 190–195, 2012.

[5] H. Omori, T. Nakamura, and T. Yada, “An underground explorer robot based on peristaltic crawling of earthworms,” *Ind. Robot An Int. J.*, vol. 36, no. 4, pp. 358–364, 2009.

[6] M. Ueno, T. Nimura, H. Ando, K. Maeda, and K. Tamura, “On the descending motion of a deep-sea robot,” *Control Eng. Pract.*, vol. 16, no. 4, pp. 446–456, 2008.

[7] J. J. Zeng, R. Q. Yang, W. J. Zhang, X. H. Weng, and Q. Jun, “Research on semi-automatic bomb fetching for an EOD robot,” *Int. J. Adv. Robot. Syst.*, vol. 4, no. 2, pp. 247–252, 2007.

[8] J. Zhang, Y. Lyu, T. Roppel, J. Patton, and C. P. Senthilkumar, “Mobile robot for retail inventory using RFID,” *Proc. IEEE Int. Conf. Ind. Technol.*, vol. 2016–May, pp. 101–106, 2016.

[9] B. Y. T. I. M. Bailey and H. Durrant-whyte, “Simultaneous Localization and Mapping

(SLAM),” *Update*, vol. 13, no. September, pp. 108–117, 2006.

[10] S. Tabbone and D. Ziou, “Edge Detection Techniques - An Overview,” *Int. J. Pattern Recognit. Image Anal.*, vol. 8, pp. 537–559, 1998.

[11] J. Badger, D. Gooding, K. Ensley, K. Hambuchen, and A. Thackston, “Robot Operating System (ROS),” *Stud. Comput. Intell.*, vol. 625, no. January, pp. 343–373, 2016.

[12] Z. Zhang, “Microsoft kinect sensor and its effect,” *IEEE Multimedia*, vol. 19, no. 2. pp. 4–10, 2012.

[13] R. Burtch, “Lidar principles and applications,” *Current*, pp. 1–13, 2002.

[14] L. R. Newcome, “Commercial UAV Operations in Civil Airspace,” in *Proceedings of SPIE - The International Society for Optical Engineering 4127*, 2009, vol. Volume 412, pp. 31–39.

[15] H. Eisenbeiß, E. T. H. Zurich, H. Eisenbeiß, and E. T. H. Zürich, *UAV photogrammetry*, no. 18515. 2009.

[16] L. Wallace, A. Lucieer, C. Watson, and D. Turner, “Development of a UAV-LiDAR system with application to forest inventory,” *Remote Sens.*, vol. 4, no. 6, pp. 1519–1543, 2012.

[17] F. Bachmann, R. Herbst, R. Gebbers, and V. V Hafner, “Micro UAV based georeferenced orthophoto generation in VIS + NIR for precision agriculture,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci. Vol. XL-1/W2, 2013 UAV-g2013*, vol. XL, no. September, pp. 11–16, 2013.

[18] M. R. Pedersen, L. Nalpantidis, R. S. Andersen, C. Schou, S. Bøgh, V. Krüger, and O.

Madsen, “Robot skills for manufacturing: From concept to industrial deployment,” *Robot. Comput. Integr. Manuf.*, vol. 37, pp. 282–291, 2016.

[19] J.-H. Han, M.-H. Jo, V. Jones, and J.-H. Jo, “Comparative Study on the Educational Use of Home Robots for Children,” *J. Inf. Process. Syst.*, vol. 4, no. 4, pp. 159–168, 2008.

[20] M. N. Kiyani and M. U. M. Khan, “A prototype of search and rescue robot,” *2016 2nd Int. Conf. Robot. Artif. Intell.*, pp. 208–213, 2016.

[21] H. Robinson, B. MacDonald, and E. Broadbent, “The Role of Healthcare Robots for Older People at Home: A Review,” *Int. J. Soc. Robot.*, vol. 6, no. 4, pp. 575–591, 2014.

[22] M. Morenza-Cinos, V. Casamayor-Pujol, J. Soler-Busquets, J. L. Sanz, R. Guzmán, and R. Pous, “Development of an RFID inventory robot (AdvanRobot),” in *Studies in Computational Intelligence*, vol. 707, 2017, pp. 387–417.

[23] P. Liu, A. Y. Chen, Y. N. Huang, J. Y. Han, J. S. Lai, S. C. Kang, T. H. Wu, M. C. Wen, and M. H. Tsai, “A review of rotorcraft unmanned aerial vehicle (UAV) developments and applications in civil engineering,” *Smart Struct. Syst.*, vol. 13, no. 6, pp. 1065–1094, 2014.

[24] M. Quaritsch, K. Kruggl, D. Wischounig-Strucl, S. Bhattacharya, M. Shah, and B. Rinner, “Networked UAVs as aerial sensor network for disaster management applications,” *Elektrotechnik und Informationstechnik*, vol. 127, no. 3, pp. 56–63, 2010.

