UNIVERSITY OF TURKU

Design and Development of an FPGA-based Hardware Accelerator for Corner Feature Extraction and Genetic Algorithm-based SLAM System

Smart Systems

Master's Degree Programme in Information and Communication Technology

Department of Computing, Faculty of Technology

Master of Science in Technology Thesis

2021

Yuhong Fu

Supervisors:

MSc. Jorge Peña Queralta

Assoc. Prof. Tomi Westerlund

June 2021

Master of Science in Technology Thesis
Department of Computing, Faculty of Technology
University of Turku

Subject: Smart Systems
Programme: Master's Degree Programme in Information and Communication Technology
Author: Yuhong Fu
Title: Design and Development of an FPGA-based Hardware Accelerator for Corner Feature Extraction and Genetic Algorithm-based SLAM System
Number of pages: 69 pages, 0 appendix pages
Date: June 2021

Simultaneous Localization and Mapping (SLAM) systems are crucial parts of mobile robots. These systems require a large number of computing units, have significant real-time requirements and are also a vital factor which can determine the stability, operability and power consumption of robots.

This thesis aims to improve the calculation speed of a lidar-based SLAM system in domestic scenes, reduce the power consumption of the SLAM algorithm, and reduce the overall cost of the whole platform. Lightweight, low-power and parallel optimization of SLAM algorithms are researched. In the thesis, two SLAM systems are designed and developed with a focus on energy-efficient and fast hardware-level design: a geometric method based on corner extraction and a genetic algorithm-based approach. Finally, an FPGA-based hardware accelerated SLAM is implemented and realized, and compared to a software-based system.

As for the front-end SLAM system, a method of using a Corner Feature Extraction (CFE) algorithm on FPGA platforms is first proposed to improve the speed of the feature extraction. Considering building a back-end SLAM system with low power consumption, a SLAM system based on genetic algorithm combined with algorithms such as Extended Kalman Filter (EKF) and FastSLAM to reduce the amount of calculation in the SLAM system is also proposed. Finally, the thesis also proposes and implements an adaptive feature map which can replace a grid point map to reduce the amount of calculation and utilization of hardware resources.

In this thesis, the lidar SLAM system with front-end and back-end parts mentioned above is implemented on the Xilinx PYNQ Z2 Platform. The implementation is operated on a mobile robot prototype and evaluated in real scenes. Compared with the implementation on the Raspberry Pi 3B+, the implementation in this thesis can save 86.25% of power consumption. The lidar SLAM system only takes 20 ms for location calculation in each scan which is 5.31 times faster compared with the software implementation with EKF.

Keywords: FPGA, CFE, Genetic Algorithm, SLAM.

Table of contents

List of Abbreviation

SLAM	Simultaneously Localization and Mapping
CFE	Corner Feature Extraction
EKF	Extended Kalman Filter
UAV	Unmanned Aerial Vehicle
DVS	Dynamic Vision Sensor
PnP	Perspective-n-Point
ToF	Time-of-Flight
GNSS	Global Navigation Satellite System
BDS	China’s BeiDou Navigation System
GPS	United States’ Global Position System
GLONASS	Russia's Global Navigation Satellite System
UWB	Ultra-Wide Band
DMA	Direct Memory Access
SONAR	Sound Navigation and Ranging
KF	Kalman Filter
PF	Particle Filter
IMU	inertial measurement unit
VIO	Visual-inertial odometry
DS	Dempster-Shafer
NDT	Normal Distribution Transform
BA	Bundle Adjustment
CNN	Convolution Neural Network
SMG	scan-matching genetic
GA	Genetic algorithm
ROS	robot operating system
DWA	Dynamic Window Approach
UKF	unscented Kalman Filter
RBPF	Rao-Blackwellized Particle Filter
PRNG	Pseudo random number generator
CORDIC	coordinate rotation digital computer
SCD	Singular Value Decomposition

List of Figures

List of Tables

1 Introduction

1.1 Background

In recent years, aging population has become a problem which cannot be ignored in China. The "one-child" policy has been implemented to control the size of the rapidly growing population of China. This policy led to a large fall in the total fertility rate from an estimated 5.9 births per woman in 1970, to 2.9 births per woman by 1979. Although the "full two-child" policy has been implemented, the assistants who have efficiency in provision for the aged would be in shortage comparing with the increasing aging population, which is a severe problem in the next few decades. The service robots aimed at indoor scenarios can help to release the burden of elders. They can do several works like picking up different items, sweeping the floors, or cleaning the dishes.

Therefore, the State Council of China has issued Next Generation Artificial Intelligence Development Plan. Service Robots' broad applications in education, medical care, provision for the aged, environmental protection, urban operation, which will dramatically improve targeted public service and people's livelihood.[1]

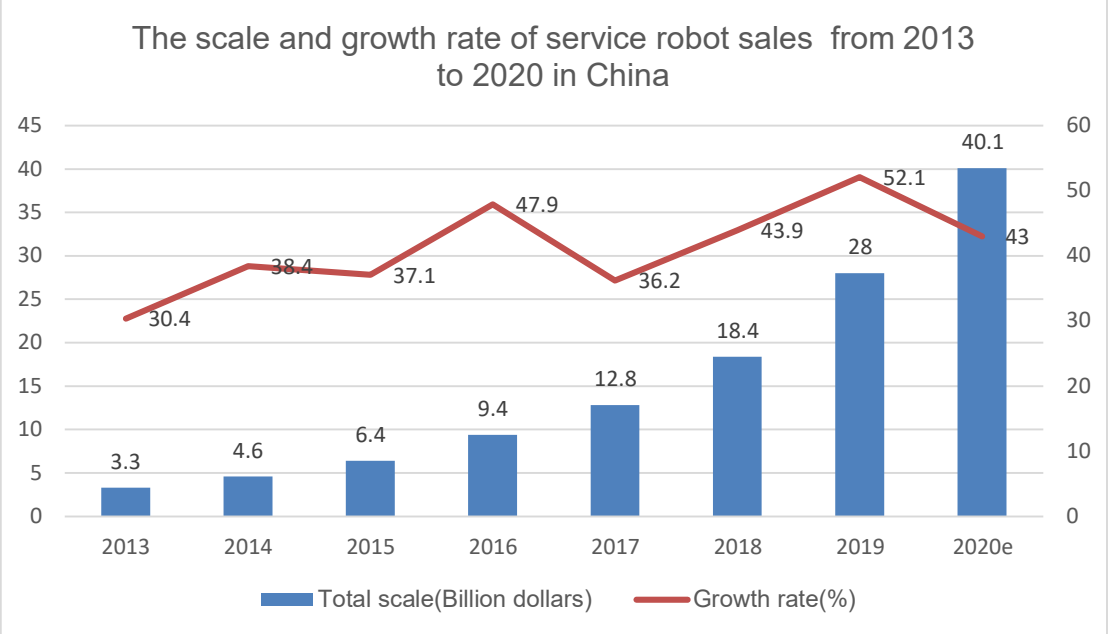

Figure 1.1 Service robot sales and growth rate from 2013 to 2020[2]

With the application of robots and agents in social life, the requirements of the robots in different specific areas will continue to increase. People need different kinds of robots to accomplish different tasks. A robot for sale needs to have a reasonable price. It should also have low power consumption to realize long-term operation tasks with

stable status.

Simultaneous Localization and Mapping (SLAM) is a high-power consumption system on robots which is broadly used in automatic-driving, service robots and Unmanned Aerial Vehicle (UAV). Any agents or robots which have the ability to move must have SLAM system to some extent to control their movement[3][4]. The SLAM system therefore determines the robustness of the whole robot.

When a robot moves from one place to another, it can collect data from the robot's sensors and use these data to localize itself and build an incremental map. It may also recognize some patterns in the environment or extract landmarks from it, which is necessary for the robot to accomplish the navigation task. The different parts in the SLAM system will influence each other, and accumulate errors in the localization and Map. It is difficult to design a robust SLAM system with low power consumption. For these reasons, SLAM is also an important research topic and the necessary components to build a robot, which also needs to be upgraded to adapt to new requirements and complex environments.

The existing SLAM system aiming at indoor scenarios such as house or laboratory has some disadvantages like having high power consumption and need lots of computation resources. For instance, most sweeping robots for indoor usage use the Cortex A7 as their computation core such as Ecovacs T5 Max and Xiaomi MiJia Sweeping Integrated Mopping Machine. Cortex A7 is an efficient microprocessor with low power consumption. But it is hard for Cortex A7 to handle a huge amount of calculation in parallel which is needed in most SLAM systems, it needs a lot of time to map an environment and localization. We therefore need a new SLAM system design with low power consumption and robust enough to deal with different accidents.

1.2 Related Work

1.2.1 SLAM system based on different sensors

SLAM is based on the data provided by distinct types of sensors. So various researches on different sensors have also been made in the world[5]. These researches can now roughly be divided into two types, one is the lidar-based SLAM, another is the vision-based SLAM.[6][7][8][9]

- Vision-based SLAM

A popular type of sensor used for SLAM is the camera. Based on the different

cameras which can be used, the vision-based can be divided into Monocular camera-based SLAM, Stereo camera-based SLAM, RGBD camera-based SLAM, and Dynamic Vision Sensor (DVS) camera-based SLAM.

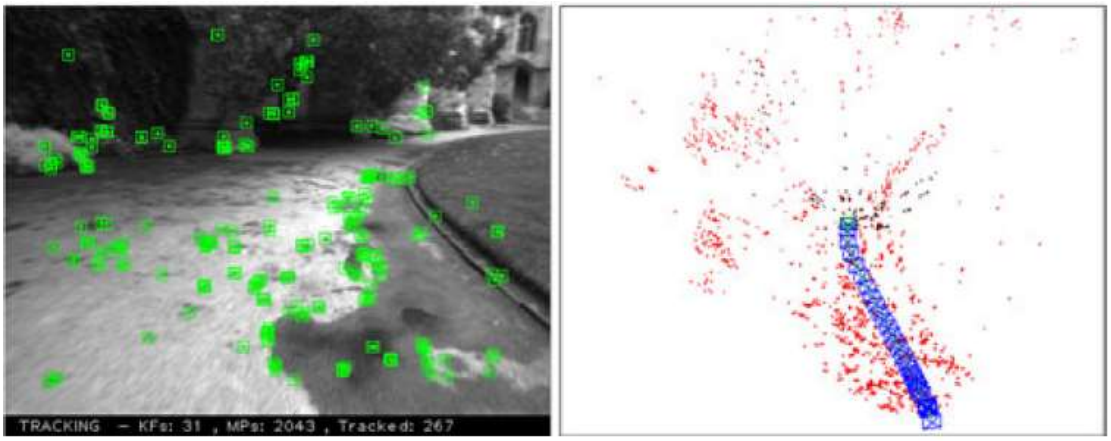

Figure 1.2 A typical vision-based method called ORB-SLAM[10]

Monocular camera-based SLAM only uses a single camera to collect the data from the environment. The images it collects are the projection of the 3D scenes. It is easy to accomplish hardware design, but will take a large number of computation resources to process in real-time. The camera needed to be moved in different places to see the same landmark or feature point, then epipolar geometry[11] can be used to calculate the ground truth of the movement. Figure 1.2 shows a typical monocular camera-based SLAM method called ORB-SLAM. The left part of Figure 1.2 contains the feature points extracted from the image, the right part contains the trajectory and landmarks of ORB-SLAM. For stereo camera-based SLAM, the relative distance between two cameras has already been known before the calculation begins, so it is convenient to use Triangulation[12] to estimate the actual location of the feature point. Through these feature points, the Perspective-n-Point (PnP)[13] can be used to estimate the pose of the robots. Compared with the Monocular camera-based SLAM, Stereo camera-based SLAM doesn't take so many computation resources.

Another method that has been extensively used is the RGBD camera-based SLAM. The RGBD camera can not only detect the image from the environment but also collect the distance information by calculating the Time-of-Flight (ToF) of the lidar beam. When distances between the feature points and the camera are known, the rebuild process of the 3D scene by using the method called Kinect Fusion[14] can be done. The RGBD camera-based SLAM only needs a small amount of the computation resources, but the cost of the RGBD camera is too high. The DVS camera-based SLAM is a new area for vision-based, it has some advantages such as low power consumption, high

speed. But the data it transmits only contains the pixel illuminate change. For this reason, if information is needed for the computation, the DVS can't be set to observe the static environment, the camera should be in motion or be set to see dynamic scenes. Now some papers have been published which talk about 3D Rebuilding by using the DVS[15]. The DVS can automatically get rid of the static objects in the dynamic scenes, but if the DVS has been set in an environment that has a periodic light source, it is also easy to use DVS to reconstruct 3D scenes.

- Lidar-based SLAM

The above methods can achieve good results in practical applications. But visual SLAM also has an inevitable shortcoming. The image provided by the camera will be affected by environmental factors such as ambient light brightness. It is difficult for the vision-based algorithm to extract useful information from the image provided by the camera in a dark environment or environment which have high brightness. The environmental data provided by lidar is less affected by environmental factors. Therefore, even if the cost of multi-line lidar is relatively high, SLAM research based on lidar will be more popular and more accurate.

According to the amount of laser scanning in one round, lidar can be divided into single-line lidar and multi-line lidar. The single-line lidar can mainly be used to avoid the collision with the high scan speed and it is also robust to use. It also has good performance to detect the surrounding obstacles and know the accurate distances to them. It is also easy for a robot to compute its messages to make fast reactions and be sensitive at any angle, but it can only scan a 2D environment without sending any height information. Now the 2D lidar is mainly used in the service robots like sweeping robots; The multi-line lidar costs much more than the single-line lidar, it is mainly used in automotive. Compared with the single-line lidar, it can recognize 3D scenes with huge information about the environment. It can also identify the height information of objects. Now there are different types of multi-line lidars with 4 beams, 8 beams and so on. Because of the high cost of multi-line lidars. A team at the University of Turku design an FPGA-based 3D lidar built with multiple inexpensive RPLidar A1 2D lidars. This inexpensive design opens a wider range of possibilities for lower-end and smaller autonomous robots, which can be able to produce three-dimensional world representations.[16]

However, no matter single-line lidar or multi-line lidar is used in the SLAM system, limited by the short wavelength of lidar, it is difficult to identify small-scale objects, so

it is only suitable for application scenes in outdoor environments or environment with big obstacles, such as automatic driving.

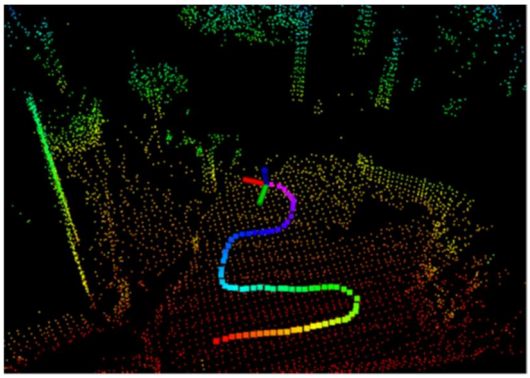

Figure 1.3 LeGO-LOAM based lidar SLAM system[17]

- Other sensors

Besides the sensors mentioned above, there are some other sensors which can be used in the SLAM system.[18]

Global Navigation Satellite System (GNSS) can use position information from China's BeiDou Navigation System (BDS), United States' Global Position System (GPS), Russia's Global Navigation Satellite System (GLONASS), or European Union's Galileo. Because the GNSS module can realize the task on outdoor positioning with low power consumption, it can be installed on mobile devices. The disadvantage of the GNSS is that it has low accuracy for non-military usage, and the satellites' signals can't be received indoors.

In order to overcome the disadvantages of GNSS positioning, the Ultra-Wide Band (UWB) can be used for positioning. A UWB positioning system has two components, one is transmitter and another is a receiver. The whole system can use echo of radar to localize and make a map. UWB labels can be used as landmarks to assist the SLAM system.[19]

Acoustic sensors have also been broadly used in solving the SLAM problem. These sensors are Sound Navigation and Ranging (SONAR) sensors. They locate objects from the echo of a signal that is bounced off the object. Because the light can refract underwater, so it is suitable for using SONAR in these scenes. The SONAR also has its disadvantages, its accuracy and speed are restricted by noise and speed of sound.

Besides, there are some sensors which can't work independently in the SLAM system but necessary for any robots to increase their accuracy and be more robust like the collision sensor and odometer. Both of them can operate precisely in a dark environment with low power consumption.[20]

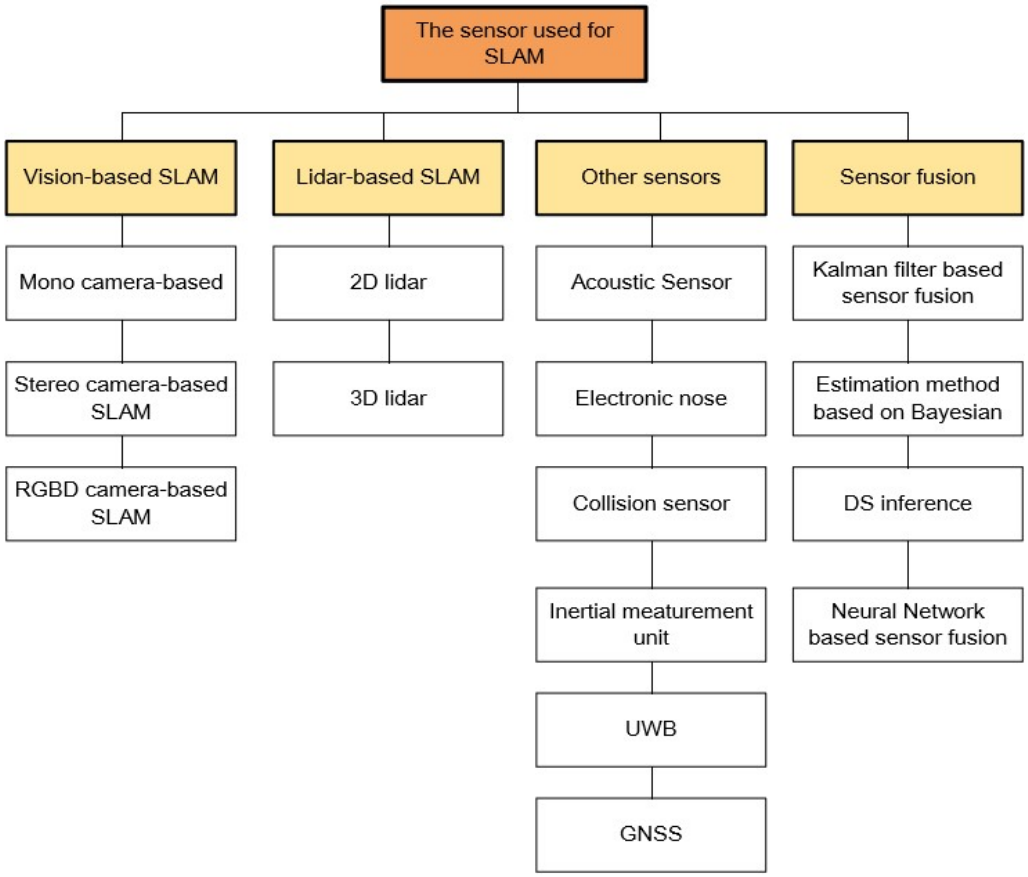

Figure 1.4 Category of sensors can be used in SLAM

There are also some interesting sensors which can accomplish the SLAM tasks like the electronic nose. It could be used with the odometer to smell different landmarks in the dark environment to generate a feature-based map. It can also be used to assist the loop closure process.

- Sensor fusion

Except to use more sensors, there are also some methods to combine the information of the sensors. Sensor fusion is now becoming a trend in SLAM system design. There are several methods which can handle sensor fusion.

1) Kalman Filter-based sensor fusion

To deal with the gaussian noise such as the observation noise and noise generated by the leakage of the circuits, the Kalman Filter-based (KF-based) methods like EKF[21] or Particle Filter (PF)[22] can be used to solve these problems. The KF can't be used directly because it can only handle linear circumstances. The low-level redundant sensor information can be deleted by using these methods.

2) Estimation method based on Bayesian

The mainly used methods are Bayesian based-methods. In order to combine the information from camera and (IMU), the method called Visual-inertial odometry (VIO) has been presented. In VIO, there are two types of methods to couple the data, one is loose coupling. It couples two types of information and uses them separately. The overall architecture of it is simple, but noise of each sensor will add errors to the whole calculation. Another is the tight coupling which first collects the information from each sensor, and corrects them with others. It can dramatically increase the accuracy of the whole calculation. The researchers have presented a method called VINS-Mono.[23][24][25] A tightly coupled, nonlinear optimization-based method is used to obtain highly accurate visual-inertial odometry by fusing preintegrated IMU measurements and feature observations.[26]

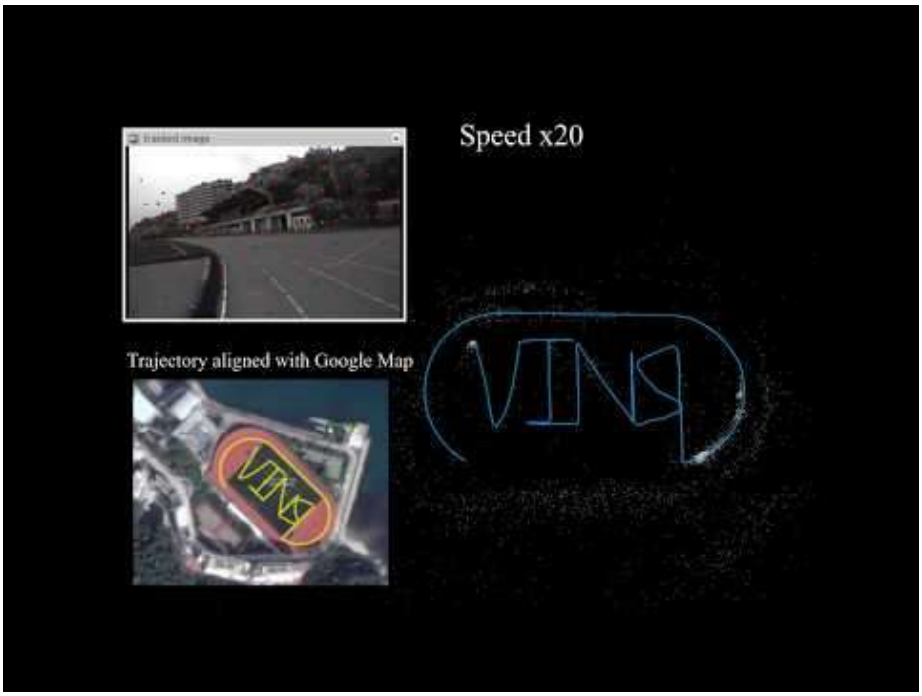

Figure 1.5 The result of VINS-Mono[27]

3) Dempster-Shafer (DS)[28] inference

This method can handle the problem that the lidar sees an obstacle in a specific place, but the camera can't see anything at the same time. These problems can be solved by voting the environment's situation by the belief or confidence of each sensor.

