Multi-robot SLAM and Map Merging

by

Hongye Li

A Thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 10, 2015

Keywords: Map merging, SLAM, Collaborative robotics, Image matching

Abstract

Although it is a very useful technique for a robot to detect an unknown environment and plot the map, mapping is a classic problem in robotics. The mapping task can be sped up by multi-robot cooperation. Each robot in a team detects a part of the environment and finally merges each individual robot’s map into a single whole one. Because different robots may use different coordinate systems, the maps built by robots cannot merge together easily. The key problem of map merging is to coordinate unification. This thesis presents a way to solve this problem by comparing robot visions. The robots are at the same place when they have the same visions. The robots’ coordinates can be unified when their visions are matched and then their individual maps can be merged together.

Acknowledgments

First, I want to thank my advisor, Dr. Thaddeus A. Roppel. After learning much in his class, I decided to do more research on robotics. He offered me many great ideas and suggestions, and I am really lucky to have an advisor like him. I also want to thank Dr. John Hung. He is a great teacher in the control systems area, and I learned a lot in his control classes. Finally, I would like to thank Dr. Robert Dean for agreeing to be a member of my thesis committee.

Additionally, I would like to offer sincere thanks to my parents. They gave me everything and kept supporting me without any complaints. Special thanks to my girlfriend, as well. She is always sweet and patient, and she comforted me when I was discouraged or felt hopeless.

Finally, I would like to thank all of my friends, especially my friends at Auburn University: Jian Bao, Zhan Xu, Tao Ye, and Yan Li. Thank you for providing me so much help.

Table of Contents

List of Figures

Chapter 1 Introduction

The progress of a society is always accompanied by improvements in productivity. Humans have tried to invent tools to improve productivity since ancient times. A robot is just a more advanced tool. After William Walter invented the first electronic robots in 1949, the robot started changing human life. Robots can do many jobs that humans cannot do or find hard to do. For example, industrial robots are widely used to perfectly finish heavy, repetitive, monotonous jobs.

Currently, many researchers are focusing on mobile robots. Industrial robots are usually fixed so they are restricted in a relative small space. Mobile robots can move by themselves so they can do some jobs in a large space. For example, the robot vacuum cleaner is very popular with consumers. This robot can move around a room and clean the floor automatically without any human help. Another useful task for mobile robots is mapping. Because some places are difficult or dangerous for humans to step into, such as dark caves or disaster ruins, we hope that a robot can independently detect and map an unknown environment in place of a human.

Usually, a task can be finished faster when people cooperate. This also applies to robots. In the science fiction film, *Prometheus*, a search team explored an unknown, dangerous underground place by setting up several flying robots. These robots flew into different tunnels, built different maps, and finally merged the individual maps into a single one. My goal in this thesis is to try to find such a way to merge the maps.

Before merging multiple maps, we first need each robot to individually and automatically detect and map an unknown environment. This is the simultaneous localization and mapping (SLAM) problem. The goal of SLAM can be described as follows: Starting from an arbitrary initial point, a mobile robot should be able to autonomously explore an environment with its onboard sensors, gain knowledge about it, interpret the scene, build an appropriate map, and localize itself relative to this map [1].

After multiple robots' SLAMs, each robot will produce an individual map. The next step is map merging. Because different maps may use different coordinate systems. The key problem of map merging is to find out the map transformation matrix between individual maps [2]. In this thesis, the map merging was based on a prerequisite that different individual maps have common regions. I used computer vision to find the common regions. When the robots are at the common region, then they have the same visions. By using image matching technique to compare the robots' visions, the common region can be determined when the robots' visions are matched. Then the map transformation matrix can be calculated from different robots' visions.

The remainder of the thesis is organized as follows: Chapter 2 is an overview of the relevant literature; Chapter 3 introduces the simulation tool USARSim; Chapter 4 describes the SLAM problem; Chapter 5 relates to image processing, including an image matching algorithm and basic camera calibration; Chapter 6 shows the experiment and the results; and Chapter 7 discusses conclusions and future work.

Chapter2 Literature Review

2.1 Autonomous mobile robots

An autonomous mobile robot (AMR) has some advantages compared to fixed robots or remote-controlled robots. First, an AMR can move into different places. This is very useful for commercial robots, such as robot vacuum cleaners and robot lawn mowers. An AMR has the ability to finish a task without a human's control. In other words, the robot should make and execute decisions by itself. Although the decisions made by an AMR cannot be as rational or comprehensive as human decisions, these decisions are still not easy to achieve. Robots understand the world only by sensors. According to data collected by the sensors, the robot's processor makes a decision. In reality, robots may encounter many different situations in which they need to do their jobs properly. For example, a robot mower's users definitely do not want it cutting their pet's leg. Using the sensors' data to make a correct decision is a big challenge for the robot. Thus, an AMR should have the following necessary abilities, as summarized in [3]:

1. Ability to obtain information on the operation environment.
2. Ability to work for an extended period of time without human intervention.
3. Ability to move itself in the environment without human assistance.
4. Ability to avoid situations that are harmful to people, property, or itself, unless

those are part of design specifications.

There are four basic problems for an AMR to solve: perception, localization, navigation, and path planning. In recent decades, many researchers have focused on solving these problems.

2.2 SLAM

SLAM is a classic robotics problem of constructing and updating a map of an unknown place while simultaneously keeping track of a location within the map. The aim of SLAM is to recover both a robot's position and a map using only the data gathered by the robot's sensors. The whole process includes many basic problems. The first problem is the perception problem. The robot needs to acquire knowledge about the environment. It usually does this by taking measurements using various sensors, such as sonar and a camera—and then extracting meaningful information from those measurements [1]. The localization problem can be simply described as letting the robot know where it is. After localization, the robot can then undertake navigation and path planning to determine where it is going and how to get there.

The SLAM problem is not easy to solve because of sensors' noise. Localization is based on robot sensor data. The sensors' noise will cause inaccurate localization, and this inaccuracy will accumulate. In recent years, researchers have found some ways to reduce the errors, such as an Extended Kalman Filter (EKF) SLAM [4], Graph-SLAM [5], and particle-filter SLAM [6].

2.3 Image feature extraction and matching

A digital image is formed by multiple pixels. It is complicated and difficult to match that much data. To achieve fast and accurate matching, it is necessary to find a matching algorithm. There are two categories of matching methods [7]. The first is gray matching, which is finding point matching by using statistical methods. Another method is feature-based matching. This method is based on matching the key features extracted from images such as points, lines, and surfaces. Each method has its own advantage. Gray matching has high accuracy but consumes high computational power. The feature-based matching method, on the other hand, usually needs less computation and has a better anti-noise ability.