[25] S. M. Adams and C. J. Friedland, “A Survey of Unmanned Aerial Vehicle (UAV) Usage for Imagery Collection in Disaster Research and Management,” *9th Int. Work.*

Remote Sens. Disaster Response, no. January 2011, p. 8, 2011.

[26] I. Maza, F. Caballero, J. Capitán, J. R. Martínez-De-Dios, and A. Ollero, "Experimental results in multi-UAV coordination for disaster management and civil security applications," *J. Intell. Robot. Syst. Theory Appl.*, vol. 61, no. 1–4, pp. 563–585, 2011.

[27] C. Zhang and J. M. Kovacs, "The application of small unmanned aerial systems for precision agriculture: A review," *Precis. Agric.*, vol. 13, no. 6, pp. 693–712, 2012.

[28] W. Ni, J. Liu, Z. Zhang, G. Sun, and A. Yang, "Evaluation of UAV-based forest inventory system compared with LiDAR data," *Int. Geosci. Remote Sens. Symp.*, vol. 2015–Novem, pp. 3874–3877, 2015.

[29] S. M. Bae, K. H. Han, C. N. Cha, and H. Y. Lee, "Development of Inventory Checking System Based on UAV and RFID in Open Storage Yard," *ICISS 2016 - 2016 Int. Conf. Inf. Sci. Secur.*, pp. 1–2, 2017.

[30] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 3, pp. 108–117, 2006.

[31] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[32] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D Mapping : Using Depth Cameras for Dense 3D Modeling of Indoor Environments," *RGBD Adv. Reason. with Depth Cameras Work. conjunction with RSS*, vol. 1, no. c, pp. 9–10, 2010.

[33] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-D mapping with an rgb-d camera," *Robot. IEEE Trans.*, vol. 30, no. 1, pp. 177–187, 2014.

[34] F. Endres, J. Hess, N. Engelhard, and … J. S., “An evaluation of the RGB-D SLAM system,” *Icra*, vol. 3, no. c, pp. 1691–1696, 2012.

[35] A. Al Arabi, P. Sarkar, F. Ahmed, W. R. Rafie, M. Hannan, and M. A. Amin, “2D mapping and vertex finding method for path planning in autonomous obstacle avoidance robotic system,” *2017 2nd Int. Conf. Control Robot. Eng. ICCRE 2017*, pp. 39–42, 2017.

[36] Z. Gong, J. Li, and W. Li, “A low cost indoor mapping robot based on TinySLAM algorithm,” *Int. Geosci. Remote Sens. Symp.*, vol. 2016–Novem, pp. 4549–4552, 2016.

[37] A. Basu, “Autonomous Navigation and 2d Mapping Using SONAR,” 2016.

[38] G. Huo, R. Li, L. Zhao, and K. Wang, “A novel fusion method for robot indoor environment mapping,” *2014 IEEE Int. Conf. Robot. Biomimetics, IEEE ROBIO 2014*, pp. 2507–2510, 2014.

[39] R. C. Luo and C. C. Lai, “Multisensor fusion-based concurrent environment mapping and moving object detection for intelligent service robotics,” *IEEE Trans. Ind. Electron.*, vol. 61, no. 8, pp. 4043–4051, 2014.

[40] G. Klein and D. Murray, “IMG Parallel Tracking and Mapping for Small AR Workspaces (PTAM),” *Mix. Augment. Reality, 2007. ISMAR 2007. 6th IEEE ACM Int. Symp.*, pp. 225–234, 2007.

[41] J. Engel, J. Sturm, and D. Cremers, “Camera-based navigation of a low-cost quadrocopter,” *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, pp. 2815–2821, 2012.

[42] X. Wang, S. Mao, S. Pandey, and P. Agrawal, “CA2T: Cooperative Antenna Arrays Technique for pinpoint indoor localization,” in *Procedia Computer Science*, 2014, vol.

34, pp. 392–399.

[43] Xuyu Wang, Lingjun Gao, Shiwen Mao, and S. Pandey, “DeepFi: Deep learning for indoor fingerprinting using channel state information,” *2015 IEEE Wirel. Commun. Netw. Conf.*, no. October, pp. 1666–1671, 2015.

[44] X. Wang, L. Gao, S. Mao, and S. Pandey, “CSI-Based Fingerprinting for Indoor Localization: A Deep Learning Approach,” in *IEEE Transactions on Vehicular Technology*, 2017, vol. 66, no. 1, pp. 763–776.

[45] X. Wang, L. Gao, and S. Mao, “Phasefi: phase fingerprinting for indoor localization with a deep learning approach,” in *2015 IEEE Global Communications Conference, GLOBECOM 2015*, 2015.

[46] I. Sa and P. Corke, “System identification, estimation and control for a cost effective open-source quadcopter,” *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 2202–2209, 2012.