Normal Distribution Transform(NDT)[29] is an algorithm designed for matching in a gridded map. Some researchers have used the NDT+IMU to realize the localization by using DS inference.[30]

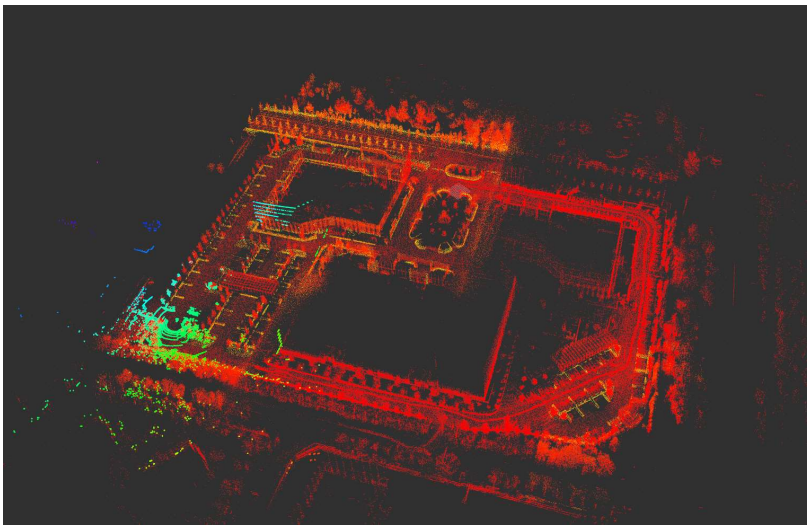

Figure 1.6 NDT-based sensor fusion[30]

4) Neural network-based sensor fusion

Sensor fusion can also be done by neural networks. An appropriate structure for the network has been chosen first. Then the network has been trained to get suitable weights. Finally, the information from different sensors can be combined to illustrate the status of the map with the pose of the robot.

This method is robust enough to get rid of errors. After getting the weight of the networks. The knowledge about how these sensors are combined can also be known. The researchers now have published a method which can combine the wheel encoder information with the collision sensor to realize the SLAM tasks.[31]

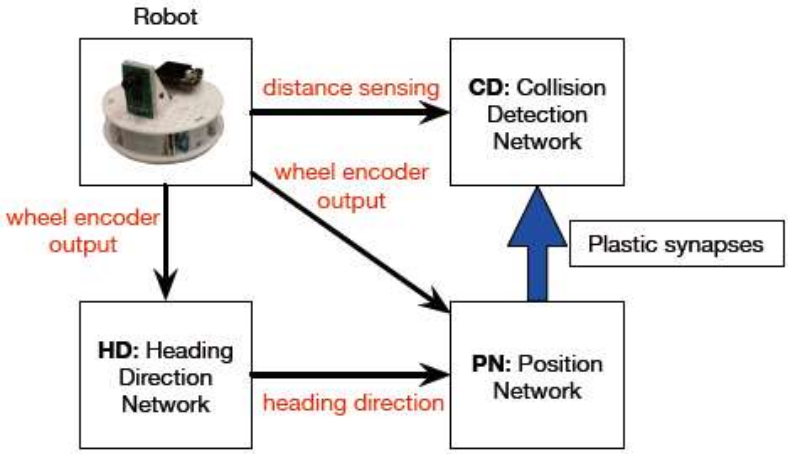

Figure 1.7 Pose estimation and map formation with neuromorphic SLAM[31]

1.2.2 SLAM system based on different algorithms

Different SLAM algorithms have been used on different robots. After choosing a suitable environment and a small robot. Different steps are needed to realize the whole

implementation.

In the area of lidar-based SLAM, There are some popular solutions like FastSLAM[32] and Gmapping[33]. These filter-based methods are based on 2D lidar, it assumes that the current pose is only affected by the previous pose and the previous control or movement signal. Gmapping uses the traditional PF method to predict and calibrate the pose of the robot. This method is limited by the quality of the particles, it is suitable in a small-scale environment for service robots. For the outdoor environment, increasing the amount of particles is required for the SLAM system which will dramatically increase the computation cost. Without the loop closure process, filter-based method also can't operate robustly and always accumulate errors in outdoor environments.

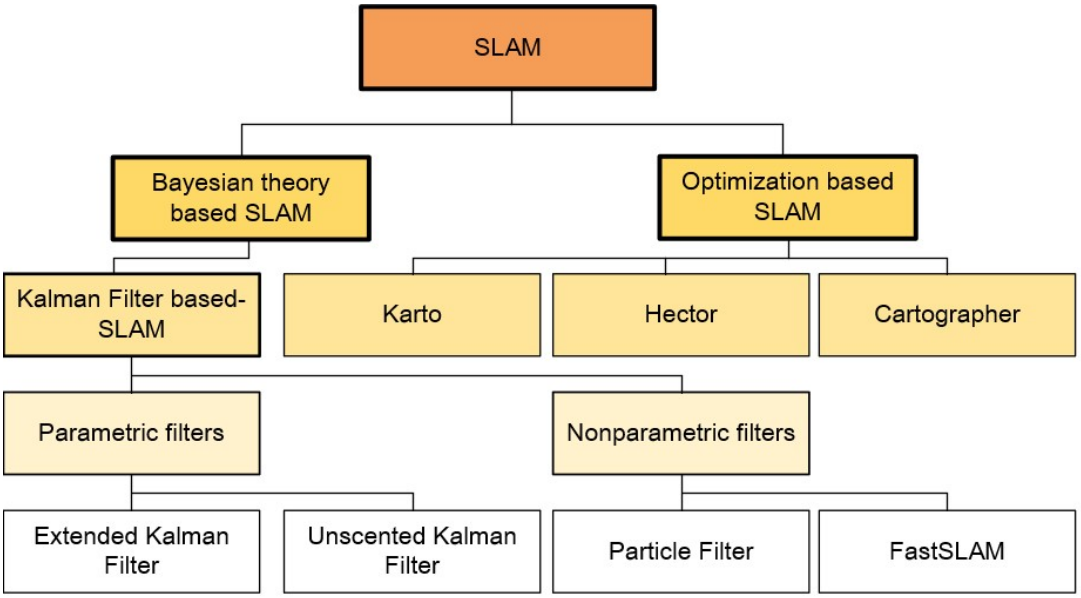

Figure 1.8 The category of SLAM

Except for the methods above, there are also some other solutions which assume all poses have some connections to some extent. The method called Bundle Adjustment (BA)[34] could optimize all poses together. In the 2D lidar-based SLAM area, there are also some algorithms based on this assumption like Hector[35], Karto[36], and Cartographer[37]. These methods all support the loop closure process to get rid of the error accumulated by the sensor noise. In Cartographer, it uses a branch-and-bound approach for computing scan-to-submap matches. It therefore is robust and has low computation complexity.

In the 3D lidar area, the most popular solution is LOAM[38], this method can

calculate the curvature of the feature points in a single scan line and compare them with their neighbors. It can also filter out the bad features like the features on the boundary of an occluded region or feature points on a surface roughly parallel to the laser beam. This method increases the quality of point cloud matching and registration and allows maps in real-time.

Table 1.1 The comparison between different lidar-based SLAM methods

Algorithm Name	Filter based or not	Application
Fast SLAM	√	Indoor and fast
Gmapping	√	Indoor
Karto		Both but hard for hardware
Cartographer		Both but hard for hardware

Because the LOAM has bad performance on the road in the countryside. LeGO-LOAM[17] therefore appears and it has been optimized for pose estimation on ground vehicles. This method will use the ground plane to increase the accuracy and performance of SLAM compared with LOAM.

In the area of vision-based SLAM, extracting the features should be done first. Matching the features just like the LOAM should be done next. There are different feature extraction methods like FAST[39], SURF[40], SIFT[41], and ORB[10] with different speeds and feature quality. These feature extraction methods can adapt to various environment. For the environment which needs a real-time reaction, FAST is recommended to extract the features. FAST first selects a candidate feature point, then it calculates its surrounding points. If the candidate point has 12 or more contiguous pixels which are brighter than a specific threshold, this point could be considered as a feature point. FAST will also take fewer computation resources. If high-quality features are required to perform reliable matches between different views of an object or scene, the feature extraction methods like SIFT are also needed for the computation. SIFT will take much more computation resources than FAST and it will also be slower. ORB adds a very fast and binary descriptor based on BRIEF to check whether it is the right FAST features matching or not, it is also a good method for feature extraction. The ORB-SLAM[10] based SLAM is extensively used in lidar-based SLAM.

1.2.3 SLAM system based on different platforms

If the parallel computations could be done by the hardware platform like FPGA or ASIC to optimize the overall performance, it could increase the speed of feature

extraction by 1000 times[42]. Therefore, low-cost hardware devices could be used to increase the speed of computation with low power consumption. In order to reach this destination, a hybrid system of both software and hardware has been used. A comparison between the hybrid system and the hardware devices with VHDL has also been made.

In another example of hybrid system, groups in National Taiwan Normal University optimize the vision-based process by using Nios II. It could accomplish the task of communications with FPGA. The FPGA will do the job of the feature matching and pose estimation. It can increase the speed of extracting SIFT features 121 times.[43]

For pure FPGA solution, the group mentioned above also designed a 2D lidar-based SLAM system with CFE.[42] It can speed up the process of Initialization, Prediction, CFE, Derivation of Likelihood, Landmarks Updating, and States updating. The group from the University of Bridgeport presents a mobile robot navigation mapping system based on neuromorphic technology[44] that counts the functional characteristics of the grid cell in an FPGA. The group from Tsinghua University has designed a Convolution Neural Network-based (CNN-based) feature-point extraction method for real-time visualization SLAM on embedded FPGA[45]. This system can run at a speed of 20 fps. The group from Aristotle University design a scan-matching genetic SLAM (SMG-SLAM)system[46] based on the principle of the PF. This system can generate a gridded map for further usage.

1.3 Thesis Work and Contributions

As introduced in previous section, the SLAM system has been used for different kinds of service robots. The demand for service robots which help to release the burden of elders will be increased, the scene in our thesis is the indoor environment at home. To fulfill the requirements of low cost and low power consumption, the 2D lidar is chosen as our main sensor of the robot to adapt to the dark environment like space under the sofa or space under the bed.

As shown in previous section, the feature matching methods based on Cartographer and LOAM are suitable for this SLAM system on 2D lidar. It is called Corner Feature Extraction (CFE) algorithm which uses curvature to judge the features' quality.

As mentioned in previous part, the software-based implementation for SLAM calculation has high power consumption and low efficiency in real-time processing.

The matching process in the front-end part of SLAM should be fast enough to produce real-time matching information. This information can also be translated into the information of odometry. This thesis therefore proposes a method of building a hybrid SLAM with both software and hardware, this method can dramatically increase the accuracy of this SLAM system and realize an FPGA-based SLAM system for scenes indoor. For the computation of CFE, an implementation of a hybrid system is used to speed up the process of feature extraction and feature matching.

For the back-end design, this thesis extracts the features based on the CFE algorithm and combines it with a genetic algorithm (GA)-based SLAM system, it can therefore dramatically decrease the usage of the computation resources in SMG-SLAM. This thesis can also simultaneously predict the robot's movement and calibrate the result with the observation of the environment.

The main work and contributions of this thesis are summarized as below:

1) A front-end with parallel processing of the SLAM system is proposed and designed. The part is based on CFE to extract the information of landmarks and store it into the map. The front-end of our system can perform parallel feature extractions and matches. Six parallel processing units are employed to accelerate the process to transfer the points from the polar coordinate system to the Cartesian coordinate system. It can increase the speed of Points Matching and Map Initiation by 360 times. Some improvements are made in matching the feature points and the registration of the feature points. The FPGA implementation therefore can increase more than 4 times of computation concurrency.
2) An efficient GA-based back-end of the SLAM system is developed. This thesis designs a PF based FastSLAM like SLAM system which can match and map in real-time. This part can adapt to the indoor environment with the localization and mapping in a relatively small hardware area usage. The proposed GA-based back-end is implemented in FPGA and integrated with the SLAM, leading to up to 5-time speed up after optimization for map initialization.

A complete SLAM system in the robot operating system (ROS) is implemented for evaluation. The front-end part of the system can recognize different objects' features and make real-time comparison and matching processes. The back-end part of the system can transmit the information of the surrounding environment to generate a feature-based map. It can also localize itself in real-time on the map. The SLAM system is mapped to FPGA with hardware-software co-design based on Xilinx PYNQ Z2.

1.4 Thesis Structure

Chapter 1 briefly introduces the background and state-of-the-art of the SLAM systems. elaboration of the SLAM sensors being used these days, the developments of the SLAM technology with the SLAM algorithm which was mostly used today are also described. After that, the overview of this thesis is provided.

Chapter 2 describes the main components of a typical SLAM system and the overall architecture of the SLAM design in this thesis. Different technology paths for different parts and their advantages and disadvantages are analyzed in this part. Finally, a structure that is suitable for our environment of application with low power consumption and high accuracy is selected in our design.

In Chapter 3, the front-end part of our GA-based SLAM system with VHDL is described. An introduction about the CFE algorithm with its limitation and how to solve these problems in the filter design is also given. The coordinate rotation digital computer (CORDIC) technology, AXIS, and DMA are also described in it to realize hardware and software hybrid system.

In Chapter 4, a GA-based SLAM system is designed with the PF structure. The principle of the PF is introduced. A hardware data-flow graph is given in this part. The ROS[47] platform is also introduced to simulate its function in the software environment.

In Chapter 5, the evaluation of the ROS is reported. Comparison between different methods in hardware area usage are also given with the analysis. The experiment environment and the devices are also shown in this part with their performance. The comparison of hardware utilization and calculation time are discussed.

In Chapter 6, a conclusion is made with future works.

2 The Overview and Design of the SLAM System

In this chapter, the overall architecture of the SLAM system is described with the SLAM system which has been designed in this thesis.

A typical SLAM system contains front-end part and back-end part[48]. The connection and functions of them could be seen in Figure 2.1. In order to decrease the power consumption, the long-term loop closure is replaced by the feature matching in the back-end as shown in Figure 2.2. Besides, imu data can be used in the back-end part to increase the accuracy. The whole SLAM system architecture is optimized in this thesis. The specific overall design is shown in this chapter.

The goal of this thesis is to use FPGA to implement SLAM with the function of visualization, mapping, front-end, back-end, and other tasks that need to be implemented in hardware. The front-end and back-end designs are realized in this thesis, the IMU can be replaced by the odometry data provided by the feature matching.

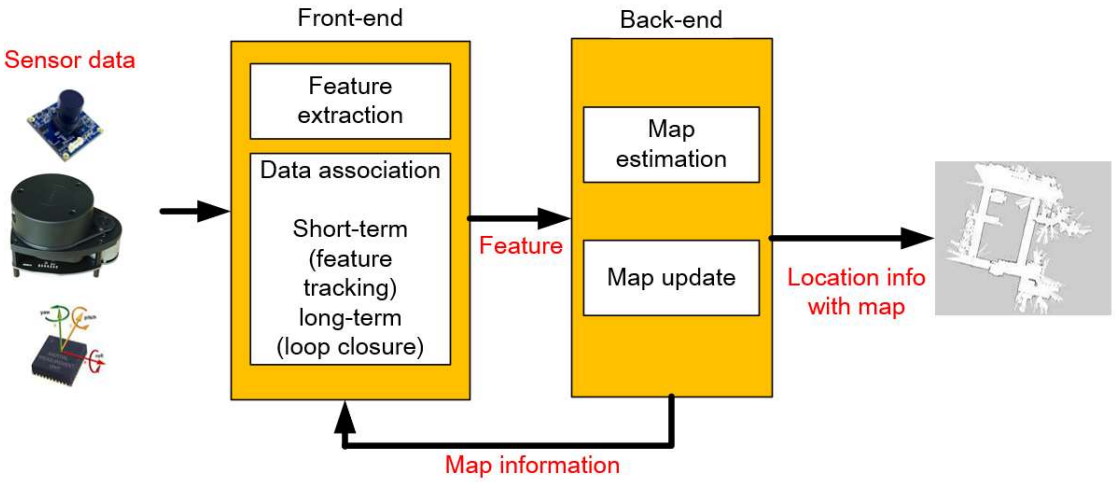

Figure 2.1 Front-end and back-end in a typical SLAM system.

The front-end of the SLAM first extracts the feature points from the data sent by the sensors, then to accomplishes the rough estimation of the pose for the robot or the agents. The pose made by feature matching both in the short-term and long-term can be estimated. The short-term process needs to match the features simultaneously to output the pose of the robots. The pose of 2D robots mainly contains three variables. These variables are x, y to represent the localization and the angle to describe the robots' head

direction. The long-term process is used to eliminate the error accumulated by the sensors' noise. This procedure is also called loop closure. Loop closure can also influence the map which is built in the part of the back-end.

The back-end part of the SLAM is a process of optimization. In this process, the back-end will translate the local coordinate system to the world coordinate system. After that, the feature points or other points can be registered on the map. This map information will be used in the front-end part. The loop closure process will also rebuild the map. The back-end part and the front-end part connect and influence each other.

The robots with the SLAM system contain both front-end and back-end can use some navigation algorithm like Dynamic Window Approach (DWA)to make the robots accomplish different movement tasks.

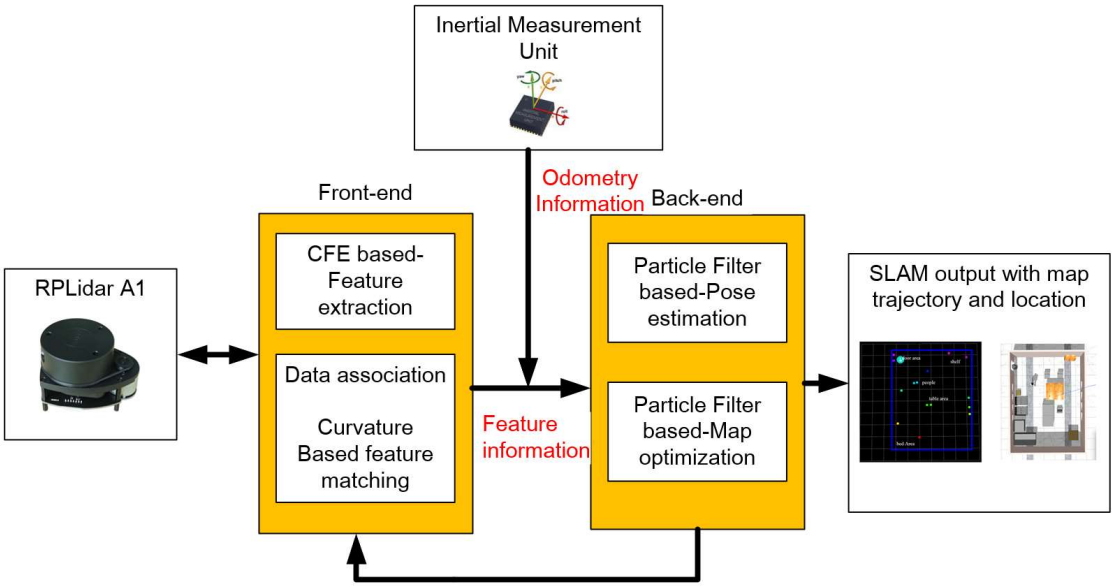

Figure 2.2 Overall architecture of the front-end and back-end SLAM in this work

2.1 Front-end Design of the SLAM system

The main function of the front-end part is the pose estimation based on the old poses and the map. It therefore should also be able to extract some features from the environment to match them in the current map to simultaneously update the pose of the robot. The sensors should have a high operating frequency or speed compared to the environmental changes. Otherwise, the SLAM can't get any information from the surrounding environment. When the sensor is collecting the data, an assumption that

the robot is relatively static can be made.

2.1.1 The feature extraction and matching in the lidar-based SLAM system

The feature extraction method used in this thesis is the CFE. It means that extracting the corner features from the environment by calculating its curvature is needed for other calculations. The feature with bigger curvature will be a more remarkable feature in the environment. In some simple environment, the curvature can also be the description of a feature for matching, but for complex environment, it is impossible to use it.