Feature-based matching first requires extracting features from images. There are many kinds of feature detection algorithms for different purposes. The Harris algorithm [8] and SUSAN algorithm [9] extract the corner points. The Canny [10] and Sobel [11] algorithms and Hough transform [12] can detect the edges in images. The feature detection algorithm used in this thesis was Speeded Up Robust Features (SURF). SURF is a robust and fast feature detector that was presented by Herbert Bay et al. [13] in May 2006. SURF is based on another feature detection algorithm, SIFT [14], but it has some advantages over SIFT, such as being more robust and faster.

2.4 Camera calibration

A camera captures a digital image in which the two-dimensional (2D) points are mapped from a three-dimensional (3D) object. The camera calibration is the process used to estimate the camera's parameters by the photos. A camera has many parameters. The focal length, image sensor format, and principal point are called intrinsic parameters. The extrinsic parameters define the position of the camera's center and the camera's heading in world coordinates [15]. Every camera parameter influences the object-to-image mapping result. There are various approaches to calculating the camera parameters. The direct linear transformation (DLT) method [16] is the most commonly used camera calibration method. The Tsai algorithm is an accurate camera calibration technique that was introduced by Tsai R.Y. in 1986 [17]. Additionally, Zhang Z.Y. proposed a flexible new technique for camera calibration in 2000 [18].

2.5 Map merging

In a robot team, each robot uses its own coordinate system during the SLAM process. So the different maps built by different robots can't be merged directly. The maps usually need rotations or translations before merging. Researchers have used different ways to achieve map merging. Usually, map merging can be done successfully with basically two kinds of prerequisites. The first prerequisite is the robots know their relative positions. For example, Zhou et al. [19] proposed a way to calculate a map transformation matrix by measuring the distance between robots by using omnidirectional cameras when robots encounter during the detection process. Another prerequisite is that the maps need common regions. In [20], H.C. Lee et al matched the maps’ point features. Carpin [21] proposed a merging algorithm based on the maps’ spectral information.

Chapter 3 Simulation tools

Given its many advantages over real robots, simulation is a good way to verify an algorithm. Real robots can be very expensive and may also be difficult to build, use, or maintain. Furthermore some robot test environments, such as disaster ruins or emergency environments for rescue robots, are hard to build. These environments are relatively rare and almost impossible to set up for testing real robots. Thus, simulation is important and sometimes necessary. All of the experiments in this thesis were based on simulations using the USARSim simulation tool.

3.1 Introduction

USARSim was designed as a high-fidelity simulation of urban search and rescue (USAR) robots and environments and was intended as a research tool for the study of human–robot interaction and multi-robot coordination [22]. This simulation tool was built based on the Unreal Engine game engine. Unreal Engine provides great visual rendering and physical modeling. The USARSim developers built many environmental models, commercial and experimental robot models, and sensor models. The simulation user can use USARSim to build user interfaces, automations, coordination logic etc.

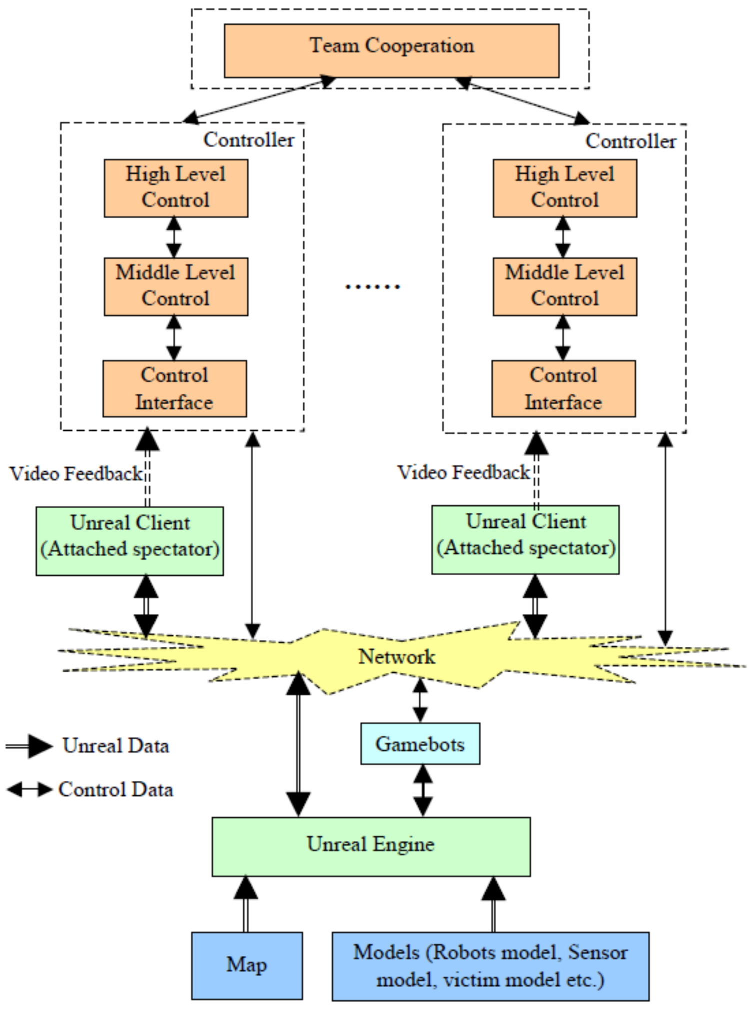

Figure 1. The simulation system architecture

3.2 System architecture

The USARSim system contains three main components: (1) the Unreal Engine, which is the server, (2) Gamebots, which connects the server and the client, and (3) the controller, which controls the robots in the simulator.

3.2.1 Unreal engine

Unreal Tournament 2004 is a 3D multiplayer first-person-shooting game. The game development engine, Unreal Engine, which was developed by Epic Games, created *Unreal Tournament 2004*. A game engine is a software framework designed for the creation and development of video games [23]. The core functions of a game engine typically include a renderer for 2D or 3D graphics, a physics engine or collision detection, sound, scripting, animation, artificial intelligence, and networking. Unreal Engine is a very good game engine. It provides everything for game developers. This engine was used to develop many famous 3D games including *BioShock* and *Borderlands*. The Unreal Engine provides the necessary functions for robot simulation: a rendering engine to display robots, sensors, and map models, and a physics engine to give a realistic sense of the laws of physics in the simulation. Unreal Engine provides an editor called Unreal Editor that the game developers can use to build their own maps and models.

3.2.2 Gamebots

Gamebots is a modification of *Unreal Tournament 2004*. To control the robot models in the Gamebots game, users need to access *Unreal Tournament 2004* from other applications. Gamebots built a bridge from Unreal Engine to other outside applications [24]. It uses a TCP/IP socket in Unreal Engine and transfers data with outside applications. By using Gamebots, users can send their own control commands from different applications to USARSim.