[47] F. Wang, J. Cui, S. K. Phang, B. M. Chen, and T. H. Lee, “A mono-camera and scanning laser range finder based UAV indoor navigation system,” *2013 Int. Conf. Unmanned Aircr. Syst. ICUAS 2013 - Conf. Proc.*, pp. 694–701, 2013.

[48] C. Teuliere, E. Marchand, and L. Eck, “3-D model-based tracking for UAV indoor localization,” *IEEE Trans. Cybern.*, vol. 45, no. 5, pp. 869–879, 2015.

[49] J. Tiemann, F. Schweikowski, and C. Wietfeld, “Design of an UWB indoor-positioning system for UAV navigation in GNSS-denied environments,” *2015 Int. Conf. Indoor Position. Indoor Navig. IPIN 2015*, 2015.

[50] K. Bipin, V. Duggal, and K. Madhava Krishna, “Autonomous navigation of generic monocular quadcopter in natural environment,” *Proc. - IEEE Int. Conf. Robot. Autom.*,

vol. 2015–June, no. June, pp. 1063–1070, 2015.

[51] B. S. Çiftler, A. Tuncer, and I. Güvenç, “Indoor UAV Navigation to a Rayleigh Fading Source Using Q-Learning,” pp. 1–3, 2017.

[52] G. Ajay Kumar, A. K. Patil, R. Patil, S. S. Park, and Y. H. Chai, “A LiDAR and IMU integrated indoor navigation system for UAVs and its application in real-time pipeline classification,” *Sensors (Switzerland)*, vol. 17, no. 6, 2017.

[53] J. Q. Cui, S. Lai, X. Dong, P. Liu, B. M. Chen, and T. H. Lee, “Autonomous navigation of UAV in forest,” *2014 Int. Conf. Unmanned Aircr. Syst. ICUAS 2014 - Conf. Proc.*, pp. 726–733, 2014.

[54] L. V. Santana, A. S. Brandao, and M. Sarcinelli-Filho, “Outdoor waypoint navigation with the AR.Drone quadrotor,” *2015 Int. Conf. Unmanned Aircr. Syst. ICUAS 2015*, pp. 303–311, 2015.

[55] K. Schmid, T. Tomic, F. Ruess, H. Hirschmuller, and M. Suppa, “Stereo vision based indoor/outdoor navigation for flying robots,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 3955–3962, 2013.

[56] M. Santos, C. Rosales, and R. Carelli, “A Novel Null-Space Based UAV Trajectory-tracking Controller with Collision Avoidance,” vol. 16, no. 8, pp. 1–11, 2017.

[57] R. Cronin, “RFID versus Barcode.,” *Pharm. Technol.*, vol. 32, no. 11, pp. 177–178, 2008.

[58] V. M. Thomas, “A universal code for environmental management of products,” *Resour. Conserv. Recycl.*, vol. 53, no. 7, pp. 400–408, 2009.

[59] S. Sarma, D. Brock, and D. Engels, “Radio frequency identification and the electronic product code,” *IEEE Micro*, vol. 21, no. 6, pp. 50–54, 2001.

[60] Dobkin, *The RF in RFID: Passive UHF RFID in Practice*. 2007.

[61] Daniel Dobkin M., *The rf in RFID: uhf RFID in practice*. Newnes, 2012.

[62] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, “Consistency of the EKF-SLAM algorithm,” in *IEEE International Conference on Intelligent Robots and Systems*, 2006, pp. 3562–3568.

[63] M. Calonder, “EKF SLAM vs. FastSLAM,” *Cvlab-Report-2010-001*, pp. 1–5, 2006.

[64] G. Grisetti, C. Stachniss, and W. Burgard, “Improved techniques for grid mapping with Rao-Blackwellized particle filters,” *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, 2007.

[65] M. Basu, “Gaussian-based edge-detection methods-a survey,” *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 32, no. 3. pp. 252–260, 2002.

[66] M. Keidar and G. A. Kaminka, “Efficient frontier detection for robot exploration,” *Int. J. Rob. Res.*, vol. 33, no. 2, pp. 215–236, 2014.

[67] E. Marder-Eppstein and M. Ferguson, “navigation - ROS Wiki,” *ROS.org*, 2016. [Online]. Available: http://wiki.ros.org/navigation?distro=kinetic.

[68] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, 1997.

[69] T. Luukkonen, “Modelling and Ccontrol of Quadcopter,” *J. Am. Soc. Mass Spectrom.*, vol. 22, no. 7, pp. 1134–45, 2011.

[70] A. Chovancová, T. Fico, E. Chovanec, and P. Hubinský, “Mathematical modelling and parameter identification of quadrotor (a survey),” *Procedia Eng.*, vol. 96, no. October, pp. 172–181, 2014.