The most broadly used method for feature-points matching in lidar-based SLAM is Iterative Closest Point (ICP)[49]. ICP wants to find the best matching between point clouds. It is the same that the location on our map should be known by calculating the observed features.

ICP has two steps. (1) finding the point correspondence and (2) registering the corresponding points. ICP algorithm is iterative, repetitions of step (1) and step (2) are needed until convergence.

In order to realize the process, some mathematic assists are needed. The first one is the Singular Value Decomposition (SVD).

Calculation of the centroid position $p, p^{'}$ of two different point clouds which need to be matched should be made first. The p_i represents the first point cloud's points, the $p_i^{'}$ represents another point cloud's points.

Then the rotation between two point clouds without the centroid can be calculated.

$$q_i = p_i - p, q_i^{'} = p_i^{'} - p^{'} \qquad (2-1)$$

Then the rotation matrix is calculated by Equation (2-2). The R^* is the rotation matrix. The most suitable R which makes the minimum value is calculated in Equation (2-2). In this step, the optimization method like Gauss-Newton Algorithm is used in this Equation.

$$R^* = arg \min_R \frac{1}{2} \sum_{i=1}^{n} |q_i - Rq_i^{'}|^2 \qquad (2-2)$$

According to the rotation matrix got in the last step, the movement t^* between two points clouds could be calculated. Now the rotation and movement of the two robot's poses are calculated.

$$t^* = p - Rp' \tag{2-3}$$

In Equation (2-2), a non-linear optimization method is used. It can find the best variable by iterating.

For lidar-based SLAM, there is also a method called NDT[29]. It is a method aim at 2D lidar. In this method, the 2D plane is subdivided into cells. It is mainly used in the gridded map. To each cell, a gaussian distribution will be assigned with it which can models the probability of a point. The most important part of the NDT is calculating the NDT parameter in the first scan and initialize the whole map space, it is first described below.

The 2D plane is first divided into different cells. These cells have the same area and size. For each cell which contains at least 3 points. Collect all the points x_i as Equation (2-4).

$$x_{i=1,\dots,n} \tag{2-4}$$

Then, the mean coordinates q could be calculated by Equation (2-5). n is the number of points which lie in the cell.

$$q = \frac{1}{n}\sum_{i=1}^{i=n} x_i \tag{2-5}$$

Now the covariance matrix Σ could be calculated with the normal distribution of this cell as Equation (2-6) and (2-7) below.

$$\Sigma = \frac{1}{n}\sum_{i=1}^{i=n}(x_i - q)(x_i - q)^T \tag{2-6}$$

$$p(x) \sim exp\left(-\frac{(x-q)^T \Sigma^{-1}(x-q)}{2}\right) \qquad (2-7)$$

Comparing with the gridded map whose value of each cell in gridded symbolize the probability of its occupancy, the probability in this cell represents the probability that the point lies in this location within this cell. It is the end of the calculation of the first scan. The overall steps are as below.

1) Calculate the NDT of the first scan.
2) Predict the pose of next time step.
3) Map the points in the second scan into the coordinate system of the first scan.
4) Determine the corresponding normal distribution for each mapped point.
5) Calculate the score by sum the distribution for each mapped point.
6) Create a new parameter to optimize the score.
7) Go back to step 3 until meeting the convergence criterion.

After these steps, the exact movement between two timesteps is known. The system could update the map and prepare for the next time step with the new map.

2.1.2The loop closure design in front-end SLAM

There are three main methods in Lidar front-end SLAM to realize loop closure. They are loop closure methods called scan-scan, scan-map, and map-map.

The scan-scan is the most unstable algorithm in the front-end part for lidar-based SLAM system. it could not be seen as a loop closure method because it only compares the current scan with the last scan. It is impossible to make the comparison directly because of the uncertainty of the lidar. Even the robots are stable, the current scan will be different from the last scan. Different points on the same landmarks in these two scans could be seen. The scan-scan method therefore has low accuracy. There is no front-end SLAM system which uses it in the market.

The scan-map method is the most commonly used method with high accuracy and can be used in rea-time. The principle of it is to compare the current scan with the local map or submap which has been optimized by multiple scans. scan-map therefore has high accuracy. The Cartographer from google uses it in its front-end SLAM design.

The method with the highest accuracy is. It uses a submap made by the current scan to match with the submap in the global map to publish odometry information. Because a large number of computation resources are needed for this method, it is hard for the researchers to realize map-map algorithm in real-time. But this method will be the future of the front-end SLAM design.

2.2 Back-end Design of the SLAM system

The principle of back-end design is to filter out the bad odometry information from the front-end part to optimize the location and the map. The back-end part should also be able to initialize or refresh the map simultaneously based on the sensor data for the front-end part's usage.

Nowadays, there are two main directions for SLAM in back-end optimization. One is the filter-based SLAM and another is graph optimization-based SLAM.

The filter-based SLAM system[50] is mainly based on the Bayesian theory and Markov assumption which assumes the current state of the robot or the agent only has connections with the last state of it and the control variable it had between the last state and the current state. The prediction therefore can be made by the Bayesian inference. In the next step, the system will combine the prediction state with the observation value. Finally, the system can have the best relative accuracy result after the optimization of the noise.

There are lots of filter-based back-end SLAM algorithms like KF[51], EKF[52], unscented Kalman Filter (UKF)[53], PF[54], etc.

Because the status matrix and observation matrix which the KF needed should be all linear and it is impossible for us to have linear status or observation in reality, it is impossible to use it in our application. Therefore, EKF is discussed in this thesis. This method uses Taylor series-based linearization approximation on a certain point to make the matrix to be linear. UKF uses weighted random linear regression to perform random linearization, it has higher accuracy than EKF. Different from the KF-based methods which have been mentioned above, PF is a non-parametric implementation of the Bayesian filter. Its principle is to use a set of random state samples to express posterior probability, the steps of its computations are as below.

1) GA-based PF prediction

The purpose of particle prediction is to generate a large number of particles in the environment. These particles will move according to the state transition matrix and can

be used for weighting to approximate the posterior probability density later. It is just like generating a large number of individuals in the GA.

2) GA-based PF reweighting

When the particles have received the observation information, each particle will be calculated with a relative weight according to its possibility. The calculation about the chance of survival and leave offspring for each individual in the GA is made in this step.

3) GA-based PF resampling

This thesis uses the number of particles in the specific area to simulate the probability of particle appearance in this area. The resampling process, therefore, deletes all the old particles and regenerates the new particles by the probability or weight which is mentioned above. Like the GA, the individual who adapts to the environment will survive and have more chances to have offspring. The new particles will also be used to do the prediction step recursively.

4) Map estimation and optimization

After the particles have been converged, an estimation of their trajectory with the landmarks could also be made. The trajectory can be used to optimize the whole map.

The SLAM system based on graph optimization is mainly constructed by BA. This method arranges the poses of the robot in time sequence with the information of

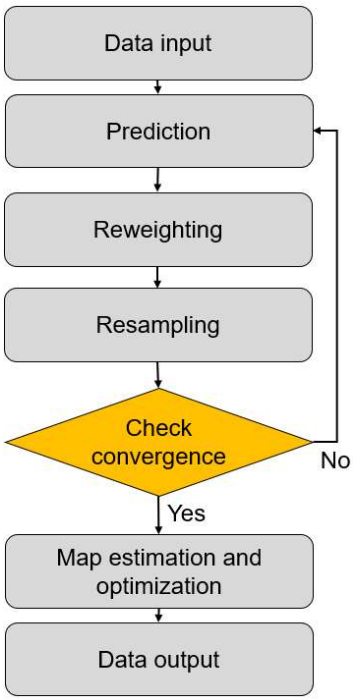

Figure 2.3 Genetic Algorithm based PF

surrounding landmarks in a matrix. It will get a sparse matrix that can be solved through

a special matrix solution method to obtain the latest odometry information and make real-time predictions of the pose.

Limited by the area of hardware in the low power consumption platform, the robot used in this thesis is not suitable for large-scale matrix operations. At the same time, the accuracy of the PF algorithm is generally higher than that of EKF and other KF based algorithms. It is also more robust for the robot with variable speed control and steering. What's more, the method based on PF is easier to implement in parallel. This method therefore can run in real-time. However, directly applying the PF to the SLAM system is doomed to fail due to the large number of variables required to describe the surrounding environment. This thesis therefore uses the Rao-Blackwellized Particle Filter (RBPF) to adapt to the SLAM architecture.

2.2.1 The map in the back-end part

At present, there are three different types of the map which were mostly used in the back-end part of SLAM[55]. One is the topology map, which records the pose changes of the robot and its related motion nodes. This action can only be reproduced, so it is difficult to perform alternative tasks with different changes for the topology map. Besides, this map is also difficult to adapt to large and complex scenes. It will fall into a local optimum in the navigation problem. However, compared to the grid map, it takes up less memory and runs more efficiently in small-scale environment. With the improvement of hardware performance, this method has gradually withdrawn from the stage of history. The grid map is the most-used map in RatSLAM like designs.[56]

The second type of map is the grid map, which quantifies the map of the entire area as the corresponding points. It is the most extensively used type of map in SLAM. Each grid point has three states which are undetected, unoccupied, and occupied. This kind of map contains more map information, which is more conducive to the implementation of arbitrary navigation tasks. This kind of map is more scalable than topological maps, so it has a wide range of application scenes. But at the same time, the grid map is difficult to interface with the symbolic problem interpreter, and its requirements for pose estimation are very high and will take up more memory space and hardware resources.

The last type of map is the feature-based map, which is different from the grid map and the topology map. The grid map stores the information of the entire location, while the feature map such as the FastSLAM algorithm only needs to store some significant

features or landmarks. Only storing the data of the features and landmarks will greatly reduce the memory space occupied by the normal points. This thesis uses the feature map to construct the whole SLAM system. In the part below about visualization, this thesis maps the feature map to the actual map for better performance. It should be noted that this thesis does not use the grid map for map construction.

2.2.2 Other consideration in the back-end SLAM

The basic principle of loop closure is that when the robot recognizes the same scene comparing with the previous scenes, it gradually changes the trajectory of itself with the optimization of the previous map by calculating the offset value of each step. It will also change the world coordinates of the corresponding sensor data to correct the cumulative error.

In addition, there is also a problem of movement sequence in a single scan. The environment cannot be seen as static when the lidar is scanning. This mechanism will produce motion blur which causes the objects in the environment that were originally straight to become curved. Therefore, estimation of the relative time and coordinates of each lidar point based on the lidar acquisition frequency and the speed of the robot is needed. Estimating the data coordinates of the lidar point and its relative position to remove motion blur is critical in running objects with high speed.

2.3 Summary

This chapter introduces the basic structure of a standard SLAM system, which usually includes a front-end part and a back-end part. The front-end part is responsible for calculating the current pose of the agent by extracting and matching the feature points, which includes the direction and the location of the robot. When the front-end part recognizes a familiar scene in the environment, it will start the process of loop closure. The back-end part is responsible for correcting the cumulative error generated by the front-end and initializing or optimizing map data in real-time. The generated map and the current pose of the robot can be used for path planning and navigation in the future.

After comparing the various algorithms, the PF-based SLAM system with the GA has finally been chosen in this thesis for this design with the usage of the feature map and the optimization without loop closure.

3 CFE Algorithm-based Front-end Design and Implementation

In this chapter, a description of the CFE algorithm-based front-end design in the SLAM system is made with the implementation. This front-end system is first described with the dataflow which contains the main function of the front-end system. Then, it comes to the overall hardware architecture of the front-end system which including the modules of CFE-based implementation with its connection. The specific module description including the processor system and programmable logic comes after the overall design. In the end, the prototype of the front-end system and its evaluation is made with the PYNQ Z2.

3.1 The Function Design of the CFE-based Front-end System

According to the SLAM system designed in Chapter 2. The front-end system can be divided into two parts. One is the feature extraction (Section 3.1.1 & Section 3.1.2) which needs to extract good features without noise, another is the feature matching (Section 3.1.3).

The aim of feature extraction is to find stable landmarks that can be used as features in the environment and find the descriptors of them. Therefore, the CFE method is chosen in this thesis, which will extract the corner points as features, and use the curvature as the descriptors.

In order to increase the efficiency of feature matching, the current scan would be compared with the previous scan only with the descriptors from the feature extraction. Further optimization will be done in the back-end system.

3.1.1 CFE algorithm-based feature extraction algorithm

In order to realize fast feature extractions in real-time, this thesis references the design of the CFE in the LOAM[17] and applies it to the 2D scene. The specific description is as follows.

As shown in Figure 3.1, the blue point in Figure is the center of the lidar. The green lines represent lidar rays that are emitted from the center and rotate clockwise sweeping through points A, B, C, and D. The blue lines represent the targets in the environment,

which are the obstacle or walls.

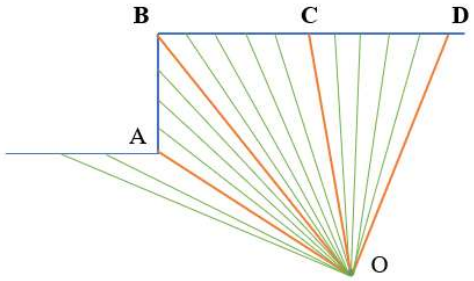

Figure 3.1 The illustration of the CFE

The points sampled by the lidar are all in the polar coordinate system. The conversion from the polar coordinate system to the cartesian coordinate system should be made first by Equation (3-1). In this Equation, θ means the angle data received. d means the detected distance at this angle. Therefore, CORDIC is needed for the data processing module in Section 3.2.1.

$$x = d * cos\theta, y = d * cos\theta \qquad (3-1)$$

Then the two vectors $\overrightarrow{AB}$ and $\overrightarrow{BC}$ built by the lidar points on the wall are taken to calculate its dot product. Then, find the molds of the $\overrightarrow{AB}$ vector and $\overrightarrow{BC}$ vector. Divide the dot product of the two vectors by the product of the two molds to get the projection ratio α between $\overrightarrow{AB}$ and $\overrightarrow{BC}$, which is the descriptor needed for feature matching in Section 3.1.3. The whole process could be seen in Equation (3-2). p means vector $\overrightarrow{AB}$. q means vector $\overrightarrow{BC}$.It can be seen that when the descriptor is equal to 0, the feature it describes is a vertical corner point, like the $\angle ABC$ in Figure 3.1. When the descriptor is equal to 1, like the $\angle BCD$ in Figure 3.1, it could not be regarded as a corner, it could only be regarded as a plane.

$$\alpha = \frac{p.q}{|p||q|} \qquad (3-2)$$

Therefore, a threshold is set here. When a corner point's descriptor is less than the threshold and greater than 0, its location could be stored as the feature point and its corresponding projection ratio as a descriptor.

Extracting feature points by using this method will cause various problems. Filters or other methods are needed to solve these problems. The specific problems are as follows:

1) Multiple features error

In one scan, when the lidar scans through a corner, it may get several feature points. However, it only has one corner in reality. As Figure 3.2 shows, $\angle ABC$ and $\angle BCD$ could be recognized in the scan, but only one corner exists in reality.

When the receive module of the front-end meets sequence corner points, it will calculate its projection ratio and choose the one with the smallest ratio to be stored in the map. This error will be eliminated in the feature postfiltering module in Section 3.2.1.

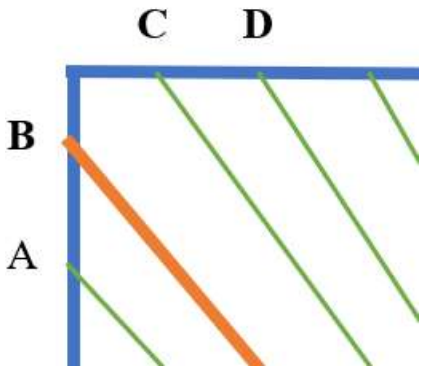

Figure 3.2 The example of the multiple features extraction from one feature in reality

2) Occluded region error

The objects in the environment can occlude the environment in its back. If the SLAM system only has lidar, it can't know the real environment of this area. On the boundary of the occluded, CFE will extract one feature point from it. This feature point

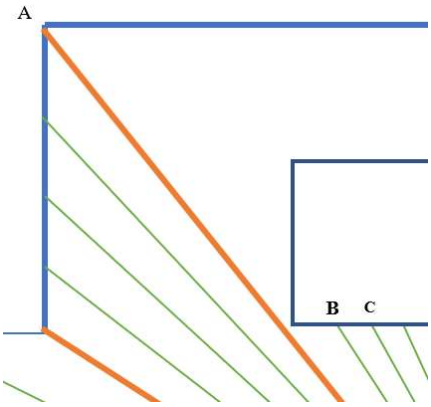

Figure 3.3 The example of occluded region error

doesn't exist in reality. Different features could be seen when the robot is moving. This error will be eliminated in the prefiltering module in Section 3.2.1.

In the occluded region, there is a large distance between its point. Like the $\angle ABC$ in Figure 3.3, the distance between A and B is too large, so these points will be deleted.

3) Parallel lidar ray error

The reflected ray quality of the lidar points is determined by their surface. If the surface of the point is approximately parallel to the ray of the lidar. The quality is extremely poor. Therefore, the prefiltering module will scan and identify the quality of the return rays of the lidar and delete the lidar rays with poor quality. Because the information of the quality is included in the lidar's output packages, bad points can be filtered out before the transform of the coordinate system.

4) Others

The lidar has a maximum and minimum recognition distance range. When the points appear within the minimum recognition range, they will be regarded as noise points. The point out of the maximum recognition range will also be deleted, thereby improving the robustness of the lidar. This error is also eliminated by the prefiltering module.

3.1.2 Design of feature extraction

Based on the CFE algorithm and its consideration above, the data flow of the overall front-end system is shown in Figure 3.4. The feature extraction is first described below.

1) Data pre-filtering

In this part, the lidar ray information with bad quality and unreasonable distance is deleted.

2) Sampling

Sampling is the input port of the CFE algorithm. It is important for the design in the front-end. If the sampling interval is too large, the feature could not be recognized. If the sampling interval is too small, the computation is too complex for the computation and waste too much hardware resources. The specific sampling interval is described in the implementation part. A first in first out memory component is implemented.

3) CFE processing

CFE processing based on the contents above has been designed. It can extract the

lidar information from the sampling part can calculate its curvature value.

4) Data post-filtering

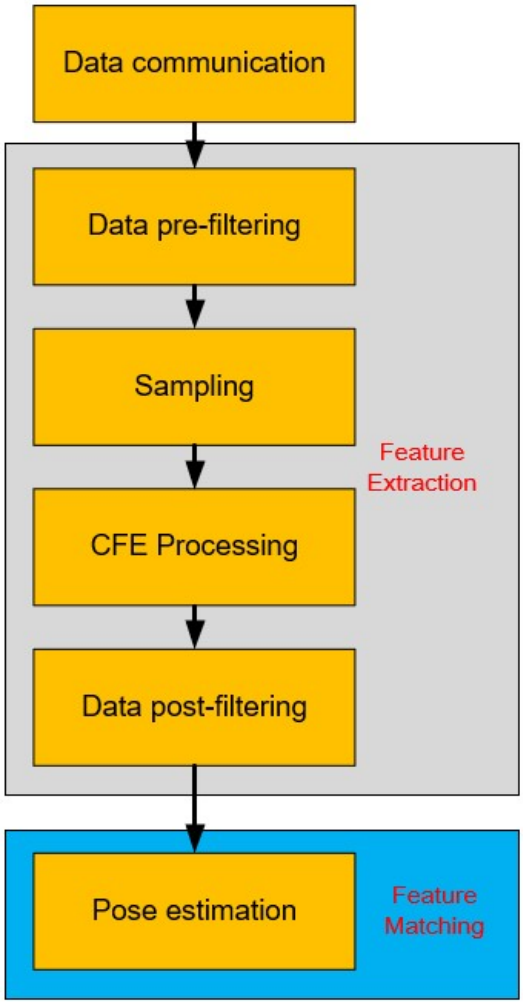

Figure 3.4 The data flow CFE-based front-end algorithm

The point generated by the occluded region will be deleted in this part with the points out of range or have low quality.

3.1.3 Design of feature matching

Pose Matching receives data from the CFE algorithm. It describes how to calculate the movement by a feature which has just been recognized. When the whole system just initializes, the pose matching part will store the features generated during the first scan cycle in the internal memory and calculate the centroid of the features.

Consider the lidar scan is much faster than the movement of the robot as prerequisites. When the information from the next cycle has arrived at the pose

matching, the present scan should be similar to the previous scan. The previous features should also be similar to the previous features. Therefore, the movement of the robot is the same as the movement of the centroid of features. As long as the Euclidean distance of the observed centroid is less than a certain value and the difference between its curvature meets a certain threshold, it can be considered as the same feature. The movement can be considered as the distance between two centroids.

Next comes the calculation of the rotation. Consider the previous feature points cloud as P_s consider the next feature points cloud as P_t. The P_t has been translated to the same coordinate system. Rotation R that meets Equation (3-3) should be found.

$$P_t = RP_s \tag{3-3}$$

So, the loss $F(R)$ can be calculated to evaluate the rotation in Equation (3-4).

$$F(R) = \sum_{i=0}^{N} ||RP_s - P_t||^2 \tag{3-4}$$

This Equation can be simplified to Equation (3-5). The question can be transformed to Equation (3-6).

$$F(R) = \sum_{i=0}^{N} (||P_s||^2 + ||P_t||^2 - 2P_t^T RP_s)^2 \tag{3-5}$$

R^* is needed to find the minimum $F(R)$.

$$R^* = \arg m\,ax_R(2P_t^T RP_s) = \arg m\,ax_R trace(P_t^T RP_s) \tag{3-6}$$

$$trace(P_t^T RP_s) = trace(RP_t^T P_s) \tag{3-7}$$

SVD[57] is used for the next calculation. In linear algebra, the SVD is a factorization of a real or complex matrix that generalizes the eigen decomposition of a square normal matrix to any $m \times n$ matrix via an extension of the polar decomposition.