3.2.3 USARSim Models

Unreal Engine provides a foundation for simulation. It is like an empty world that users need to put robots, sensors, and maps to conduct experiments. The USARSim developers already prepared several robots, sensors, and maps for Unreal Engine.

3.2.3.1 Robot simulation

The USARSim developers built a robot model to simulate the mechanical robot. This model includes the chassis, parts, and other items such as headlights and cameras. All the chassis and parts are connected through simulated joints driven by torques. In USARSim, the robots are built based on this model. There are eight ground robots provided by USARSim: P2AT, P2DX, ATRV-Jr, Zerg, Tarantula, Talon, Telemax, and Soryu. USARSim also provides two two-legged robots and an aerial robot. Users can build their own robots based on the robot model.

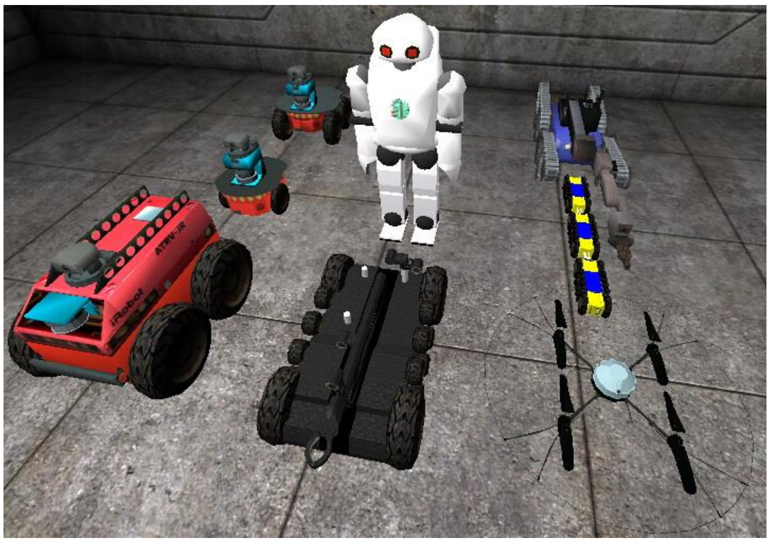

Figure 2. Various robot models in USARSim

3.2.3.2 Sensor simulation

Sensors are crucial to robot control. By checking the object's state and performing some calculations, USARSim provides multiple sensors that can be mainly divided into three categories:

1) Proprioceptive sensors, including battery state and headlight state

2) Position estimation sensors, including location, rotation, and velocity sensors

3) Perception sensors, including sonar, laser, and RFID

USARSim provides a file to configure all the sensors' parameters, such as the sensors' maximum range, precision, and resolution.

3.2.3.3 Maps

In addition, there are a few maps in the USARSim toolbox. These maps were used in some robot rescue competitions. In this thesis, the simulation was based on a simple map which has walls and obstacles. This map was built by Unreal Editor.

3.2.4 Matlab

Users can choose different controllers; the controller in my simulation was MATLAB. MATLAB is a multi-paradigm numerical computing environment and fourth-generation programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran, and Python. The Scalable Autonomous

Systems Laboratory at Drexel University built a MATLAB toolbox for USARSim [25]. This toolbox gives users the ability to interface with USARSim and develop robot control programs in MATLAB. Users can send commands from MATLAB to USARSim to complete tasks like initializing a robot and setting a robot's speed. Also, the MATLAB program can read the sensor models' data from USARSim. By using this toolbox, users can write their own robot control programs with MATLAB.

Chapter4 SLAM

4.1 Introduction

Autonomous navigation is a basic, but important, function for an autonomous mobile robot. The task entails deploying multiple robots in an unknown environment. Each robot can detect the environment separately and eventually build a whole map together. Basically, this task contains four main problems for robots. Each robot needs to know: (1) “Where am I?” (2) “Where am I going?” (3) “How can I get there?” and (4) “What does the place around me look like?” The first two problems are localization problems, the third is a path plan problem, and the fourth is a mapping problem [26]. The whole autonomous mapping task for multi-robots is actually a task that integrates mapping, localization, and path planning tasks and all three tasks are interconnected. The solutions of localization and path planning are usually based on a known map. However, the map is initially unknown and is waiting for detection in this task. To build a map, the robot’s location is necessary. The key challenges for the mapping problem are how to express the environment and how to process the sensors’ data.

The map is actually built step by step with the robot’s motion. The whole map is like the integration of many separate maps at different locations. Thus, localization is necessary for mapping. The robot must first know where it is and then detect the surroundings and build a map of this position. At the same time, it is hard for the robot to estimate where it is in an unknown place without a map. Accordingly, the SLAM problem is often described as a chicken-and-egg problem—the robot needs to know its location for map building, yet it is hard to know its location without the map.

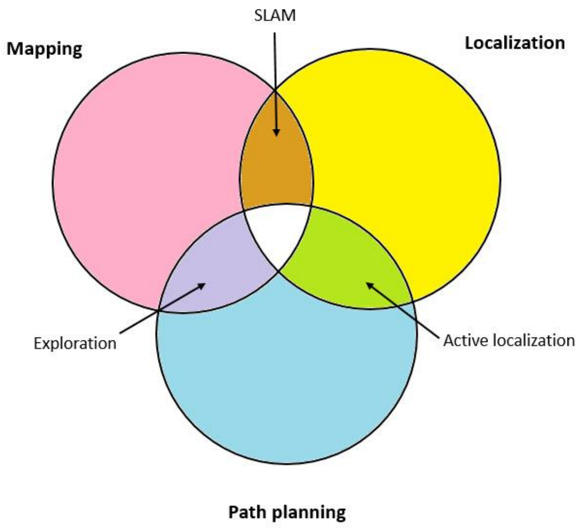

Figure 3. Localization, mapping and path planning

4.2 SLAM errors

Another reason that the SLAM problem is not easy to solve is because of the noise problem. The robot acquires knowledge of the environment by using sensors, which always have noise. Suppose a robot's initial position has no uncertainty. As the robot moves, the odometry sensors calculate the robot's pose. Because of the odometry sensors' noise, the robot's pose may not be precise. Furthermore, the inaccuracy will be greater because the next pose calculation is based on the previous pose. A partial map is built at each of the robot's positions. Thus, if the robot's poses are not precise, the map will also be inaccurate [1]. Figure 4 illustrates how the pose uncertainty increases over the robot's estimate of absolute location for one landmark.

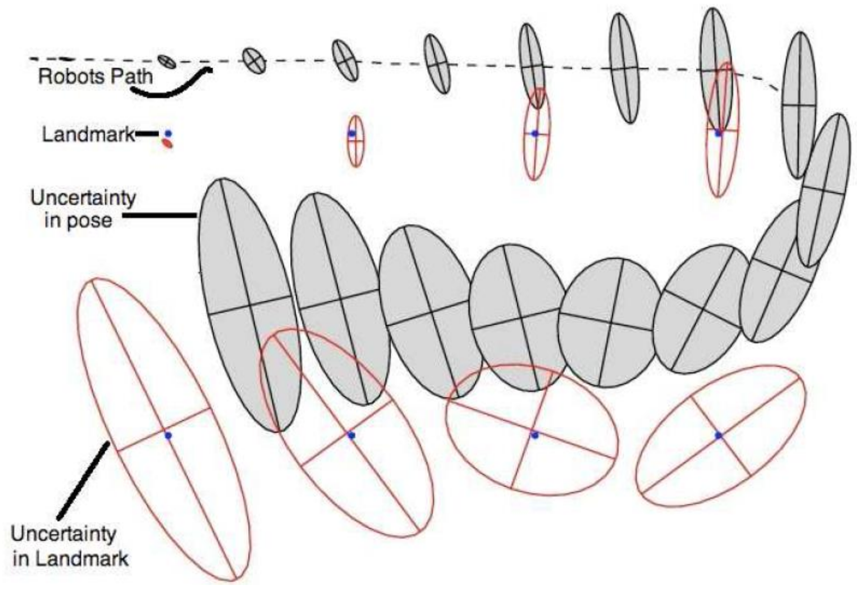

Figure 4. Pose uncertainty

4.3 Mathematical definition of SLAM

The robot path is a series of robot poses. We define the robot pose at time t by x_t, then the robot path X_T is written as:

$$X_T = \{x_0, x_1, x_2, \ldots, x_T\}$$

In SLAM, the robot's initial position x_0 is known.

Let u_T denote the robot's motion between time t and t-1. U_T is usually calculated using sensor data, such as data from the robot wheel encoders. The sequence of robot motion U_T is:

$$U_T = \{u_0, u_1, u_2, \ldots, u_T\}$$

Letting M denote the true map of the environment, the map is described as a series of landmarks m_i, with i = 0, 1, … n-1 as the landmarks. The landmarks can be points, lines, or planes:

$$M = \{m_0, m_1, m_2, \ldots, m_{n-1}\}$$

Finally, let Z_T denote the robot's observation at each pose. The observation could be sensor readings or photos captured by a camera:

$$Z_T = \{z_0, z_1, z_2, \ldots, z_T\}$$

Using this terminology, the SLAM problem can be defined as recovering a model of the map M and the robot path X_T from the odometry U_T and the observations Z_T. There are two kinds of SLAM problems, full SLAM and online SLAM. The full SLAM problem tries to recover the whole robot path X_T, whereas the online SLAM problem builds the map based only on its current pose x_T.

In order to solve the SLAM problem, the probabilistic motion model and probabilistic measurement model are necessary. The probabilistic motion model is derived from robot kinematics. The robot's current location x_t is computed as a function of the previous location x_{t-1} and the odometry sensor readings u_t.

$$P\ (x_t|x_{t-1}, u_t)$$

The probabilistic measurement model is derived from the sensor model, such as through the laser sonar or camera.

$$P\ (z_t|x_t, M)$$

The robot observation z_t is based on the known map M and the location x_t where the robot takes the observation.

Chapter 5 Image processing

5.1 Introduction

The key problem for map merging is figuring out the map transformation matrix. This matrix can be defined if different robots' relative positions are known. In this thesis, robots' relative positions were found by comparing their points of vision. If two robots have the same view, then these two robots are at the same place and their relative positions can be determined. Vision comparison belongs to the image matching technique. The first part of this chapter introduces an image matching algorithm. Usually, two robots' visions cannot be exactly the same. The second part of this chapter introduces the way to calculate the difference between robots' poses by image matching.

5.2 Feature detection

A digital image contains a lot of pixels. It is neither efficient nor robust to compare every pixel from two images for image matching. The solution is to extract some interest features from the images. These features contain the characteristics of the image, and then these features can represent the image. Feature detection is usually performed as the first operation in image matching. The detection examines every pixel to see if there is an interest feature present at that pixel.

5.3 SURF

5.3.1 Interest points detection based on Hessian Matrix

The SURF's approach for interest point detection uses Hessian-matrix approximation [27]. For a given pixel (x, y) in an image, the Hessian matrix H(x, σ) in x at scale σ is defined as:

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix}$$

$L_{xx}(x, \sigma)$ is a second order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ of a Gaussian. $g(\sigma)$ is a Gaussian filter:

$$g(\sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

A Gaussian is optimal for scale-space analysis [28]. The SURF is fast because it uses integral images and box filters to approximate $\frac{\partial^2}{\partial x^2} g(\sigma)$. This approximation entails a very low computational cost. The original image can be formed by different scales of image pyramids by using different sizes of boxes. A 9*9 box filter in Figure 5 is the approximation of $\frac{\partial^2}{\partial x^2} g(1.2)$. The approximate convolutions of pixels with box filter results are denoted by D_{xx}, D_{yy}, and D_{xy}.

Then the Hessian's determinant is approximated as:

$$\det(H_{approx}) = D_{xx} D_{yy} - (0.9 * D_{xy})^2$$

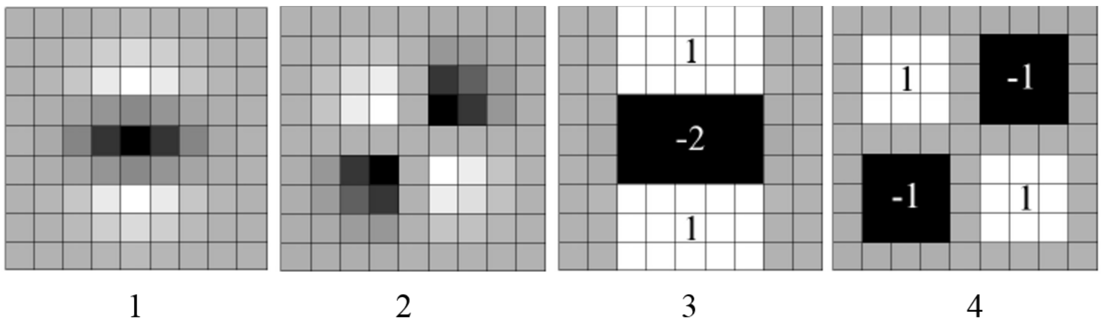

Figure 5. Box filter approximation. (1) Gaussian second-order partial derivative in y (L_{yy}); (2) Gaussian second-order partial derivative in xy (L_{xy}); (3) approximate box filter in y; and (4) approximate box filter in xy.