Consider $P_t^T P_s$ as H to compute SVD. U is the left unitary matrix. Σ is the singular matrix. V is the right unitary matrix. The calculation is shown in Equation (3-

8).

$$trace(RP_t^T P_s) = trace(RH) = trace\left(RU\,\Sigma\,V^T\right) = trace\left(\Sigma\,V^T RU\right) \qquad (3-8)$$

In order to get the maximum trace. $V^T RU$ should be equal to I. That makes Equation (3-9).

$$R^* = VU^T \qquad (3-9)$$

Therefore, the rotation could be calculated correctly. Movement information with the rotation information will be sent to the back-end system. The calculates of the rotation and movement is shown in the pose estimation module in Section 3.2.1.

3.2 The Front-end SLAM System Implementation in FPGA

CFE is a lightweight feature extraction method which can be realized in parallel, it is therefore suitable to be implemented based on FPGA. Implementation of the CFE

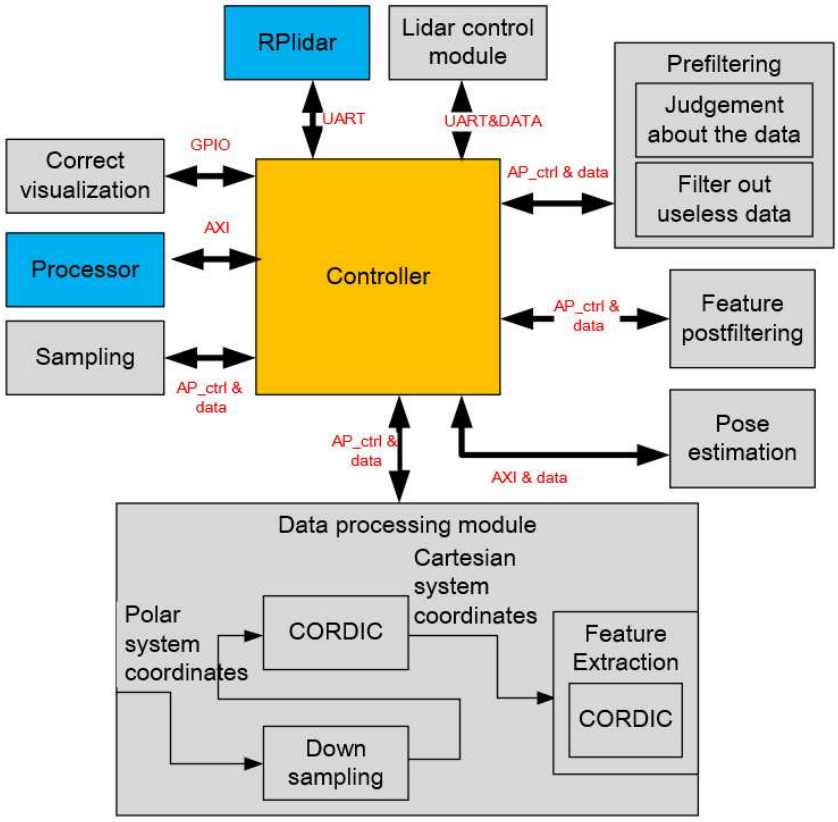

Figure 3.5 The VHDL-based FPGA architecture of front-end design.

algorithm-based front-end design in the SLAM system is described below. The hardware architecture is shown in Figure 3.5. It mainly includes the part of programmable logic part and the processor system part.

3.2.1 The implementation of hardware modules

In this thesis, implementation based on VHDL is finished. As shown in Figure 3.5, The gray and yellow modules in the architecture are designed in programmable logic. The blue modules in it are peripheral devices or modules designed by the processor system.

There are 8 main modules. The controller module is designed to control the overall front-end system like the data flow above. Other modules are described below.

1) Lidar control module

The lidar control module can be divided into two parts, one is the part to communicate reset signal with lidar, another is the part that communicates scan signal with lidar.

The lidar communication module is responsible for sending control signals to the lidar and receiving the signals from the lidar to put them into the prefiltering module. All data are transmitted through the UART (Universal Asynchronous Receiver and Transmitter) data protocol. This data transmission protocol is robust and can be simply implemented by hardware. This protocol is also very suitable for implementation by hardware description language.

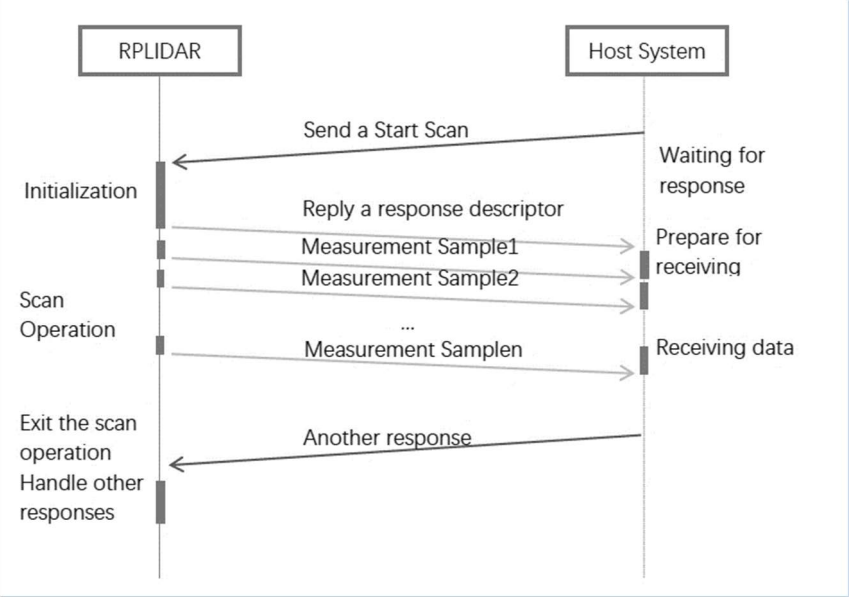

Figure 3.6 The communication process of the lidar scan[58]

The normal communication process of the lidar is shown in Figure 3.6. The FPGA board is connected to the lidar through the PMOD port. This thesis also implements the hardware of the UART port. When the button is pressed, the board will automatically send the start scanning command to the lidar, and the lidar will send the measured value back.

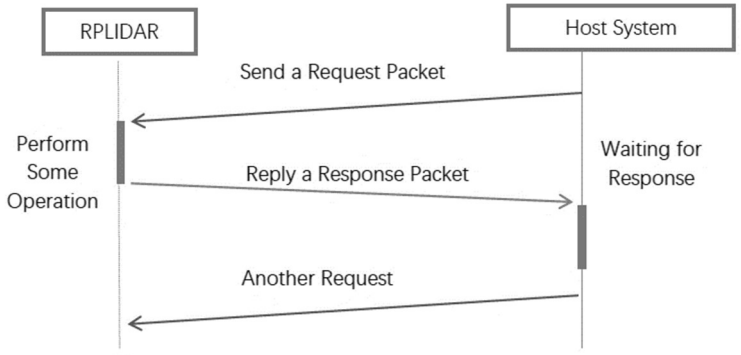

Figure 3.7 The processing logic of the reset signal[58]

The reset control signal can be sent by pushing the reset button on the board. The processing logic of the reset signal shows in Figure 3.7. It is slightly different from the scan control signal.

Based on the mechanism above the finite state machine of the lidar control module

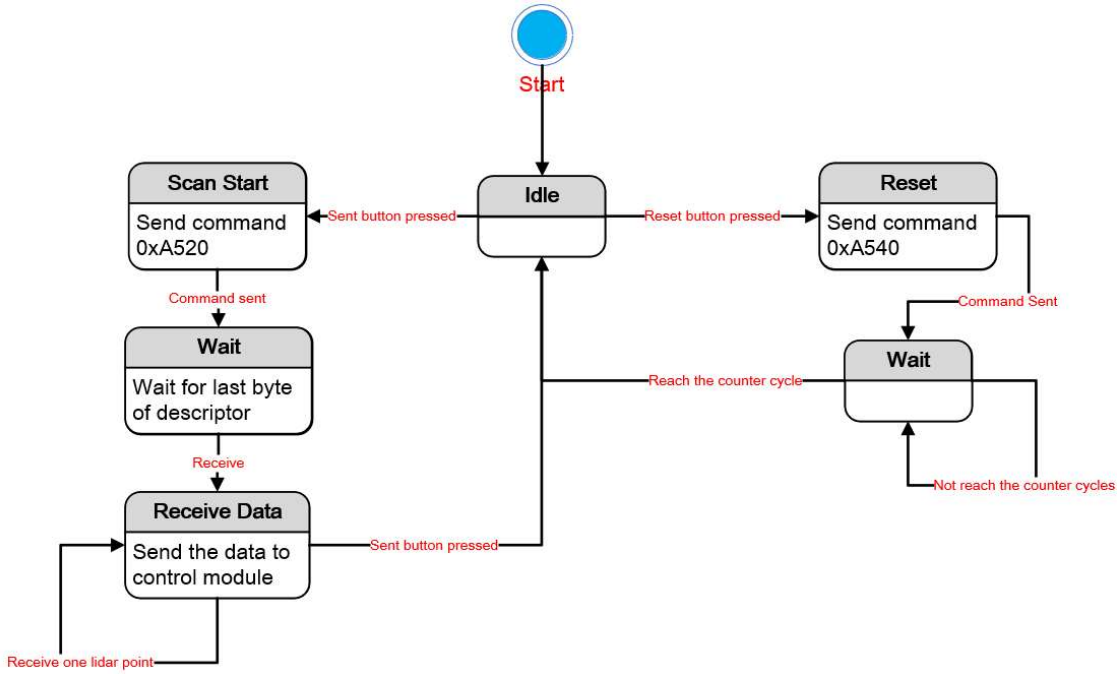

Figure 3.8 The finite state machine of lidar control module

is designed as Figure 3.8. It takes the waiting time into consideration. The received data

of the lidar control module is sent to the prefiltering module by the control module. The check bit information is sent to the correct visualization module.

2) Prefiltering module

As mentioned in Section 3.1.1, the prefiltering module is responsible for preprocessing the data from the lidar. The return data of the lidar contains some messages with errors or garbled due to timing issues. One of the subsequent goals is to remove the points on mirrors. The echoes returned from mirrors are extremely unstable and should not be used in feature extraction. Under the circumstances that the lidar beam is approximately parallel to the surface of lidar points, the quality of the lidar data is not high and should also not be used. Quality is the data received from the lidar. it can be judged directly. if the quality is too bad, the data pre-filtering module will wait for the next point input from the lidar control module.

Another consideration in this module is the distance of the point, if the distance is too large or too small, the data pre-filtering module will also throw the current value and wait for the next point input. The data pass through the prefiltering module is sent to the sampling module.

3) Correct visualization module

The goal of this module is to determine whether the format of the data is correct or not. If the format is right, the subsequent filtering process can be performed. If there is something wrong with the format because of a problem with the timing, the board will light up a led to display. Users can control the reset button to reset the lidar control module.

4) Sampling

The design of the sampling module is built by a First In First Out (FIFO) structure as shown in Figure 3.9, which is responsible for extract the data with three equally

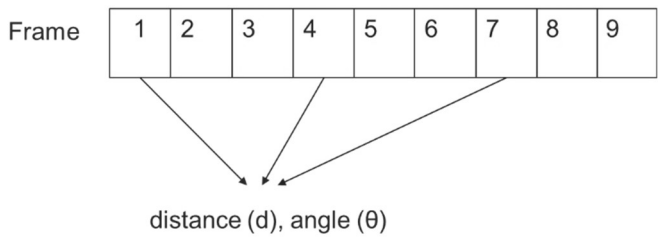

Figure 3.9 The design of data buffer

spaced points according to the time sequence and pushing them into the data processing module. Once the lidar input one point into this buffer, the buffer will throw the oldest

point like moving the sliding window. Although interval sampling of the points has been done in the buffer, points information will not be lost in the buffer. The data after sampling will be used to generate features in the data processing module.

5) Data processing module

The data processing module is responsible for calculating three points from the buffer into a Corner Feature by using the CFE method. This processing step includes the angle-to-radian conversion module. Besides, the CORDIC-based block is used in this thesis to calculate the sine and cosine values. After that, the coordinates of x and y can be calculated.

The data processing module is three-way parallel and is optimized by the design of the data buffer module, and this optimization can significantly increase the calculation efficiency by three times.

After obtaining the coordinates of the three points, the vectors of two adjacent points will be calculated and these two vectors will be multiplied to get the dot product.

Because the format used to calculate in VHDL is integral, the projection ratio must be 0 after the transformation of the value smaller than 1 to an integer. So, the feature processing module is designed to first multiply the dot production value by time, then it divides the result by the multiple of the vectors' molds.

When calculating the molds, it is necessary to use the square root calculation, which will also be implemented by using the CORDIC IP core in Vivado. After the calculation, the output values can be put into the feature postfiltering module.

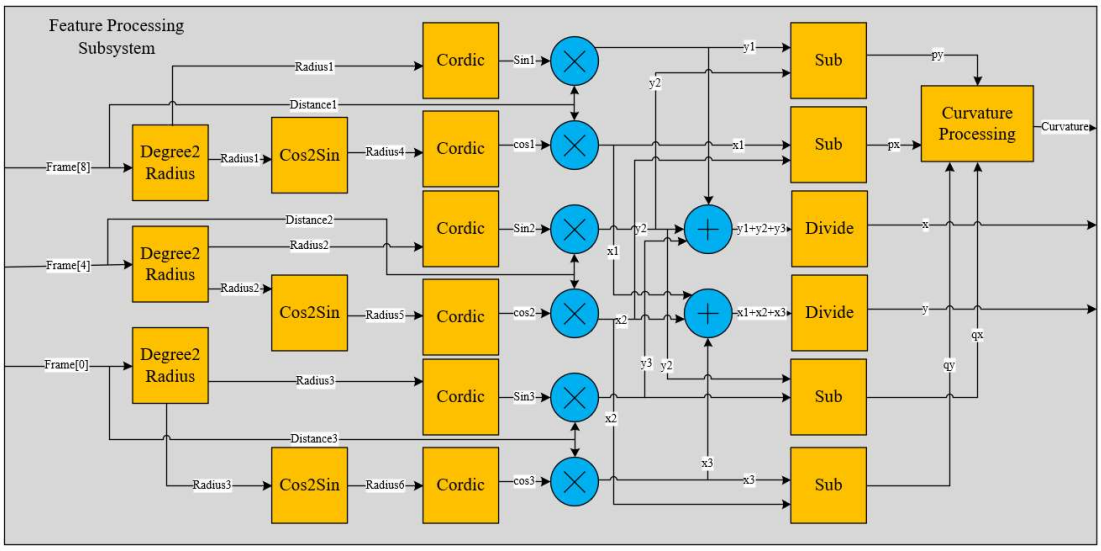

Figure 3.10 The feature processing dataflow

6) Data post-filtering module

Some of the situations mentioned in the previous Section cannot be directly eliminated when the receive module gets the lidar data, so it is necessary to add a post-data filtering module.

The multiple features error should be dealt with first in this module. Feature points that are close enough are combined into a set through buffers. In this set, the post-filtering module will select the point with the minimum projection ratio bigger than 0 as the most representative point to output and use it for feature matching. This point is also closest to the actual corner feature.

The last one is the occluded region error, the point which has a negative projection ratio will be eliminated. The features which have at least one mold of the vectors that are bigger than the threshold will also be deleted.

7) Pose estimation module

This module is engaged in estimating the odometry information of the current time step from the last time step. SVD can be simplified in 2D circumstances. Therefore, the matching process is made by constructing a comprehensive matrix of related poses with environmental feature points. The finite state machine of the pose estimation module is

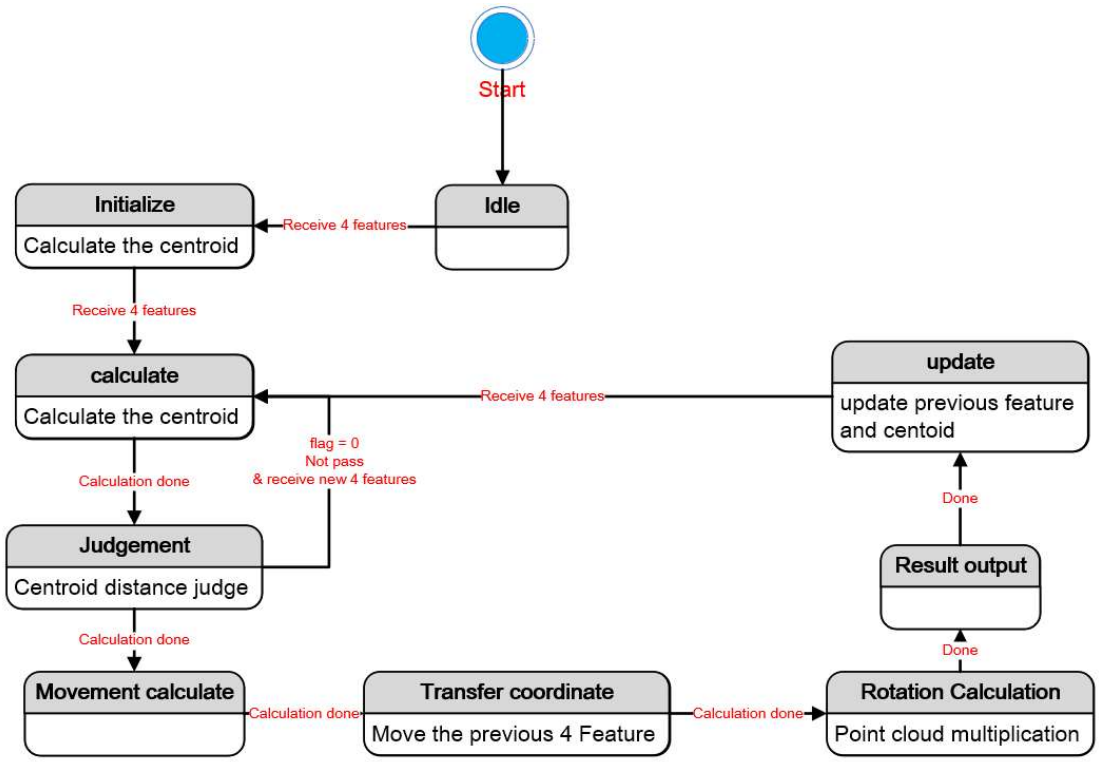

Figure 3.11 The state machine of pose estimation module

shown in Figure 3.11.

3.2.2 The implementation of software system

Next comes the implementation of the software system, the software system is the part which can connect to the laptop by LAN. The architecture of the PYNQ is shown in Figure 3.12. On the laptop, users can edit the processor system and program the programmable logic by using the bitstream file in the SD card.

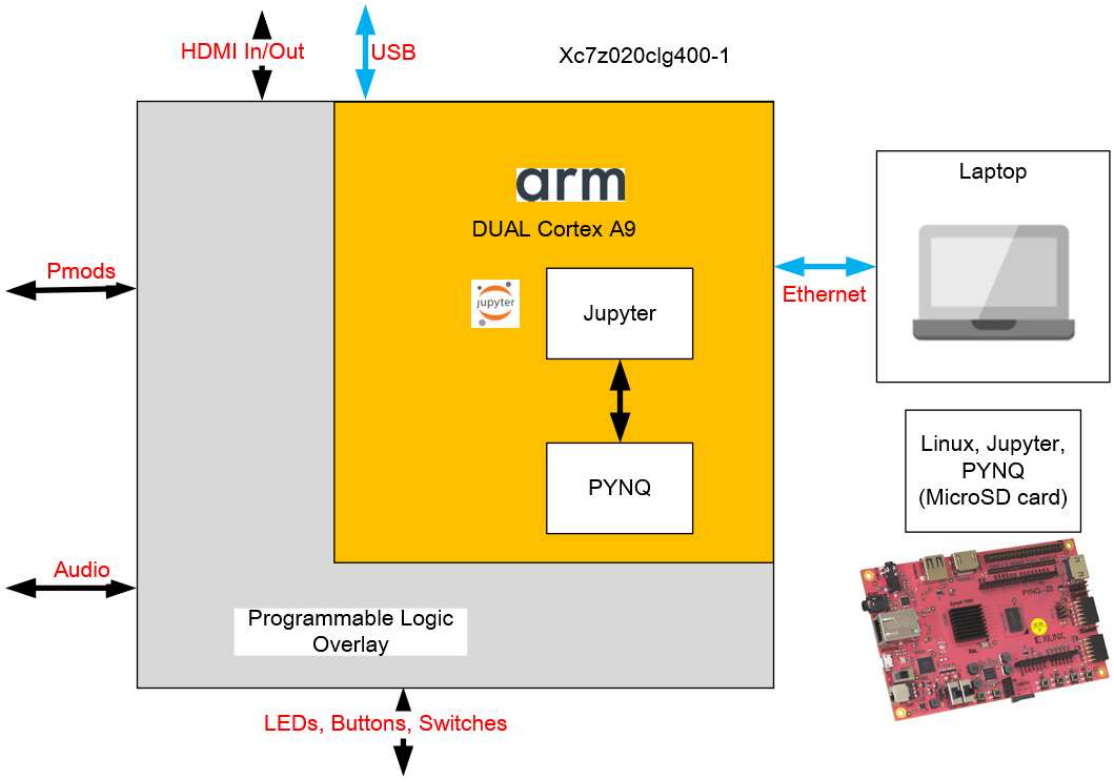

Figure 3.12 The architecture of PYNQ-Z2

By using the AXI interface, the processor system can easily connect to the programmable logic by just read the address and write the specific address. The program can be easily built by the library functions. The processor system will visualize the movement result with feature points.