5.3.2 Scale space representation

It is better to extract interest points from different scales to ensure that the image matching is scale-invariant. Scale spaces are usually implemented as image pyramids. In the SIFT algorithm, the pyramid layers were built by down-sampling the image. Then, different-sized images built the image pyramid. In SURF, the image pyramid is built by changing the size of the box filter. This approach increases the processing speed.

5.3.3 Interest point localization and orientation

An interest point is defined by comparing the value with its neighboring 9*9*9 points in 3D space. The biggest values of the determinant of the Hessian matrix are interpolated in scale and image space with the method introduced in [29]. The descriptor of SURF is based on the sum of Haar wavelet responses. To make sure that the interest points are rotation invariant, it is better to define an orientation for every interest point. The dominant orientation is defined by calculating the sum of all responses within a sliding 60-degree orientation window. The longest vector defines the orientation of the interest point (Figure 6). Figure 7 shows the result of interest point detection.

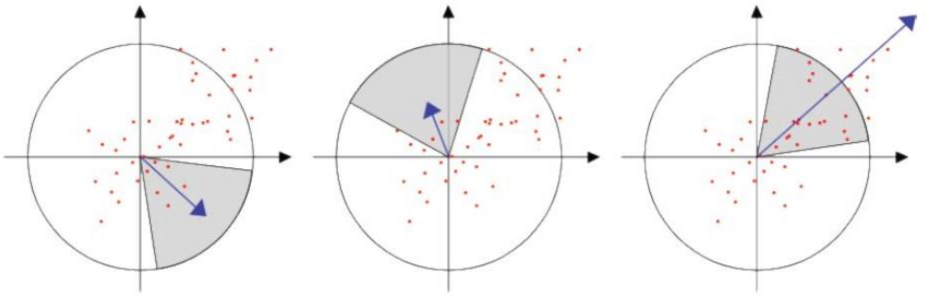

Figure 6. Interest point orientation

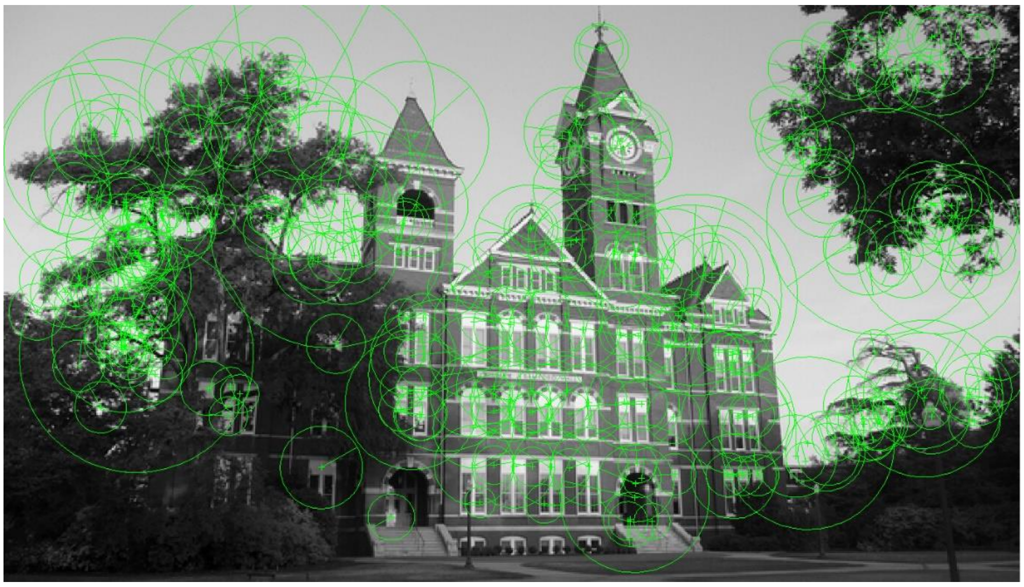

Figure 7. Detected interest points by SURF

5.3.4 Image matching

After the interest features detection, the key features are extracted from the images and the images can be matched. Figure 8 shows an example of image matching. The matched features between two images are identified and connected with several yellow lines. According to the matched features, a green quadrangle is plotted on the right image to refer to the matched region. It means that the left image matches this quadrangle region in the right image. By comparing the quadrangle's side length with the image's side length, the image's shrinkage value can be calculated. By comparing the center point's coordinates, the image's shift value can be calculated. However, the units for both the shrinkage value and shift value calculated in this process are in pixels rather than meters. The next step is to find a way to transfer the pixels in the image into meters in the real world.

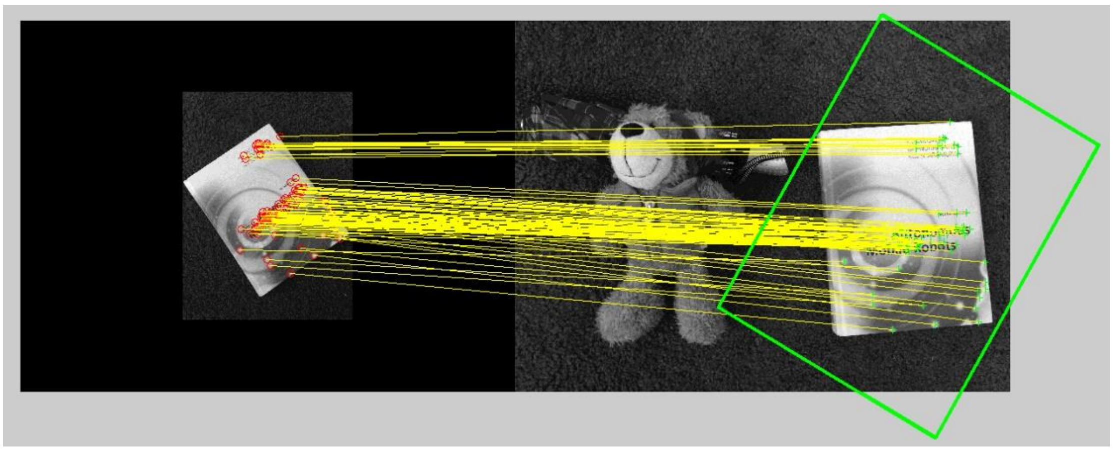

Figure 8. Image matching

5.4 Pinhole camera model

In an imaging system, every point on a 2D image is mapped from a corresponding point on a 3D object. The images are recorded by camera. The camera used in this thesis was a general projection camera. The projective geometry was used to examine the properties of the image. The basic pinhole camera geometry is shown in Figure 9 [30].