3.3 The Implementation Results of the Front-end System

After the implementation of the front-end system, the hardware resources are calculated by the Xilinx Synthesis tools as Table 3.1 below. In order to evaluate its performance, the system is built according to Figure 3.13 to test the performance of the front-end SLAM system.

Table 3.1 The resource utilization of front-end system

Resource	Utilization	Available	Utilization Ratio(%)
LUT	13222	53200	24.853
LUTRAM	100	17400	0.574
FF	7986	106400	7.506
DSP	29	220	13.181
IO	12	125	9.600
BUFG	1	32	3.125

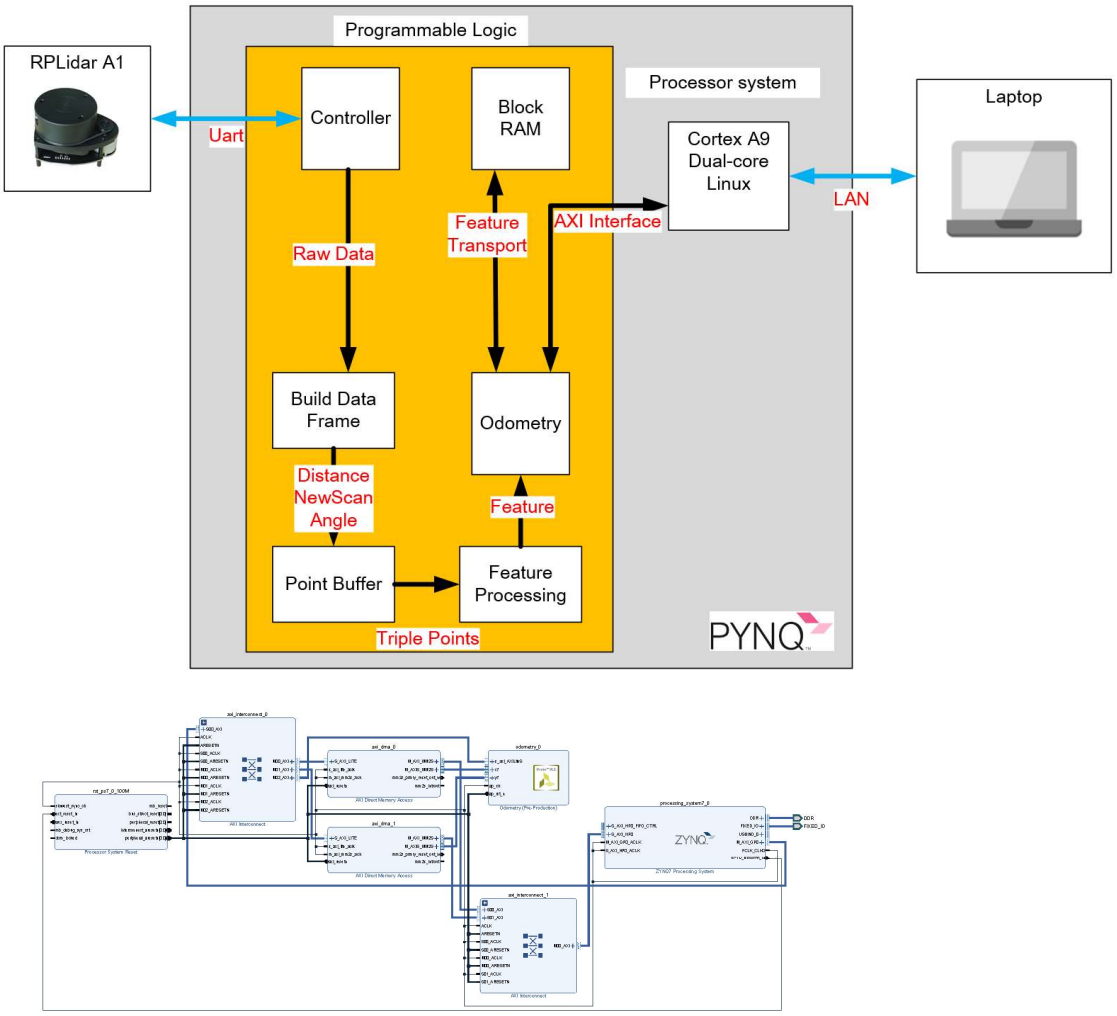

Figure 3.13 Architecture of front-end system with the block diagram in Vivado

3.4 The Evaluation Results of the Front-end System

As shown in Figure 3.13, the lidar displacement is calculated from a square experimental environment. The blue points in it are the normal points, the red points in it are the feature points. This graph is generated by the signal transmitted to the PC.

The calculation of displacement could also be calculated by these feature points. The experiments are set by ten times. The real displacement is determined by the ruler, which is 15cm in the positive direction of the x-axis and 20cm in the positive direction of the y-axis. This experiment is repeated by 10 times. The robot has no rotation in any direction.

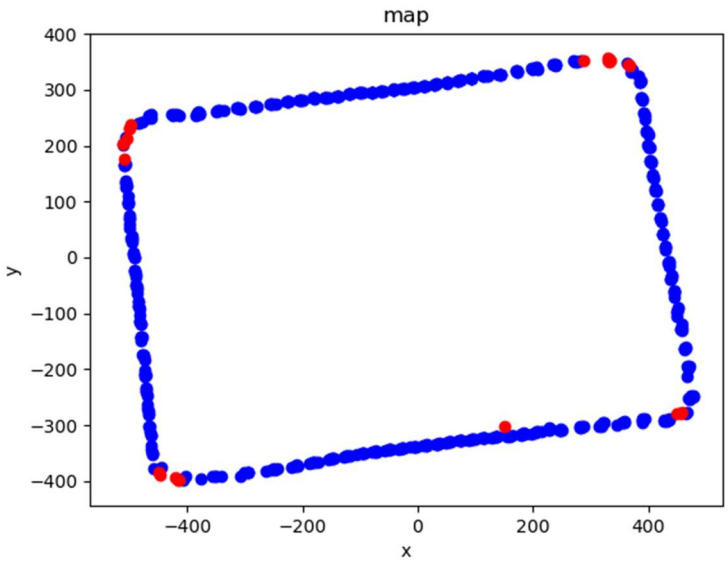

Figure 3.14 The feature extraction result from a square experiment environment

The detected movement can be shown in Table 3.2. The implementation has an average accuracy of 94%. The construction of the back-end part will be started in Chapter 4.

Table 3.2 The displacement experiments results.

Experiment number	X movement	Y movement	X accuracy	Y accuracy
1	14.013	20.979	93.42%	95.11%
2	13.619	20.717	90.79%	96.42%
3	13.981	21.268	93.20%	93.66%
4	13.727	21.241	91.51%	93.80%
5	14.061	21.054	93.74%	94.73%

6	14.493	20.611	96.62%	96.94%
7	13.711	21.429	91.40%	92.86%
8	14.456	20.933	96.37%	95.33%
9	14.335	20.905	95.57%	95.48%
10	13.710	20.851	91.40%	95.74%

3.5 Summary

In this chapter, the design for the front-end SLAM system is given. It includes the CFE based corner feature extraction and feature matching.

The implementation of building a front-end SLAM system in FPGA is also given in this chapter. The front-end system by VHDL is built in this chapter with the test in the square experiment environment. After comparing it with the ground truth, the accuracy of the front-end part allows the construction of the back-end part of SLAM.

In the next Chapter, the design and implementation of the back-end SLAM system will also be given.

4 GA-based Back-end System Design and Implementation

In this chapter, a genetic algorithm-based back-end system is designed. The back-end system in this thesis is based on GA. The GA in this thesis uses features to match with the global map. The feature map used in this thesis keeps the character of the grid map to increase the speed of matching. It is also a feature map which has low power consumption. It is therefore suitable for us to build robust architecture in the indoor environment, and can also be implemented in parallel.

The implementation of the back-end SLAM system is also described based on the design. The hardware architecture and the implementation of modules are given in this part. The connection between the front-end and back-end is also shown in this chapter. In the end, the hardware utilization has been mentioned with the analysis.

4.1 GA-based Back-end System Design in SLAM System

When the back-end system receives a front-end information bag which contains features, newest movement and rotation, the back-end system will perform the following three steps.

1) Particle generation with its pose and features transformation (Section 4.1.1)
2) Feature matching and new pose estimation (Section 4.1.2)
3) Pose update with the Map update (Section 4.1.3).

Step 1 generate the particles and move the feature of particles with the current pose to simulate the sensor noise. Step 2 filter out bad estimation generated by step 1 and remain the best pose estimation. Step 3 will use the pose estimation to update the current pose, the features of this pose will also be stored in the feature map. The pseudo code of the algorithm is shown in Table 4.1.

4.1.1 Particle generation with its pose and features transformation

In this Section, generation is based on the data generated by the front-end. The data generated by the front-end contains movement data (XY), robot rotation data (θ) and 4 feature data (XY). Based on the pose estimation result in the front-end, particles are generated to guess the real location. In Section 4.2, the parallel-based particle generation will be illustrated in the particle generator module to increase the speed of

this function.

Table 4.1 The pseudo code of genetic algorithm-based back-end SLAM

Algorithm 1 Genetic algorithm-based back-end SLAM(Simplified)

Require: robot movement, robot rotation, current pose

N = 64; movement = robot movement + current pose coordinates; rotation = robot rotation + current pose rotation; rand range = 16

function PARTICLE GENERATOR(N,movement,rotation,rand range)

for $i = 1; i < N; i++$ **do**

coordinates of $p[i]$ = movement + rand(-rand range,rand range)

rotation of $p[i]$= (rotation + rand(0,2*rand range))%360

for each feature in the each particle $p[i]$, transform its coordinates

function FEATURE MATCHING AND NEW POSE ESTIMATE

for $i = 1; i < N; i++$ **do**

for each feature in the particle $p[i]$ search the corresponding feature point

if find the corresponding feature point and their Manhattan distance <15 **then**

store the particle feature p[i] ;fitness value = fitness value + Manhattan distance

else Can't find the feature and all feature checked

throw the current particle

rotation = best fit particle rotation(with lowest sum of Manhattan distance)

movement = best fit particle movement(with lowest sum of Manhattan distance)

Set new parameter and use particle generator and feature matching or end

function POSE UPDATE AND MAP UPDATE

find the fitness particle, record its feature

After the generation of the particles, their features should also be changed to fit the new coordinate system. The specific steps are shown as follows.

The calculation can be done by Equation below. The p represents the local coordinates of the landmarks. The p' represent the global coordinates of the landmarks. The R is the transform matrix which is decided by the head direction change of the robot. t is the displacement of the robot.

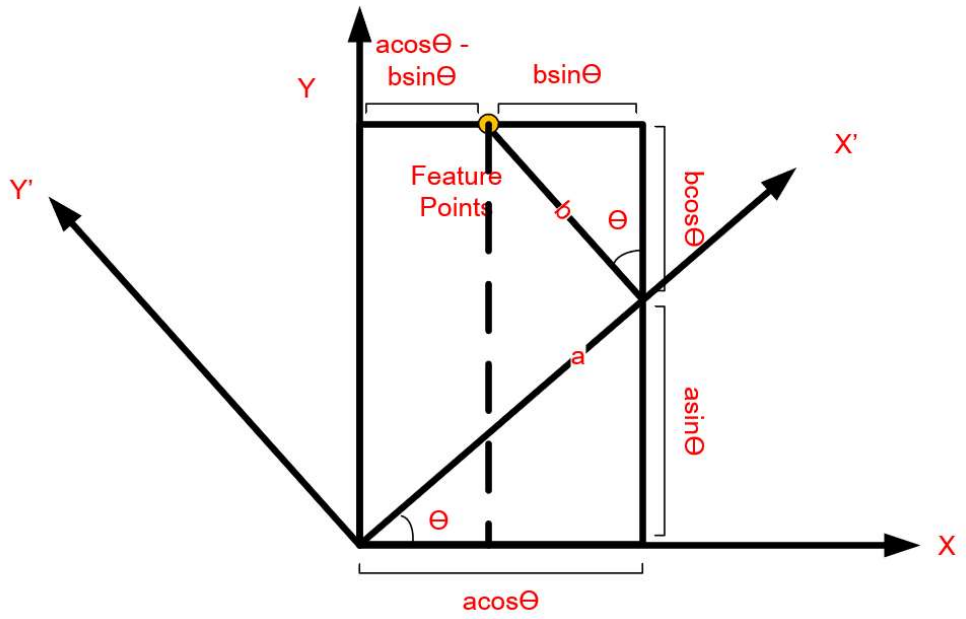

Figure 4.1 The calculation of rotation matrix

$$p^{'} = Rp + t \qquad (4-2)$$

θ is the rotation angle compared with the world coordinates. Rotation matrix R can be described as Equation below.

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \qquad (4-3)$$

Figure 4.1 can describe this matrix properly. As shown in Equation 4-3, the sin and cos calculations are needed for transformation, this will also be implemented by CORDIC.

4.1.2 Feature matching and new pose estimation

During this step, the data from particle generation is used to check the correctness of guessing. The data including the N particles and $4N$ particles' features. Then comes the matching process, each feature will find its corresponding feature in the whole map space. It is hard when the map is big and needs too much calculation

resources for comparing each of them. A special memory allocation design is given in map memory from Section 4.2, it will divide the memory space into different cells. The corresponding feature can be found in the same cell. It will dramatically decrease the comparison time.

During the comparison, the Manhattan distance is used to calculate the distance between particle's feature and global map's feature. Particle's fitness is decided by the sum of all features' distances.

If the corresponding features can't be found in the specific particle, this particle will be thrown away directly. If all particles can't find their corresponding features, the imu data will be used directly. This can be implemented as the calculation module in Section 4.2.

Although the particle generation can run in parallel, it is hard for feature matching to do the same thing which can result in data error. Therefore, once a particle feature matching is ended, the best-fit particle will also be updated.

After the calculation of fitness value and best-fit particle, the best one will be used for the next iteration to increase the correctness of guessing. The next generation will have fewer particles with less rand range. The iteration will repeat 3 times.

After the iteration, the final particle is decided with its features. It will be stored and used in the next step as the format of location, head direction and coordinates of features.

4.1.3 Pose update with the map update

Based on the data from the last step, the new contents of both map and the robot pose will be stored. The robot pose is updated directly by the data received. The map update process will find the feature in the specific map cell and replace the oldest feature information in it.

Therefore, it can also have some robustness for the hijack problem. When the robot is thrown into a new environment, the old feature map will not disturb much of SLAM processing. Although it still needs some time to adapt to the new environment.

4.2 The Back-end SLAM System Implementation in FPGA

In the following part, the description of modules in hardware implementation will be given. The algorithm proposed in Section 4.1 is used for the implementation. Unlike

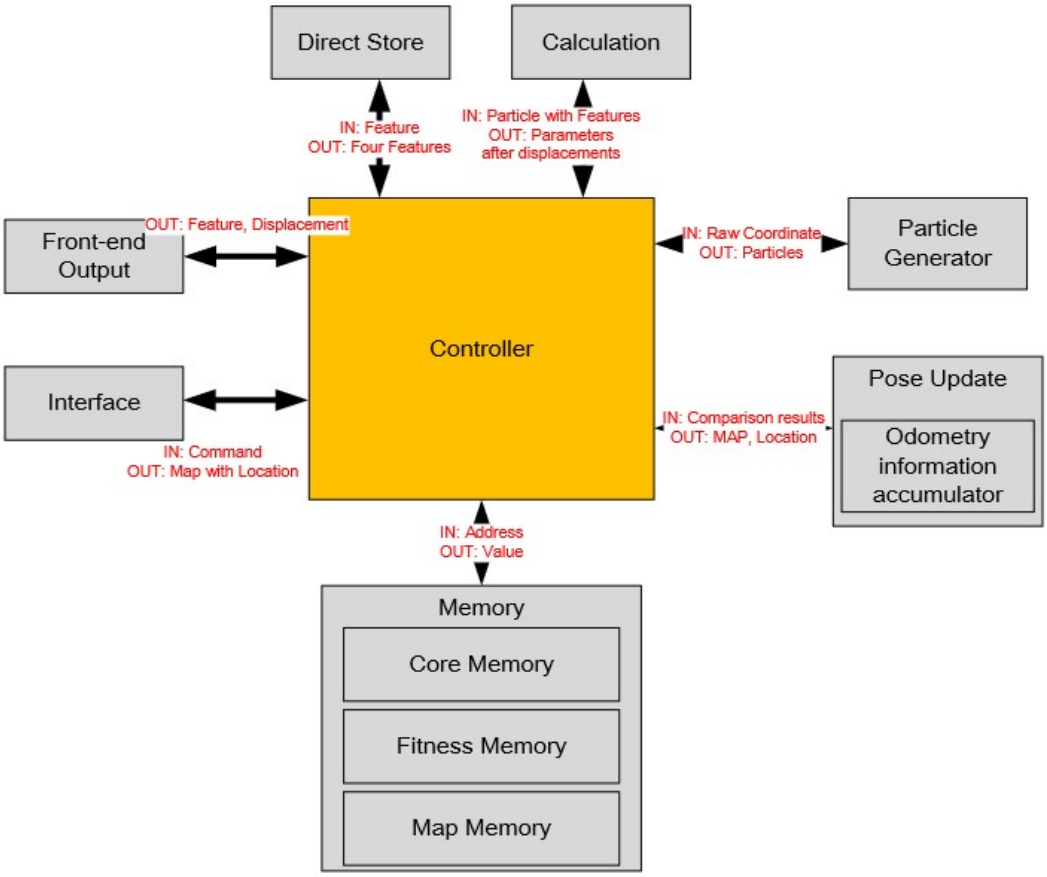

Figure 4.2 The hardware structure of the back-end SLAM system

the SMG-SLAM which uses the grid map for the total computation, this implementation only uses the feature map based on landmarks received from the front-end part. It can therefore dramatically decrease the computation cost of the implementation.

As shown in Figure 4.2, the controller part controls all the memory and the processing logic blocks and makes them work in order without inducing timing issues. It uses the state machine to build. The description of other modules is also given below.

1) Controller

It controls all the modules with the control signal, it will also control the data transmission between different modules. As shown in Figure 4.3. The finite state machine is built based on the algorithm in Section 4.1.

For the control. Below is the description of the total data flow and the finite state machine of the control system in back-end FPGA design.

- Map initial

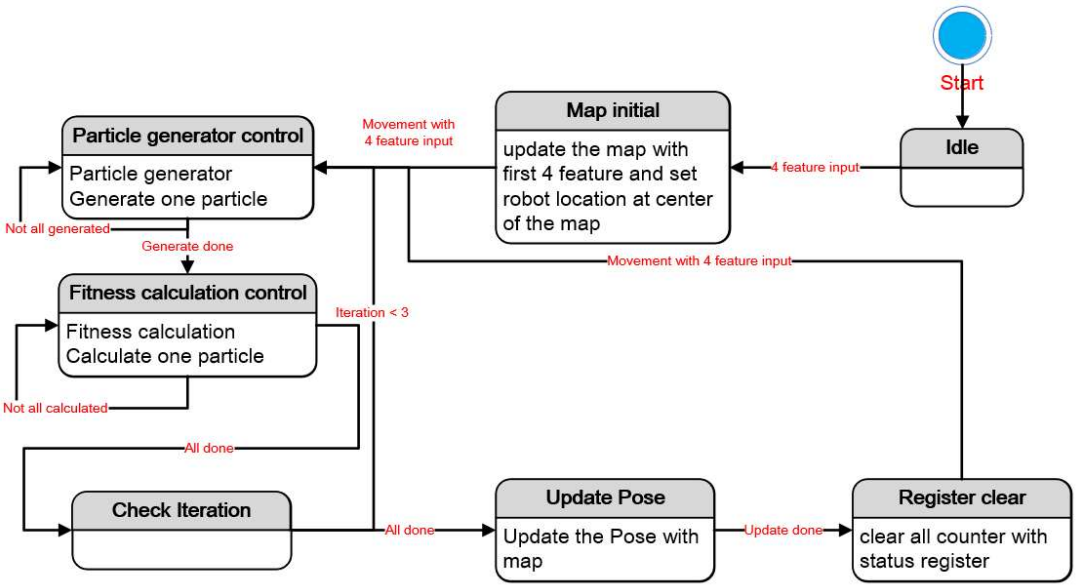

Figure 4.3 The finite state machine of the control module

Because at the beginning of the back-end system, it doesn't have any information about any environment information, it just uses the landmarks extracted from the first scan to store in the landmarks' memory in this step. In this step, 64 particles around the center of the map are established according to the sensor noise. The signal flag from the front-end is read in this state. After the initialization, the control module will never enter this state.

- Particle generator control

When the front-end movement and rotation information enter this step, the back-end system can use the Monte Carlo method to set the particles which can assume the movement of the robots according to the raw movement. Monte Carlo has been mentioned before in the previous part of the paper. In this step, a lot of predictions about the newest pose are made to find the robot's location.

The global coordinates of the four nearest landmarks around the particle will also be calculated after the displacement. In this state, the start signal will be transmitted to the particle generator module. It will also listen to the signal from the fitness calculation module.