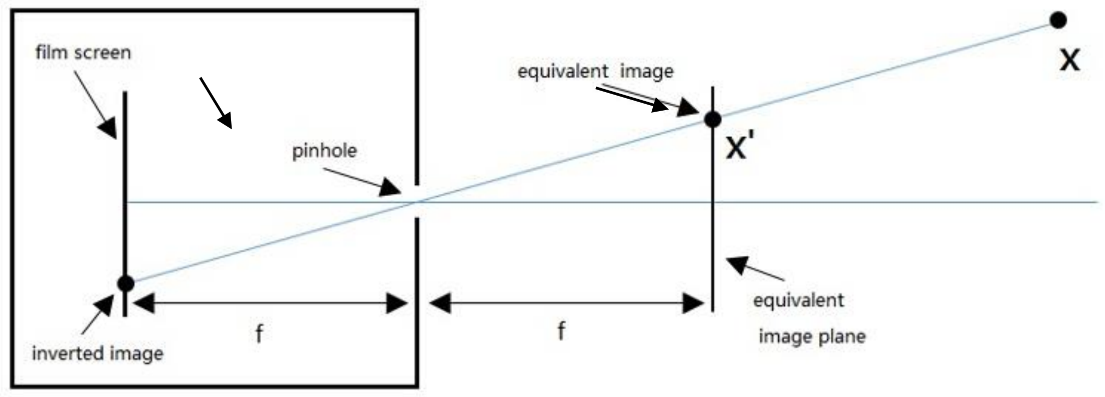

Figure 9. Pinhole camera model

The camera’s coordinate system (X_c, Y_c, Z_c) may not be the same as the real world’s coordinate system (X_w, Y_w, Z_w) (Figure 10). Two coordinate systems can be unified after translation and rotation. The equation can be written in homogeneous coordinates as:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \mathbf{M} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

R is a 3x3 orthogonal matrix represents for rotation. **t** is a 3x1 translation matrix[18].

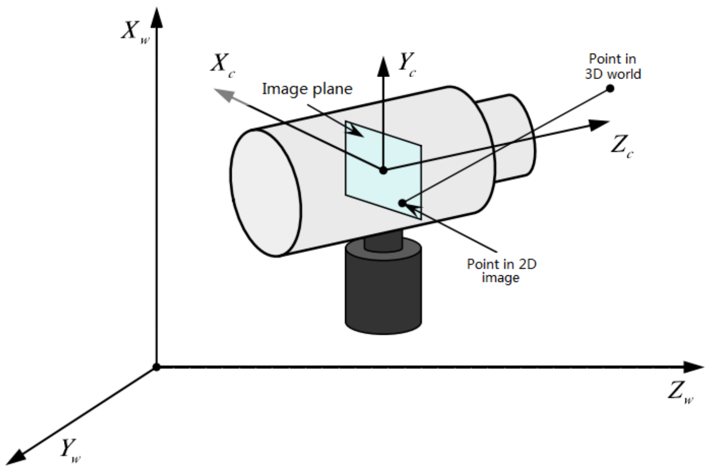

Figure 10. Real-world coordinate and camera coordinate

5.5 Image transformation with camera movement

A camera obtains the image, and the views differ with the movements of the camera. In general, the three kinds of camera movements are defined as:

1) Camera translation: The camera shifts vertically or horizontally. The direction of the camera's movement is parallel or perpendicular to the image plane.
2) Scale lens: As the focal length of the camera changes, the distance between the image plane and the object changes.
3) Camera rotation: The camera rotates along the x-, y-, or z-axis.

In this thesis, the robot only moved on flat ground, so its camera's focal length stayed the same. Thus, the camera could only do translations in the x- and z-axes and rotations in the y-axis. The goal was to find out the translation and rotation values between two cameras by comparing their views.

5.5.1 Translation in the x-axis

The camera movement direction is parallel to the image plane. The camera's moving distance is the same as the view's translation value.

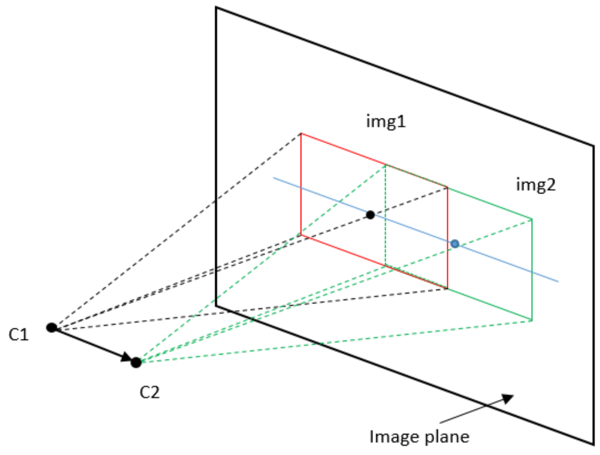

Figure 11. Camera translation in the x-axis

5.5.2 Translation in the z-axis

The camera's movement direction is perpendicular to the image plane, which causes a scaling of the view. The projection geometry and the top view of the projection are shown in Figure 12. The camera moves from C_1 to C_2. The image plane will actually move with the camera, as the focal length is fixed. In this case, the camera is closer to the object and the visual range is smaller. The top view of the projection shows the geometry relationship. The horizontal view angle will not change if the camera's focal length stays the same. Suppose the horizontal view angle is Θ. The tan ($\Theta/2$) equals half of the view's shrinkage value (M_1M_2) divided by the camera translation distance (C_1C_2).

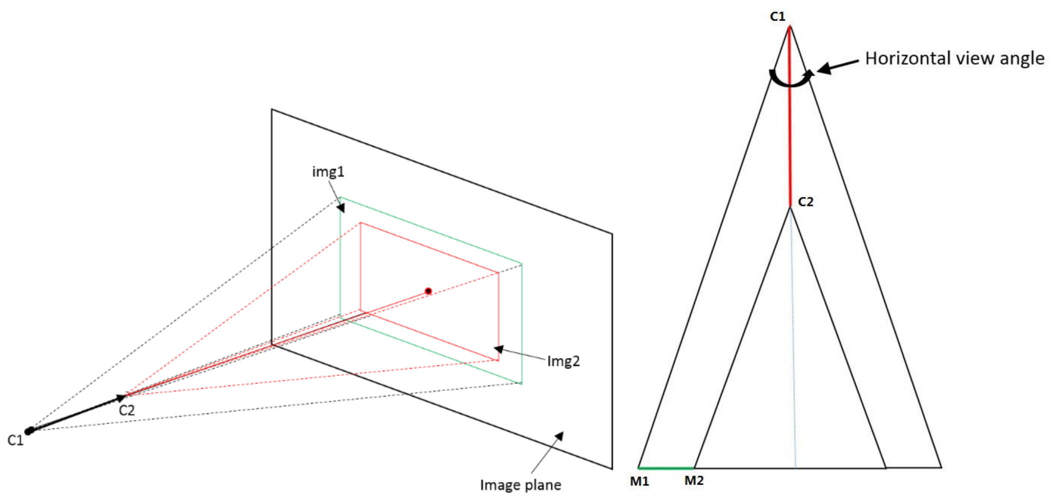

Figure 12. Camera translation in the z-axis

5.5.3 Rotation in y axis

The image transformation caused by camera rotation is more complicated than translations. It is common sense that an object looks bigger when it is nearer and smaller when it is farther away. Suppose the camera rotates to the left. It is easy to understand that the object in the image will shift to the right in reference to the original image. Also, the camera's rotation causes the object's right side to be nearer to the camera and its left side to be farther. This will lead to a transformation. After image matching, the shape of the matched region will be a trapezoid instead of a rectangle. An image matching example of camera rotation is shown in Figure 13.