- Fitness calculation control

In this state, the fitness calculation will be done by control the fitness calculation module to calculate Manhattan distances and sum them. After that, the fitness result will be checked in check fitness state.

- Check the iteration

In this state, the iteration number is checked. It will decide the next state as shown in

Figure 4.3.

- Read imu data

When the scan landmarks are put into the back-end system, then some imu data will also be put into this part for the preparation of the movement of the particles.

- Update pose and map

If the robot detects the landmarks which exist in previous timesteps, it will update it with the newest observation, or it can just create a new landmark. If the memory map is full, updated contents will also be stored in it.

2) Core memory

Core memory is used to store particle information from the particle generator module. The information contains the coordinates of the particle with 4 features. The location information in the core memory is the information after movement and rotation. The calculate module will use the data in core memory to find corresponding features for the particle in the map memory.

3) Fitness memory

It was used to store the fitness information about the difference between particles' landmarks and the observation landmarks by the sum of Manhattan distances from the calculation module. It will also automatically store the information about the best-fit particle by a small function block when the memory is updated. This function block will also extract the best-fit particle information and its features from core memory by using the index sent by the calculation module. The fitness memory also decides whether the best-fit particle can be used in the update pose module by the fit status. The fit status is also controlled by the calculation module. The best fit data in fitness memory is used in the pose update module.

Although only one particle memory space is needed for further processing, other particle information is also stored in this memory for further optimization to eliminate the possibility to fall into the local optimal particle.

4) Map memory

The map memory stores the landmarks' information. It will also be updated by the pose update module. The allocation of the map memory is shown in Figure 4.4.

In order to increase the speed of the calculation, the memory area of the features map is divided into N^2 areas. N decides the number of cells in the features map. The features can be stored in the cells. The cell's location decides the base address of the features. The base address is composed of x address and y address. Each cell has maximum M features. In the features map, each cell has a memory space for the

pointer. The feature itself also has a related address determined by the appearance sequence of the feature. The parameters M and N decide the fitness calculation. If the M is too big and N is too small, the calculation speed will be slowed down. If the N is too big and N is too small, the features will be hard to find its fitness features.

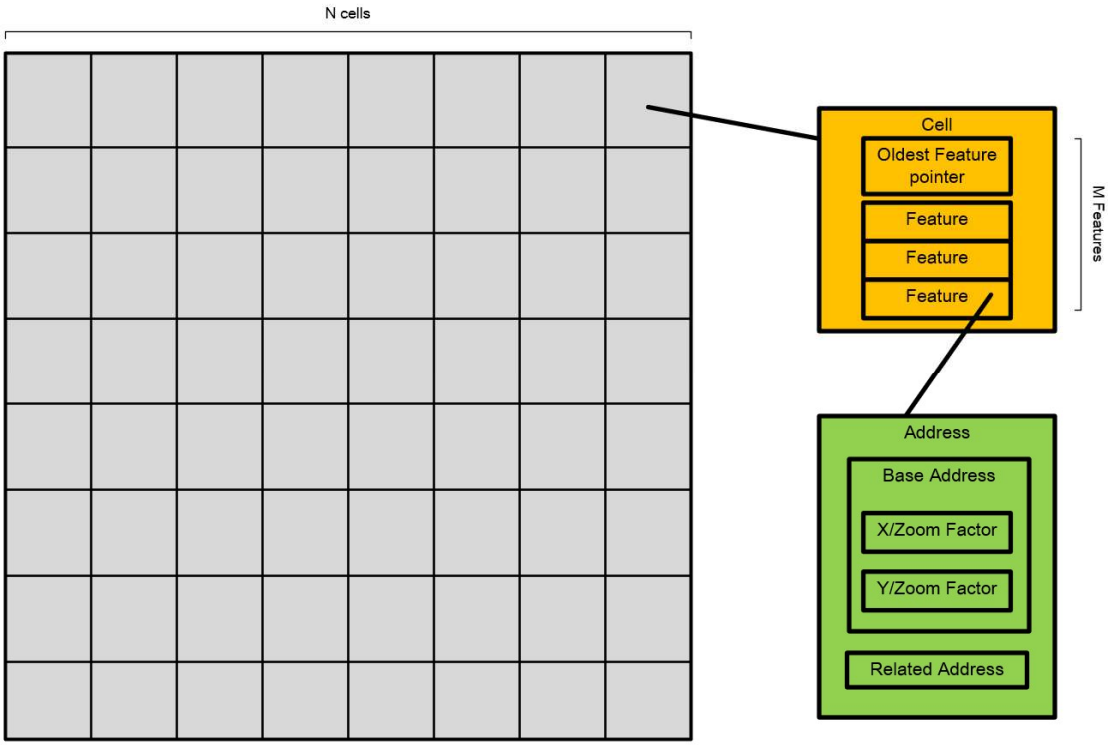

Figure 4.4 The allocation of features map

During the operation of the calculation module, it will read the corresponding cell's features' values of the current particle's feature. Each cell has a memory space to record the oldest feature. When the feature needs to be stored in the map, the designated memory space will be the first update. Then, the pointer's value will plus one. Pointer's value will never bigger than the size of the cell.

5) Calculation

The Calculation part calculates the fitness value of each particle by sum the Manhattan distances of 4 features and transmit it to the fitness memory. If the corresponding cell does not exit any features or the distance between two features is out of bond, the fit status will remain its current status. Otherwise, the smallest distance will be calculated by its Manhattan distance and the fit status will be set to 1. The sum of Manhattan distances will be transmitted into the fitness memory with its particle index.

6) Pose update

In this module, fitness memory.is first checked with the best-fit particle. The top of the memory stores the best-fit particle with fit status.

If there is at least one particle which has at least one landmark which meets the maximum threshold check (fit status = 1). The module will consider this match is successful and it will go to the next state directly to calculate the newest pose of the robot or intelligent agent or enter the next iteration.

If not (fit status = 0), it means that the robot may not detects enough landmarks to match or the robot has been moved too fast to record the old landmarks. In this circumstance, the raw imu data will be used in this module to calculate the newest pose and to add some raw feature points with raw movement and rotation to the whole map.

The odometry information accumulator is a block not only reads the odometry data but also accumulates it to get rough odometry of the robot or agent. When the imu data is used to calculate or it is not needed in this scan, it will automatically reset itself to accumulate the next displacement and rotation.

Next comes the true update, the pose update is combined with the map update, it will use the best-fit particle and its feature to update the map information. The feature map is updated from the oldest feature. The overall state machine of this module is shown in Figure 4.5.

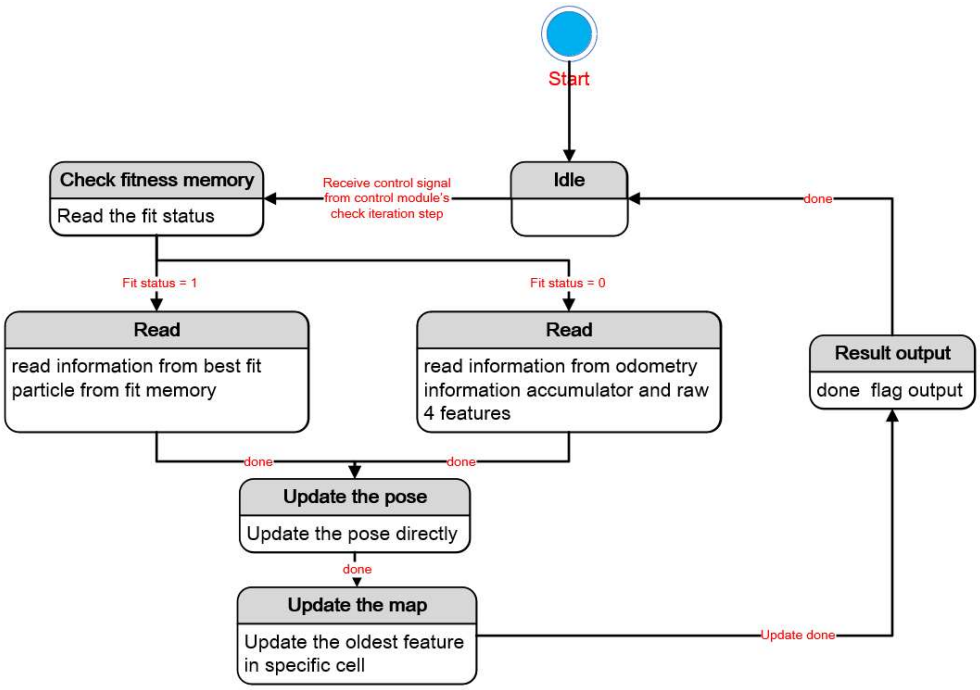

Figure 4.5 The state machine of pose update module

7) Direct store

When the back-end system started, it has no landmarks in the whole map memory, Therefore, the direct store module will put the first scan's landmarks directly into the map memory for further matching and computation.

As shown in Figure 4.6. When the robot just initializes in the new environment, its original coordinate is shown as Equation (4-3).

$$x = y = \frac{N * Zoomfactor}{2} \qquad (4-3)$$

In order to simplify the calculation of coordinates, negative coordinates are not allowed in the design. All coordinates are symbolized as 16-bits integers.

8) Particle generator

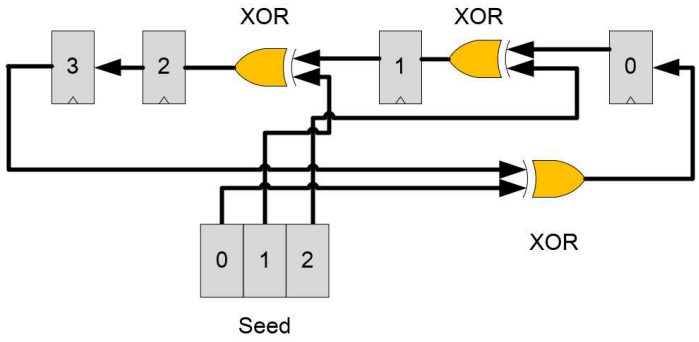

Figure 4.6 The principle of PRNG

This module is used to generate particles according to the noise of the different sensors to guess the true location of the robot. It should be noted that the 4 features with

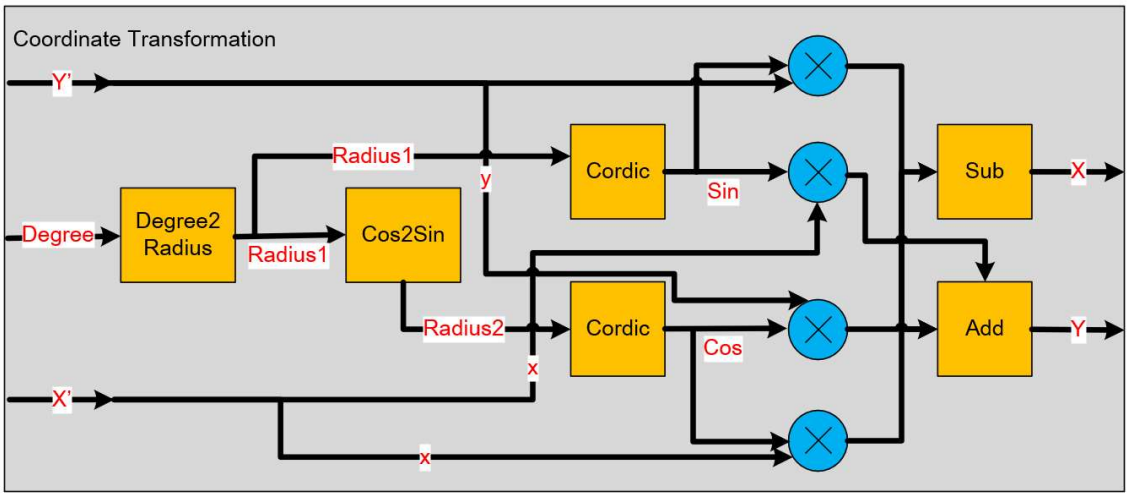

Figure 4.7 The calculation of Coordinate Transformation

the movement and rotation can come from the front-end or fitness memory's best

particle.

Pseudo random number generator (PRNG) is used for the implementation. It is a method which can quickly generate pseudo random numbers without extra physical devices. It is also easy to be implemented in FPGA devices. PRNG is popular in different areas like Monte Carlo-based simulation methods, process generation in video games, and cryptography.

The specific principle of PRNG is shown in Figure 4.6. This Figure is simplified to the calculation of four bits. The seed is used for each calculation of the PRNG. The PRNG will generate a periodic sequence in the end. In order to build the PRNG, the dataflow design is also needed for the implementation. When the PRNG block detects the rising edge of the Trigger, it will start to compute the random numbers. When the random numbers are generated, the Done_flag will be set to 1 and the control part will know that the Particle_output is valid.

The temp value from PRNG is stored in the core memory. After that, the coordinates of the original particle will be changed with the last pose and the rotation and movement result from the front-end. These results are also stored in the core memory. The feature coordinates will be transformed as Figure 4.7. The specific state machine of this module is shown in Figure 4.8. In this module, the design will take the advantage of the FPGA to generate particles' computations in parallel.

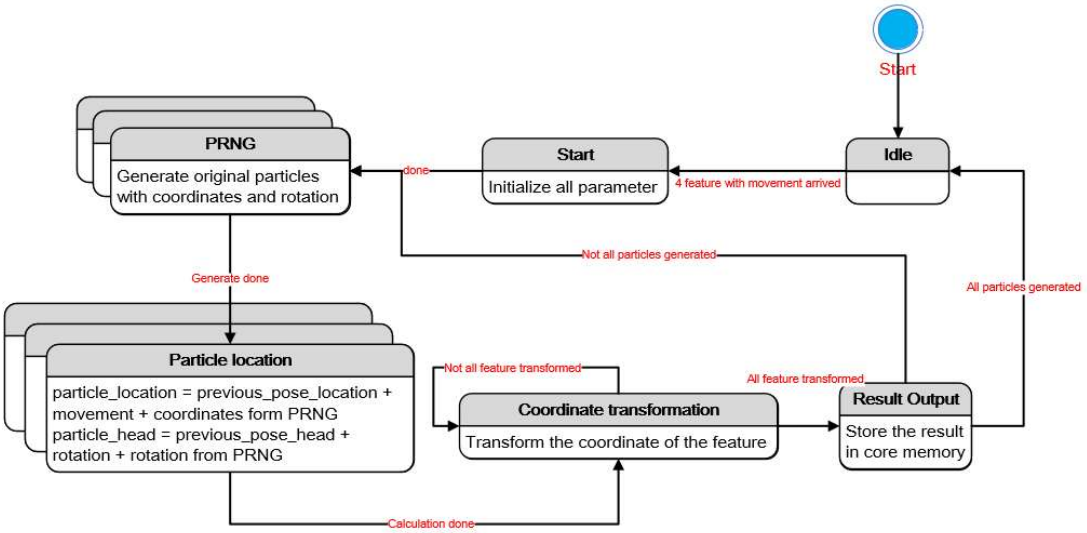

Figure 4.8 The state machine of particle generation module

4.2.1 The combination of the front-end and back-end

AXIS is used to connect the front-end with the back-end. The back-end can't do the visualization in real-time. The final map data and trajectory data are stored in the file generated by Jupyter in PYNQ.

The data will be used for visualization by the QT in C++ on PC.

4.3 The Implementation Results of the Back-end System

After the implementation of the back-end system, the utilization of hardware resources is calculated by the Xilinx Synthesis tools as Table 4.2 below. The specific evaluation of the back-end system will be combined with the front-end part and be shown in specific in Chapter 5. This time, PYNQ Z2 will still be used for implementation.

Table 4.2 The resource utilization of back-end system

Resource	Utilization	Available	Utilization Ratio (%)
LUT-6	9762	53200	18.349
LUTRAM	600	17400	3.448
FF	4623	106400	4.344
DSP	37	220	16.818
IO	12	125	9.600
BUFG	1	32	3.125

4.4 Summary

The design of the back-end of the SLAM system is described in this chapter. The advantages of the GA-based back-end algorithm are described compared with SMG.

Then, the hardware architecture is described in the Chapter. It including the part that controls the whole back-end system, the part that generates the random numbers, the part that refreshes the map, et al. The design of the back-end and the connection between front-end and back-end is also finished.

In Chapter 5, a further comparison will be made between these two methods, the evaluation of the whole SLAM system will also be implemented in the ROS.

5 Experiment and Results

In this chapter, in order to test the efficiency and usability of the feature-based SLAM system by genetic algorithm, implementation of the front-end part in C++ within the ROS is performed. It means that the front-end design can be used in the back-end part.

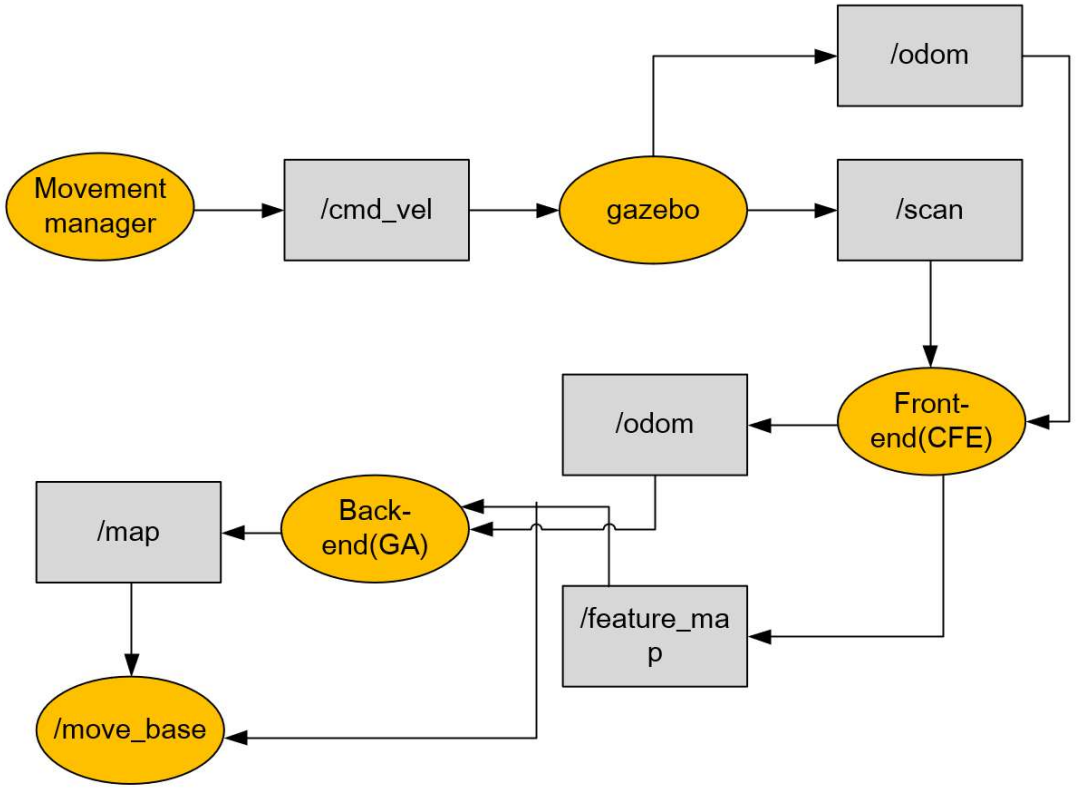

Figure 5.1 Data flow in ROS system

After that, the front-end architecture is implemented on the platform of PYNQ-Z2 to do the comparison. PYNQ-Z2 board is also used to extract map and location information for visualization. The software implementation is also proposed to make the comparison.

Next, the comparison of the resource utilization between the Raspberry Pi and PYNQ-Z2 board is made. The FPGA-based algorithm can increase the concurrency of feature matching by 4-5 times.

In the end, a test of our back-end SLAM system is made to check its possibility and performance, a comparison between the feature-based SLAM algorithm with the FastSLAM algorithm mentioned before is made. In this comparison, the GA-based

methods proposed in this thesis can save 86.25% of power comparing with the software platform with only 0.4% accuracy lost in SLAM.

5.1 The Software Simulation

In this software implementation, the Gazebo platform is used to generate a visual world for robots. Gazebo is a simulation platform for ROS which has been broadly used. In order to simulate the indoor environment at home, the components such as bed, chairs, tables and standing people are added to the world. The specific world is shown in Figure 5.2.