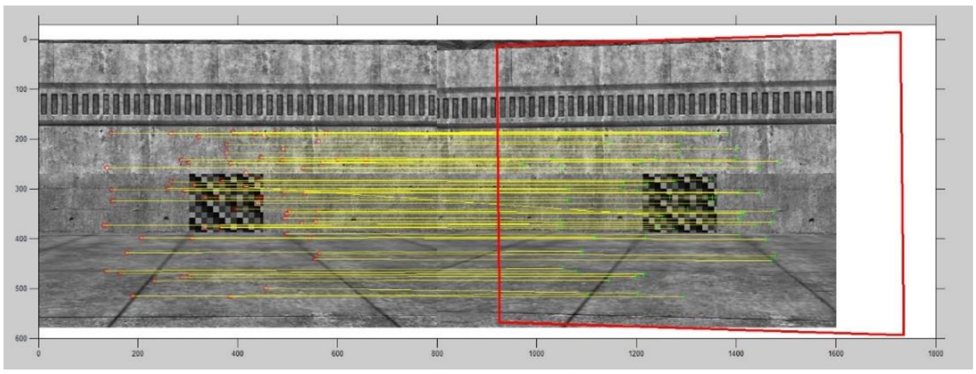

Figure 13. Image matching result with camera rotation

5.6 Image selection

This thesis used a special image selection technique to avoid camera rotation. When a robot’s movement is perpendicular to the wall in front of the robot, the robot camera’s optical axis is also perpendicular to the wall. At this time, the camera’s orientation is fixed. Therefore, the image selection rule is to only select the images that were taken at the time when the robot’s movement direction was perpendicular to the wall. This selection can ensure that the matched images have only scaling and translation, and not transformation.

Chapter 6 Experiments and results

6.1 Simulation setup

6.1.1 Models and simulation environment

6.1.1.1 Robots

The robot used in this simulation was P2DX. P2DX is a two-wheel, two-motor differential drive robot designed by Mobile Robots, Inc. The left image in Figure 14 is the real robot and the right image is the robot model used in the simulation.

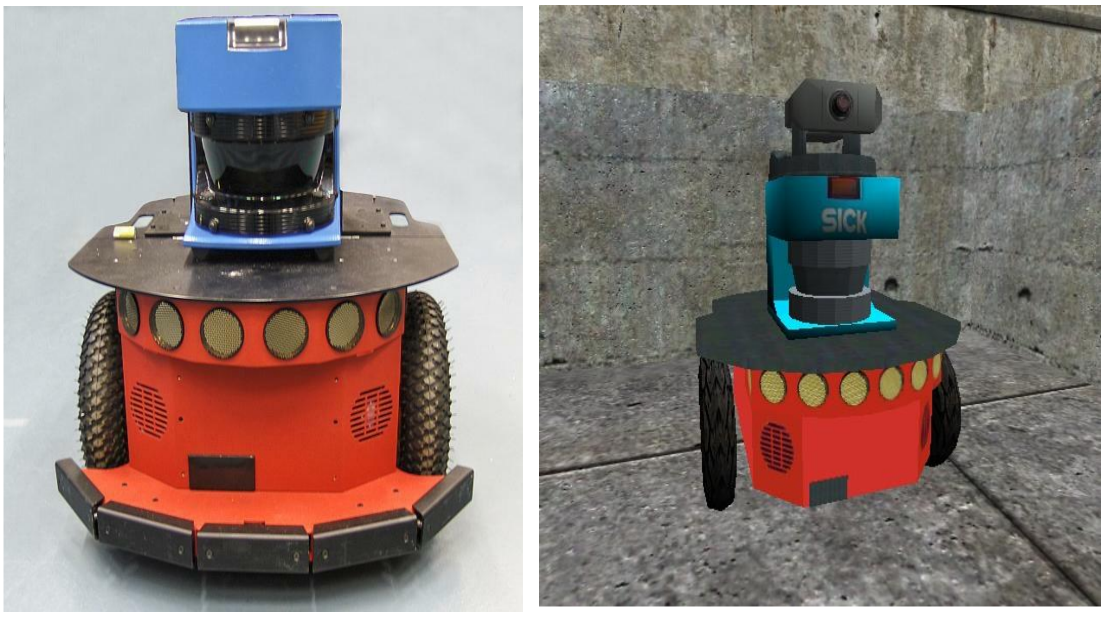

Figure 14. P2DX in the real world and in the simulation

Four kinds of sensors were used in the simulation. The first was the odometry sensor. The odometry sensor in USARSim uses the robot's left and right wheel encoders to estimate the robot's pose. The second are eight sonar sensors on the

P2DX, each with a range of two meters. The third is laser range sensor was used for mapping. This sensor is a 180-degree, four-meter range sensor with a resolution of one degree. Figure 15 illustrates the sonar sensors and a laser range sensor's reading. The laser range sensor's data was collected when the robot was in an empty room. The fourth sensor was a robot camera. The camera is a special sensor in USARSim. The developers of USARSim attached a viewpoint on the camera model. When users change to this viewpoint in Unreal Engine, the camera's view is shown (see Figure 16).