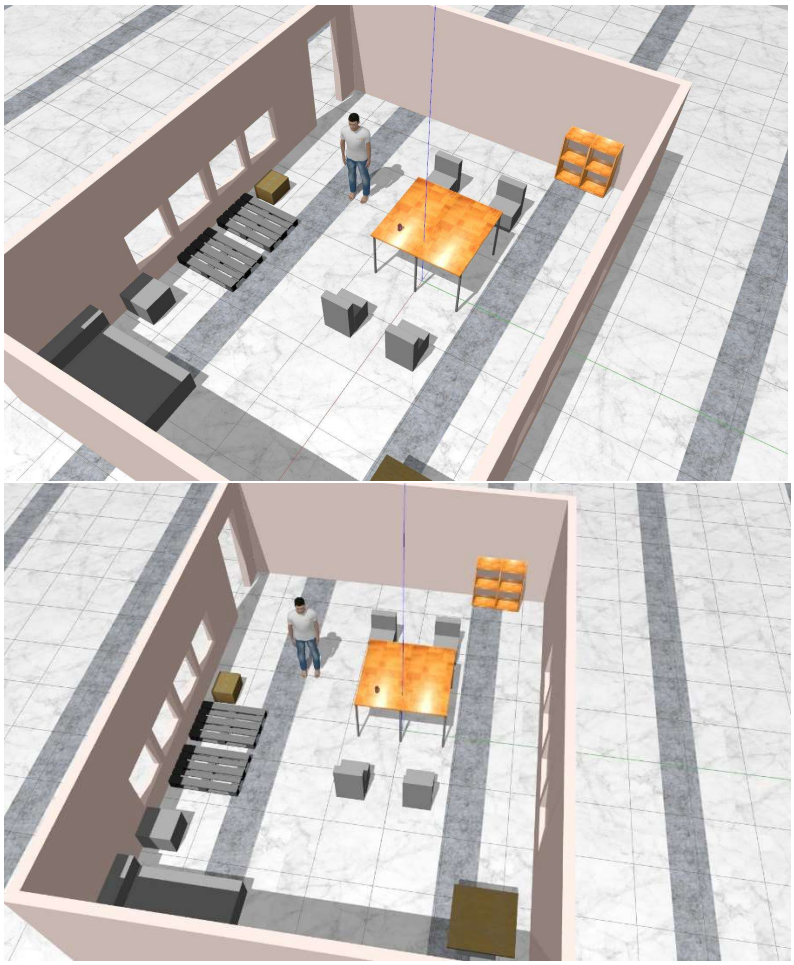

Figure 5.2 The world in Gazebo for simulation

In order to verify the feasibility of the algorithm in this thesis, an implementation based on ROS with Gazebo is also made, the architecture of the software simulation is shown in Figure 5.1.

The Movement manager is a component to control the movement of the robot, it will publish speed messages to a specific topic called cmd_vel. A topic is a data center

for other modules to subscribe to.

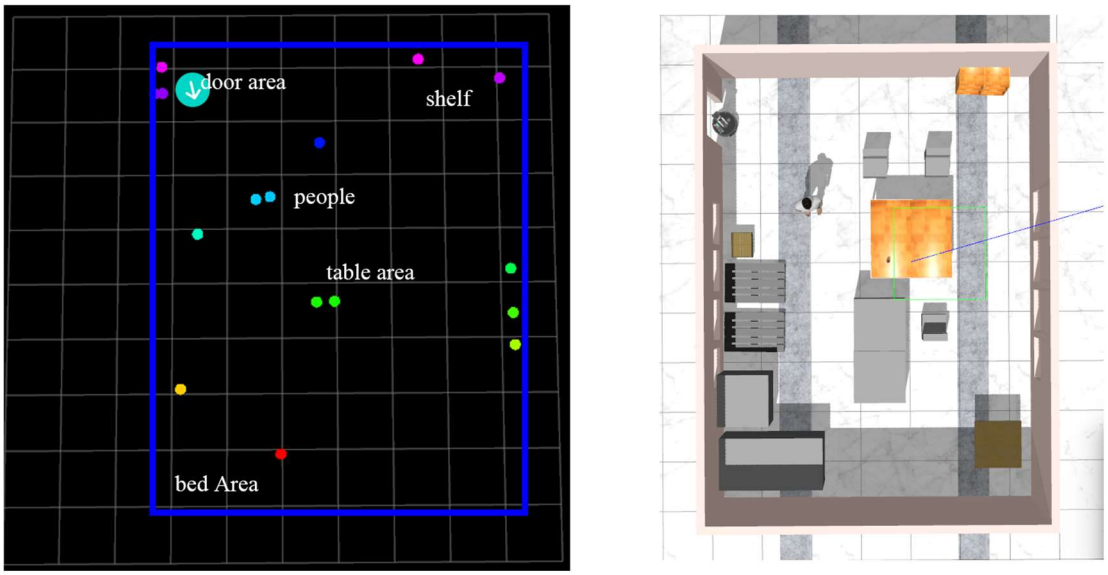

Figure 5.4 The simulation of the robot in the front of the room

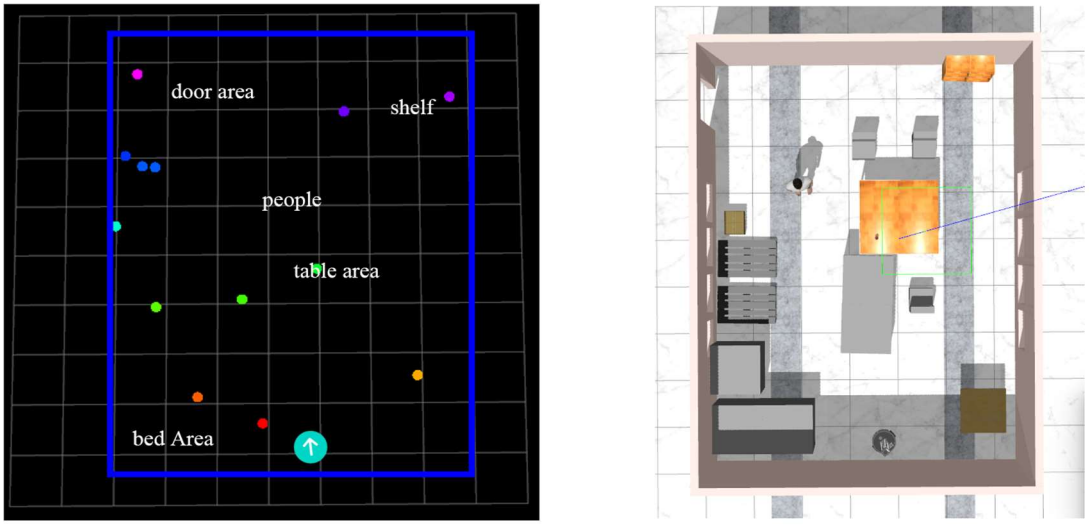

Figure 5.3 The simulation of the robot in the back of the room

Then the Gazebo node will subscribe to the command and move the robot in Gazebo world. Gazebo world itself can also generate some messages with the movement of the robot. The virtual odometer and lidar in the robot will generate the scanlines information with the odometry information by Gazebo. The lidar will send angle and distance information to the front-end. The front-end part will send movement

information with the corner information to the back-end. The back-end part will generate the position information with the map. The calculations are the same as the process which has been mentioned before. The SLAM system will then generate the map messages with the odometry for further usages.

Then, a visualization tool like rviz can subscribe to the information of the robot to generate the map with the location as Figure below. Figure 5.3 and Figure 5.4 show that the different feature points with different colors are recognized by the SLAM system. The different colors symbolize different positions.

The blue square which is added after the rviz visualization symbolizes the room area with the cyan-blue symbolizes the robot in the room. The arrow in the circle symbolizes the orientation of the robot.

5.2 The Implementation of FastSLAM and EKF in Software

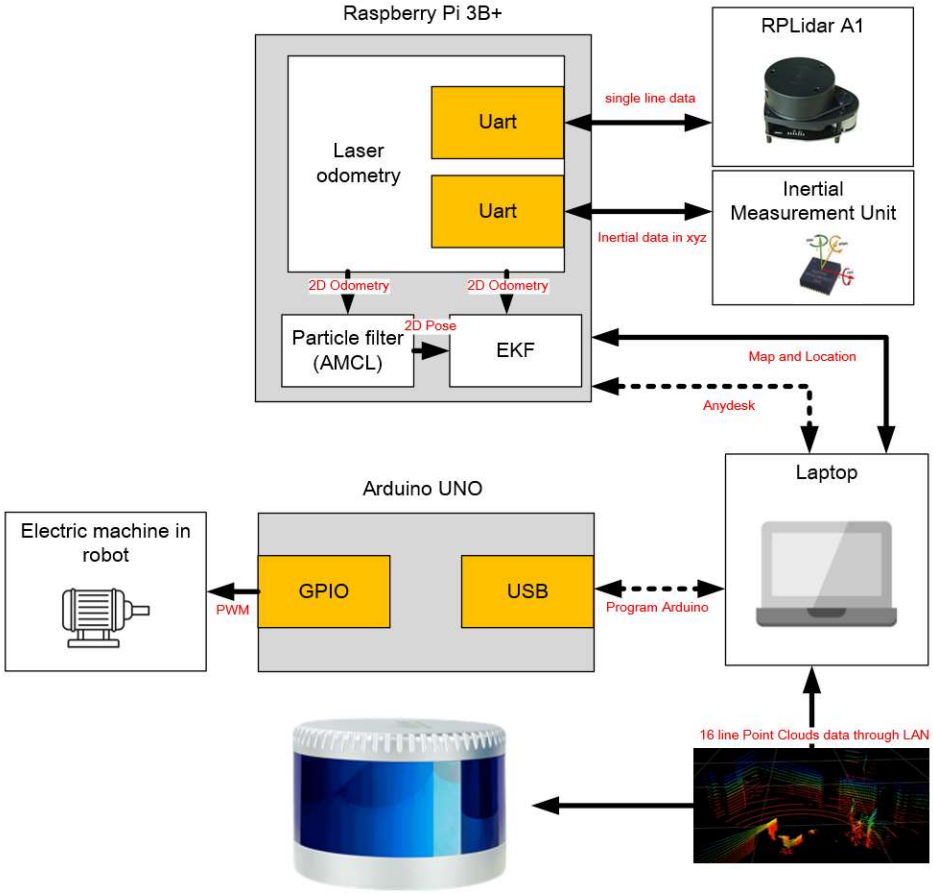

Figure 5.5 The architecture of software implementation

After the implementation of FPGA, software simulation is implemented with the same architecture. In this thesis, the FastSLAM algorithm in C++ with the QT to do the

visualization is realized. Qt[59] is a free and open-source widget toolkit for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android, or embedded systems with little or no change in the underlying codebase while still being a native application with native capabilities and speed.

The implementation of the EKF is also implemented to compare the performance with the design in this thesis. The platform used in this thesis is Raspberry Pi 3B+. It is a cheap platform for embedded systems. In this design, the map and trajectory messages are stored in the Raspberry pi. The Anydesk is used to connect the Raspberry Pi and PC as shown in Figure 5.5. QT is used to show the software performance of the EKFSLAM and visualization. A comparison of these different methods will be made in the Section below. Below is a figure to show the software result of EKF in an environment with the road and trees.

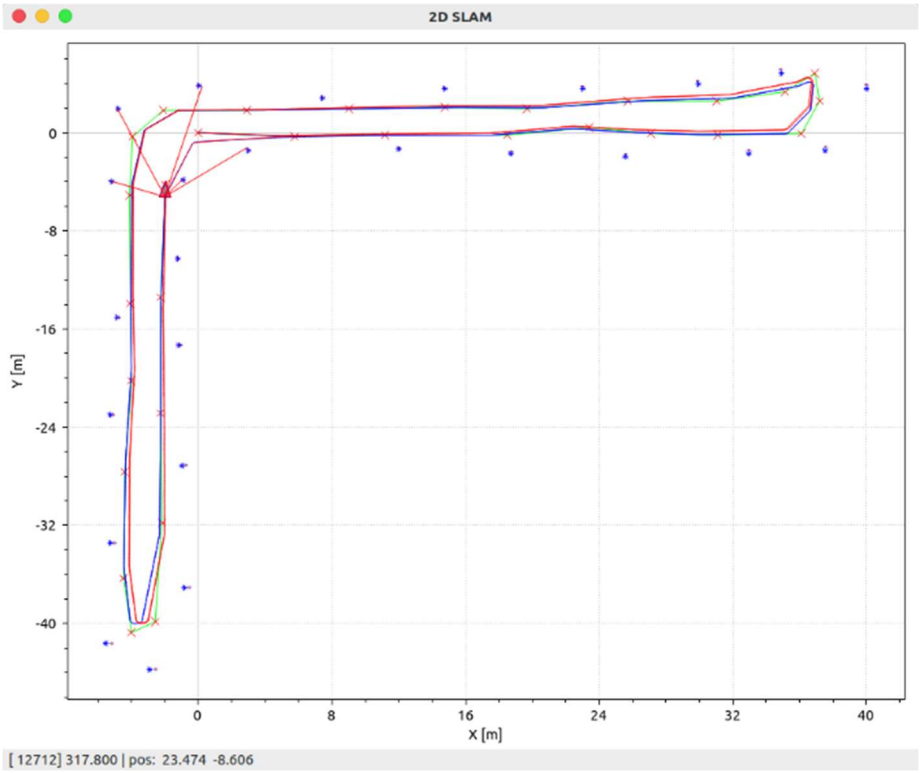

Figure 5.6 The EKF implementation of software

In Figure 5.6, a robot is moving in a road environment which has trees beside the road, the blue points in Figure 5.6 are the landmarks extracted from the environment. The red trajectory represents the ground truth of the robots, it is gained by the GNSS module in the outdoor environment. The blue trajectory is the trajectory calculated by

the EKF. The red trajectory is the route which we want to realize in the beginning. The red crosses are the sudden turning we want to realize. Because the real direction can't be changed directly, the red trajectory is formed automatically by the robot.

The performance could be calculated by Equation (5-1) below. The q_i represents the point on the trajectory estimated by the EKF. p_i represent the point on trajectory of the ground truth at the same time.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(q_i - p_i)^2 \qquad (5-1)$$

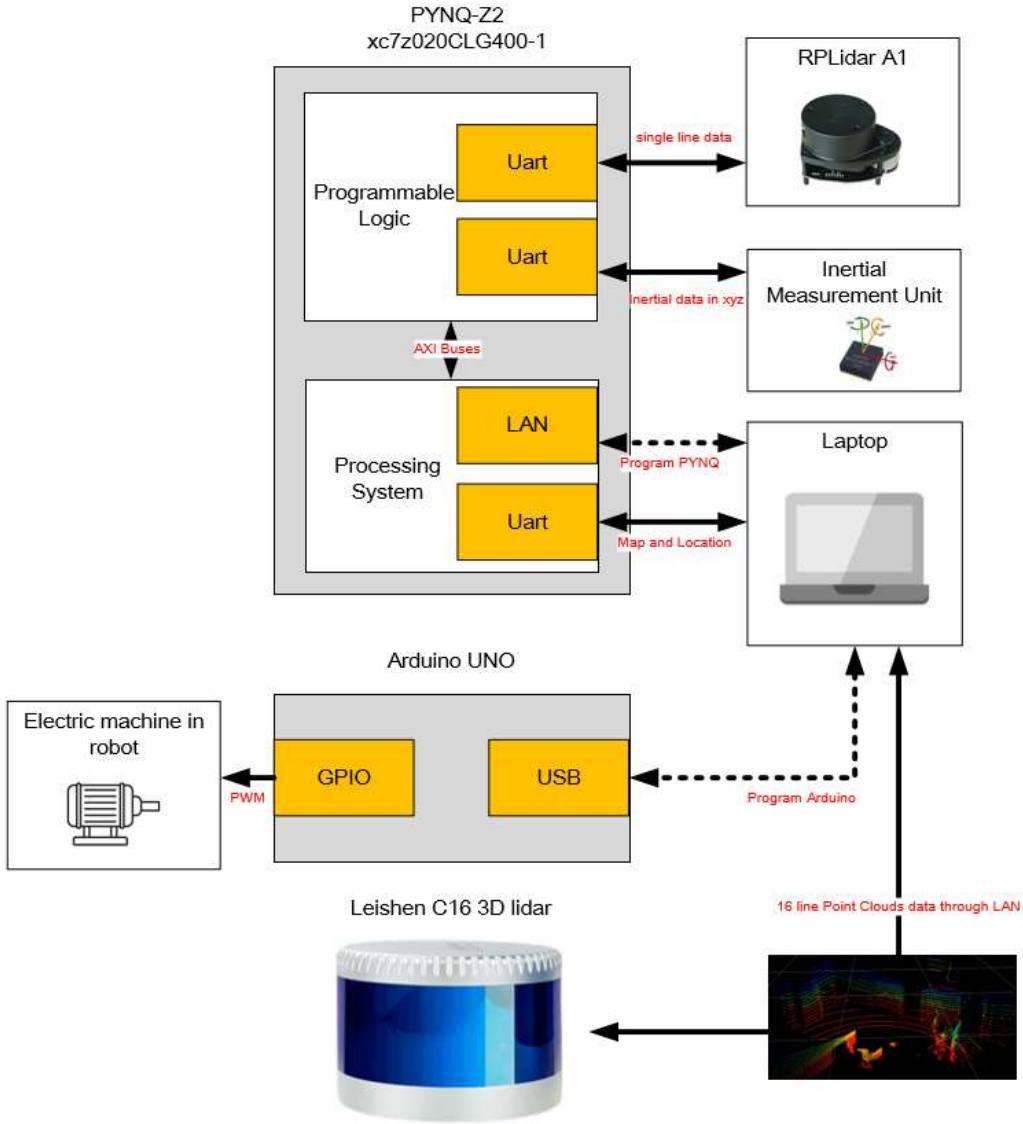

Figure 5.7 The structure of PYNQ

The MSE of the EKF in Figure 5.5 is 0.023m which shows the good performance of EKF-based SLAM in the outdoor environment.

5.3 The Hardware Evaluation

In the FPGA, the front-end part and the back-end part can be connected by the top file. Now comes to how to export the data in the FPGA to visualize. PYNQ platform provides a resolution. In PYNQ, a simple Linux core can be installed on the Cortex A9 core. The structure of the PYNQ is as Figure below. The programmable logic part is the part which can be programmed by the VHDL. The lidar and odometer can be connected to the PYNQ by using UART. The processor system is the part of Cortex A9 which can be controlled by the python program. It can be connected with a PC by using a network cable.

In this core, we can use python to control and read the register in the PL. In the back-end design in this thesis, a part of HLs system is designed with the AXIS protocol to transmit the data. The Cortex A9 core can then send the data to the computer by using UART for faster visualization.

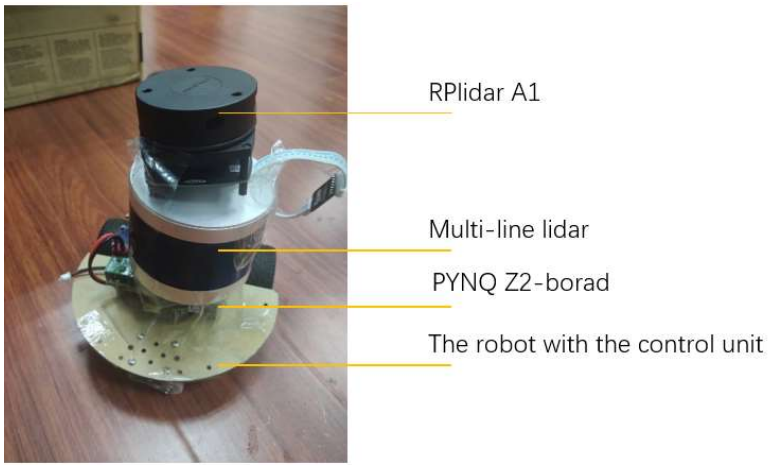

Figure 5.8 The prototype developed and used in this work

5.4 The Experiment and Device

To simulate indoor environment at home. In the lab, different obstacles are set to

increase the number of landmarks in the area and simulate complex environment. The environment is shown in Figure 5.8 in our small car. The small car contains a leishen 16 lines lidar and an RPlidar A1. The 16 lines lidar will be used to get the ground truth of the whole environment by using the Lego-LOAM.

The inexpensive RPLidar A1 is utilized for our experiment. It is a 360 two-dimensional lidar with 1 resolution at 10Hz. A simple lidar platform in a square experiment can be seen in Figure 5.9. In this platform, the lidar is connected directly to a ZYNQ-Z7 board to test its front-end performance. There is also a connection between the FPGA board and the PC by UART.

5.5 The Resource Utilization

In this part, the utilization of the hardware resources in the PYNQ-Z2 board and Raspberry Pi 3B+ are shown.

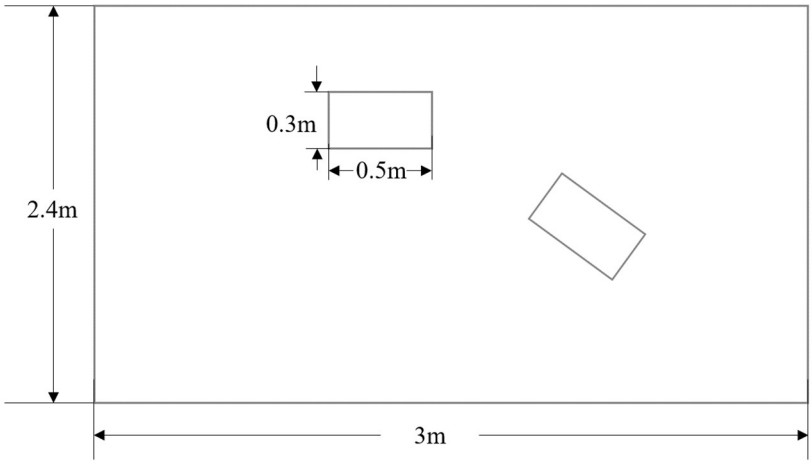

Figure 5.9 The indoor environment

5.5.1 Resource utilization of FPGA design

From table 6.1, resource utilization in PYNQ Z2 xc7z020clg400-1 can be seen. In FPGA, there are certain blocks of RAM that use LUT storage space. Users can also use RAM dynamically formed from LUTs in the logic part which might claim LUT resources for calculating (i.e. use LUT-in-logic). LUTRAM has a large storage space while RAM-by-LUT-in-logic will consume a lot of LUT resources in order to achieve

a large storage space. However, RAM-by-LUT-in-logic is more flexible and convenient to use.

Implementing the feature-extraction algorithm using VHDL in FPGA presents a lot of challenges. On one side, the need for the calculation of trigonometric functions. To solve this, a CORDIC algorithm is implemented in VHDL. In particular, the CORDIC only calculates sine samples. For the cosine calculations which are needed in both back-end and front-end systems, the angle values are transformed. Therefore, the cosine calculations could be done by the sine calculations. On the other side, conversion of the angle's type from the degree to the radius and integrating the CORDIC calculations into a state machine, requiring complimentary intermediate signals. By using CORDIC, Table 5.1 shows that DSP resources are not used much in this design.

Implementing the genetic algorithm-based back end needs more resources with LUTRAM for storing the generated particles and the map. The total IO resource utilization is not the front-end utilization add back-end utilization, because the combined system only has one UART connection with the PC. BUFG is the part for connecting. Therefore, the total utilization is one.

Table 5.1 The resource utilization of front-end back-end combined system

Resource	Front-end Utilization Ratio	Back-end Utilization Ratio	Total Utilization Ratio
LUT-6	24.853%	18.349%	43.203%
LUTRAM	0.574%	3.448%	4.023%
FF	7.505%	4.344%	11.850%
DSP	13.181%	16.818%	30.000%
IO	9.600%	9.600%	9.600%
BUFG	3.125%	3.125%	3.125%

5.6 The Algorithm Test in Lab Experiment

In this part, the test experiment of our back-end design and a comparison between this thesis with the result from EKF is also made. The ground truth was provided by the Lego-LOAM.

The result of our methods is shown in Figure 5.10. The blue points in Figure 5.10 are the landmarks that exist in the environment. It can be seen that this is a square environment with two boxes in it.

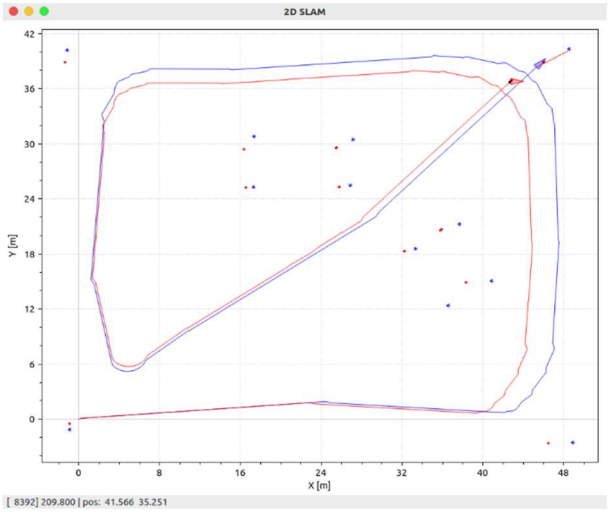

Figure 5.10 The experiment results based-on Genetic Algorithm

The red points symbolize the estimated landmarks stored in the memory. The red trajectory symbolizes the estimated trajectory of the robot. The blue trajectory symbolized the real trajectory of the robot.

The result shows that the method proposed in this thesis can be used for the SLAM system indoor with pretty good accuracy. As Table 5.2 shows, the size of the box in the area calculated by the genetic algorithm in this thesis is 0.157 m^2. The size of the box in the area calculated by the EKF is 0.149 m^2. The real size of the box is 0.15 m^2. The Genetic Algorithm based methods we proposed in this thesis have increased the concurrency 4-5 times which could speed up the computation process of SLAM with 0.4% accuracy lost comparing with the EKF.

Table 5.2 The accuracy comparison between two methods

	parameter	accuracy
EKF	0.149	94.9%
GA	0.157	95.2%

The power consumption has also been calculated in this chapter. The power consumption of EKF-based algorithm is 12W on the Raspberry Pi 3B+. The power consumption in this thesis is only 1.65W which is calculated by Vivado. The GA-based methods proposed in this thesis save 86.25% of power consumption in the SLAM system.

In Table 5.4, it is shown that the FPGA-based SLAM system can also be faster than the SLAM system based on software. The parameter in this Table symbolizes the condition of the memory of the feature map.

Table 5.3 Speed results comparison

parameter	M = 16, N = 16, Zoom factor = 16
Actual time for one iteration (GA)	4.55ms
Actual time for one iteration (EKF)	24.2ms
Achieved speed up	5.31 times

Table 5.4 The power consumption comparison between two methods

	Power consumption
EKF	12w
GA	1.65w

Although it remains some space to improve compared with the EKF based SLAM system, the SLAM system proposed in this thesis has lower-power consumption and high-speed comparison comparing with it.

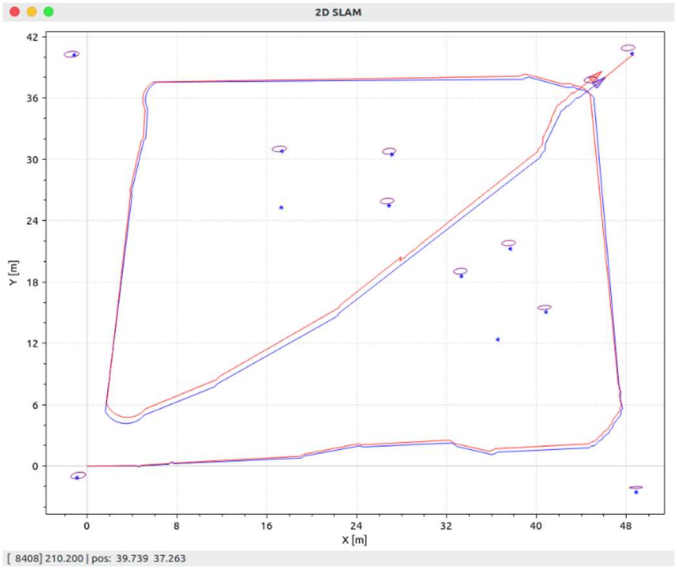

Figure 5.11 The experiment results based-on EKF

5.6.1 Resource utilization among two different platforms

Table 5.5 Performance comparison between two methods

	Xilinx Zynq XC7Z010 (VHDL Impl.)	Raspberry Pi 3B+ (C++ Impl.)
Approx. Resource Max.	24%(fixed) + 2%	23% (fixed) + 47% CPU
Max. concurrency	50-60	< 5 -15

The performance between the CPU and the design on the ZYNQ board is compared.

The result is listed below.

For future work, more of our design LUT will be shared to minimize hardware utilization. As shown in Table 5.5, The MAX. concurrency means that the amount of lidar odometry calculations can be run in parallel. It can be already seen that the FPGA design can have high efficiency on parallel computation.

5.7 Summary

In this chapter. The feasibility evaluation of the SLAM algorithm is made in the ROS. It shows that the SLAM system in this thesis can extract features from the domestic environment in Gazebo and realize the localization tasks.

The prototype of our system is second made to evaluate the performance in the lab environment. The system is based on Xilinx PYNQ Z2. The multi-line lidar is also implemented in this prototype to generate the ground truth of the prototype.

In order to compare the performance of the SLAM system in the thesis, the EKF is also implemented on the Raspberry Pi 3B+ platform. Compared with the system on the software, the system in this thesis can save 86.25% of power consumption with only 0.4% of accuracy loss. The hybrid SLAM system only takes 4.55ms for location calculation in each scan which is 5.31 times faster compared with the software implementation with EKF.

6 Conclusion and Future Work

Low-cost, stable and fast SLAM systems are expected to be of great demand and broadly used in the future. The motivation for this study was to design, develop and implement a hybrid SW-HW methodology for a robust SLAM but also with low power consumption to make the whole system have a long operating time.

In this thesis, replacing some feature-extraction work on the CPU with FPGA with low power consumption and high speed is confirmed to be possible. This results in SW-HW co-design at the system level shows that FPGA can handle multiple tasks in parallel to accelerate the multiple homogeneous tasks and CPU can have more computation resources to do some complex work like state changes and visualization. High-parallelism and low-latency SLAM systems meeting the low-cost and high energy efficiency requirements set as objectives of the thesis.

It has also been confirmed that the SLAM system with the CFE method and the genetic algorithm can operate in ROS with Gazebo for visualization. With the ROS implementation, it is easier to implement the system in FPGA.

Finally, it is confirmed that the Genetic Algorithm Based SLAM system in FPGA can be used in the indoor environment compared with the FastSLAM algorithm with a slight trajectory disturbance. The accuracy lost is 0.4% comparing with the software implementation. But the design on FPGA can save 85.25% of the computation power consumption. The hybrid SLAM system only takes 4.55ms for location calculation in each scan which is 5.31 times faster compared with the software prototype with EKF.

The main challenge in this work was to design the complete system based on FPGA. Both the back-end and the front-end systems need to control the memory to read and write in the right order, it can only be carefully realized by state machine in FPGA.

In the future, a grid-based topology map will be added to our implementation. Grid-based topology maps now can be used on global navigation with the free space and the specific target. The navigation tasks can be handled by combining the global navigation method with the local navigation method. To handle local navigation tasks, a collision-avoidance mechanism will also be built in the future.

The hijack problem in a mobile robot will also be considered in the future. If someone takes up the robot and throws it into a new environment, the robot must recognize the brand-new environment and make new navigation tasks based on the new environment.

References

[1] Department of International Cooperation Ministry of Science and Technology and P.R.China, “Next Generation Artificial Intelligence Development Plan,” 2017.

[2] HIT ROBOT Group and China Institute of Science and Technology Evaluation, “Research Report on the Service Robot Industry in China 2019,” 2019.

[3] A. Singandhupe and H. M. La, “A Review of SLAM Techniques and Security in Autonomous Driving,” in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 602–607. doi: 10.1109/IRC.2019.00122.

[4] C. Cadena *et al.*, “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016, doi: 10.1109/TRO.2016.2624754.

[5] M. Zaffar, S. Ehsan, R. Stolkin, and K. M. D. Maier, “Sensors, SLAM and Long-term Autonomy: A Review,” *2018 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2018*, pp. 285–290, 2018, doi: 10.1109/AHS.2018.8541483.

[6] D. V Nam and K. Gon-Woo, “Solid-State LiDAR based-SLAM: A Concise Review and Application,” in *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2021, pp. 302–305. doi: 10.1109/BigComp51126.2021.00064.

[7] A. Li, X. Ruan, J. Huang, X. Zhu, and F. Wang, “Review of vision-based Simultaneous Localization and Mapping,” in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2019, pp. 117–123. doi: 10.1109/ITNEC.2019.8729285.

[8] A. Li, X. Ruan, J. Huang, X. Zhu, and F. Wang, “Review of vision-based Simultaneous Localization and Mapping,” in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2019, no. Itnec, pp. 117–123. doi: 10.1109/ITNEC.2019.8729285.

[9] B. Gao, H. Lang, and J. Ren, “Stereo Visual SLAM for Autonomous

Vehicles: A Review," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020, vol. 2020-Octob, pp. 1316–1322. doi: 10.1109/SMC42975.2020.9283161.

[10] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015, doi: 10.1109/TRO.2015.2463671.

[11] Q. Huang and Y. Zhang, "Space Target Association Based on Epipolar Geometry Constraints under Multi-Star Sensors," in *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*, 2019, vol. 2, pp. 1550–1554. doi: 10.1109/EITCE47263.2019.9094972.

[12] R. I. Hartley and P. Sturm, "Triangulation," *Computer Vision and Image Understanding*, vol. 68, no. 2, pp. 146–157, 1997, doi: 10.1006/cviu.1997.0547.

[13] S. Li, C. Xu, and M. Xie, "A robust O(n) solution to the perspective-n-point problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1444–1450, 2012, doi: 10.1109/TPAMI.2012.41.

[14] T. Whelan *et al.*, "Computer Science and Artificial Intelligence Laboratory Technical Report Kintinuous : Spatially Extended KinectFusion Kintinuous : Spatially Extended KinectFusion," 2012.

[15] H. Kim, S. Leutenegger, and A. J. Davison, "Real-Time 3D Reconstruction and 6-DoF Tracking with an Event Camera," in *Computer Vision -- ECCV 2016*, 2016, pp. 349–364.

[16] H. T. and T. W. J. Pe˜na Queralta1, F. Yuhong, L. Salomaa, L. Qingqing, T. N. Gia, Z. Zou, "FPGA-based Architecture for a Low-Cost 3D Lidar Design and Implementation from Multiple Rotating 2D Lidars with ROS," pp. 2–5, 2019.

[17] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4758–4765, 2018, doi: 10.1109/IROS.2018.8594299.

[18] M. Zaffar, S. Ehsan, R. Stolkin, and K. M. D. Maier, "Sensors, SLAM and Long-term Autonomy: A Review," *2018 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2018*, pp. 285–290, 2018, doi:

10.1109/AHS.2018.8541483.

[19] Y. Song, M. Guan, W. P. Tay, C. L. Law, and C. Wen, “UWB/LiDAR fusion for cooperative range-only SLAM,” *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2019-May, pp. 6568–6574, 2019, doi: 10.1109/ICRA.2019.8794222.

[20] R. Kreiser, A. Renner, Y. Sandamirskaya, and P. Pienroj, “Pose Estimation and Map Formation with Spiking Neural Networks: Towards Neuromorphic SLAM,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 2159–2166, 2018, doi: 10.1109/IROS.2018.8594228.

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

[22] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear/nongaussian bayesian tracking,” *Bayesian Bounds for Parameter Estimation and Nonlinear Filtering/Tracking*, vol. 50, no. 2, pp. 723–737, 2007, doi: 10.1109/9780470544198.ch73.

[23] N. V Dinh and G. Kim, “Multi-sensor Fusion Towards VINS: A Concise Tutorial, Survey, Framework and Challenges,” in *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2020, pp. 459–462. doi: 10.1109/BigComp48618.2020.00-26.

[24] T. Qin, P. Li, and S. Shen, “VINS-Mono : A Robust and Versatile Monocular Visual-Inertial State Estimator,” vol. 34, no. 4, pp. 1004–1020, 2018.

[25] G. Huang, “Visual-Inertial Navigation : A Concise Review,” *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9572–9582, 2019, doi: 10.1109/ICRA.2019.8793604.

[26] N. Van DInh and G. W. Kim, “Multi-sensor fusion towards vins: A concise tutorial, survey, framework and challenges,” *Proceedings - 2020 IEEE International Conference on Big Data and Smart Computing, BigComp 2020*, pp. 459–462, 2020, doi: 10.1109/BigComp48618.2020.00-26.

[27] T. Qin, P. Li, and S. Shen, “VINS-Mono : A Robust and Versatile Monocular Visual-Inertial State Estimator,” vol. 34, no. 4, pp. 1004–1020, 2018.

[28] G. Shafer, “Dempster-shafer theory,” *Encyclopedia of artificial intelligence*, vol. 1, pp. 330–331, 1992, doi: 10.4018/978-1-59904-849-9.ch068.

[29] P. Biber and W. Straßer, “The Normal Distributions Transform: A New Approach to Laser Scan Matching,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, 2003, vol. 3, no. October, pp. 2743–2748.

[30] L. Qingqing, J. Pena Queralta, T. Nguyen Gia, Z. Zou, and T. Westerlund, “Multi Sensor Fusion for Navigation and Mapping in Autonomous Vehicles: Accurate Localization in Urban Environments,” *Unmanned Systems*, 2020, doi: 10.1142/s2301385020500168.

[31] R. Kreiser, A. Renner, Y. Sandamirskaya, and P. Pienroj, “Pose Estimation and Map Formation with Spiking Neural Networks: Towards Neuromorphic SLAM,” *IEEE International Conference on Intelligent Robots and Systems*, pp. 2159–2166, 2018, doi: 10.1109/IROS.2018.8594228.

[32] D. Rodriguez-Losada, P. San Segundo, F. Matia, R. Galan, A. Jiménez, and L. Pedraza, “FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem,” *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 6, no. PART 1, pp. 542–547, 2007, doi: 10.3182/20070903-3-fr-2921.00092.

[33] B. L. E. A. Balasuriya *et al.*, “Outdoor robot navigation using Gmapping based SLAM algorithm,” *2nd International Moratuwa Engineering Research Conference, MERCon 2016*, pp. 403–408, 2016, doi: 10.1109/MERCon.2016.7480175.

[34] B. Triggs, P. McLauchlan, and R. Hartley, *Vision Alogirithms: Theory and Practice*, vol. 7246. 2012. doi: 10.1007/978-3-642-29066-4{_}11.

[35] R. Goebel, *RoboCup 2011: Robot World Cup XV*. 2011. doi: 10.1007/978-3-642-34182-3.

[36] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, “Efficient sparse pose adjustment for 2D mapping,” *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 22–29, 2010, doi: 10.1109/IROS.2010.5649043.

[37] W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2D

LIDAR SLAM," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1271–1278, 2016, doi: 10.1109/ICRA.2016.7487258.

[38] J. Zhang and S. Singh, "Low-drift and real-time lidar odometry and mapping," *Autonomous Robots*, vol. 41, no. 2, pp. 401–416, 2017, doi: 10.1007/s10514-016-9548-2.

[39] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3951 LNCS, pp. 430–443, 2006, doi: 10.1007/11744023_34.

[40] W. E. Holzinger, H. Löcker, and B. Löcker, "SURF: Speeded Up Robust Features," *Bulletin of Insectology*, vol. 61, no. 1, pp. 121–122, 2008.

[41] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004, doi: 10.1023/B:VISI.0000029664.99615.94.

[42] C. J. H. Chien, C. C. Hsu, W. Y. Wang, W. C. Kao, and C. J. H. Chien, "FPGA-Implemented Corner Feature Extracting Simultaneous Localization and Mapping," *IEEE International Conference on Consumer Electronics - Berlin, ICCE-Berlin*, vol. 2016-Octob, pp. 98–99, 2016, doi: 10.1109/ICCE-Berlin.2016.7684729.

[43] C. J. H. Chien, C. J. H. Chien, and C. C. Hsu, "Hardware-Software Co-Design of an Image Feature Extraction and Matching Algorithm," *Proceedings - 2019 2nd International Conference on Intelligent Autonomous Systems, ICoIAS 2019*, pp. 37–41, 2019, doi: 10.1109/ICoIAS.2019.00013.

[44] P. J. Zeno, "Using an FPGA to emulate grid cell spatial cognition in a mobile robot," *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics, ICDL-EpiRob 2016*, pp. 7–8, 2017, doi: 10.1109/DEVLRN.2016.7846778.

[45] Z. Xu, J. Yu, C. Yu, H. Shen, Y. Wang, and H. Yang, "CNN-based Feature-point Extraction for Real-time Visual SLAM on Embedded FPGA," *Proceedings - 28th IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2020*, pp. 33–37, 2020, doi: 10.1109/FCCM48280.2020.00014.

[46] G. Mingas, E. Tsardoulias, and L. Petrou, “An FPGA implementation of the SMG-SLAM algorithm,” *Microprocessors and Microsystems*, vol. 36, no. 3, pp. 190–204, 2012, doi: 10.1016/j.micpro.2011.12.002.

[47] H. Yoshida, H. Fujimoto, D. Kawano, Y. Goto, M. Tsuchimoto, and K. Sato, “ROS: an open-source Robot Operating System,” *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, pp. 4754–4759, 2015, doi: 10.1109/IECON.2015.7392843.

[48] C. Cadena *et al.*, “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age,” *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016, doi: 10.1109/TRO.2016.2624754.

[49] J. Khalife, S. Ragothaman, and Z. M. Kassas, “Pose estimation with lidar odometry and cellular pseudoranges,” *IEEE Intelligent Vehicles Symposium, Proceedings*, no. Iv, pp. 1722–1727, 2017, doi: 10.1109/IVS.2017.7995956.

[50] S. Thrun, “Probabilistic robotics,” *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002, doi: 10.1145/504729.504754.

[51] G. others Welch, Greg Bishop, G. Welch, G. Bishop, and others, “An introduction to the Kalman filter,” *IEEE Antennas and Wireless Propagation Letters*, vol. 17, no. 5, pp. 833–836, 1995, doi: 10.1109/LAWP.2018.2818058.

[52] E. Hoshiya, Masaru Saito, M. Hoshiya, and E. Saito, “Structural identification by extended Kalman filter,” *Journal of engineering mechanics*, vol. 110, no. 12, pp. 1757–1770, 1984.

[53] E. A. Wan and R. Van Der Merwe, “The unscented Kalman filter for nonlinear estimation,” *IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium, AS-SPCC 2000*, pp. 153–158, 2000, doi: 10.1109/ASSPCC.2000.882463.

[54] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, “A tutorial on particle filters for online nonlinear/nongaussian bayesian tracking,” *Bayesian Bounds for Parameter Estimation and Nonlinear Filtering/Tracking*, vol. 50, no. 2, pp. 723–737, 2007, doi: 10.1109/9780470544198.ch73.

[55] S. Thrun and A. Bücken, “Learning Maps for Indoor Mobile Robot Navigation.,” 1996.

[56] H. Tang, R. Yan, and K. C. Tan, “Cognitive Navigation by Neuro-Inspired Localization, Mapping, and Episodic Memory,” *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 3, pp. 751–761, 2018, doi: 10.1109/TCDS.2017.2776965.

[57] K. Baker, “Singular value decomposition tutorial,” *The Ohio State University*, vol. 24, 2005.

[58] SLAMTEC, “RPLIDAR Low Cost 360 Degree Laser Range Scanner Interface Protocol and Application Notes,” 2018. http://bucket.download.slamtec.com/b42b54878a603e13c76a0a0500b53595846614c6/LR001_SLAMTEC_rplidar_protocol_v1.1_en.pdf

[59] The Qt Company., “Qt 6.0.2 Released,” 2021. https://www.qt.io/blog/qt-6.0.2-released