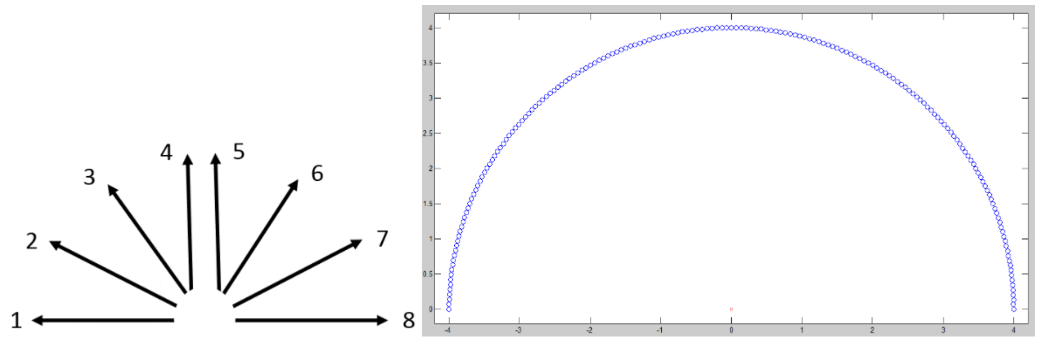

Figure 15. Sonar sensors and Laser range sensor

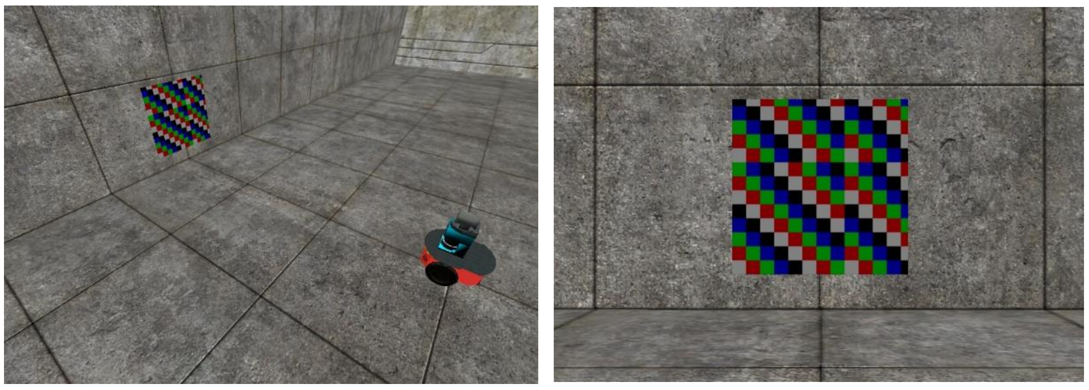

Figure 16. Robot in simulation and the scene in its camera

6.1.1.3 Map

USARSim users can build their own maps with Unreal Editor. The environment and map that were used in the simulation are shown in Figure 17.

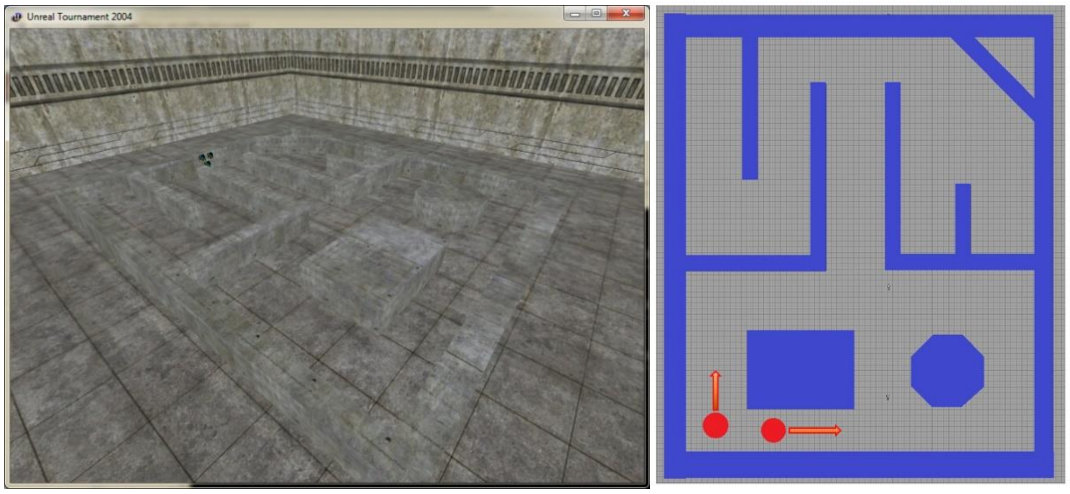

Figure 17. Simulation environment and its map

6.1.2 Fraps

Fraps is a software that can take screenshots or capture videos from the GPU and save them in a folder. USARSim users can change the viewpoint to the robot’s camera view. With this software, users can record the robot’s vision. Fraps can be set to capture screen images periodically. In this simulation, Fraps was set to take a screenshot every second. Fraps named the screenshots using the application name and the computer system’s time.

6.1.3 Matlab

The control software in this simulation was MATLAB. By using the MATLAB toolbox for USARSim, users can easily develop robot control programs. There are many functions of this toolbox. Some important MATLAB functions used in this simulation are described below [25]:

1） **initializeRobot.m**: Creates a robot in USARSim.

Usage: rob = initializeRobot('robot_name', 'robot_type', [robot position], [robot orientation]).

2） **getOdometryReadings.m**: Returns a sensor message for an odometry sensor on a robot as a MATLAB struct with: Name (String), Pose (1x3 vector).

Usage: odometry = getOdometryReadings(robot), where the robot is created with initializeRobot.m.

3） **sendDriveCommand.m**: Sends commands to drive the robot.

Usage: sendDriveCommand(robot, [left_motor_value, right_motor_value], command_Type), where the robot is created with initializeRobot.m, and command_type can be 'differential' or 'ackerman' depending on the robot type.

4） **getSonarReadings.m**: Returns a sensor message for sonar sensors on a robot as a MATLAB struct with: Name (String), Range (2 vector).

Usage: sonar = getSonarReadings(robot), where the robot is created with initializeRobot.m.

5） **getLaserSensorReadings.m**: Returns a sensor message for a laser range sensor on a robot as a MATLAB struct with: Name (String), Resolution (double), FOV (double), and Scans (2 vector).

Usage: laser = getLaserSensorReadings(robot), where the robot is created with initializeRobot.m.

6) `shutdownRobot.m` - Clears the robot from USARSim.

Usage: `shutdownRobot(robot)`, where `robot` is created with `initializeRobot.m`.

After the robot's SLAM control, MATLAB gathered a series of the sensors' data. The rest of the work, such as data processing, calculations, image matching, and map merging, was all achieved by MATLAB.

6.2 Experiments

6.2.1 Data collection

Two robots were set in the simulation. The localization problem can be solved by the function: getOdomtryReadings.m. The navigation and path planning problems can be solved by checking the sonar sensors' data. The basic navigation method is to let the robot always keep a fixed distance from the wall on its left or right side. This method is a good solution for an environment with many walls. In my simulation, one robot always stayed 0.6 meters from the walls on its left, and another always kept 0.6 meters from the walls on its right. To achieve this control method, the controller keeps reading the robot's left or right sonar sensor (sonar sensor1 or sonar sensor8, respectively) data to judge the distance between the wall and the robot. If this distance is greater or smaller than 0.6 meters, the controller will send a drive command to change the robot’s wheel speed to correct the deviation. The laser range sensor collects data during the robot’s moving process. A map can be built after the SLAM. Every time MATLAB reads the robot’s sensors’ data, a time value will be recorded so that the time and robot’s position can be associated. After the SLAM completed, Robot1 recorded 1,079 groups of data. Each group of data contains one time value, three pose data values, eight sonar sensors’ data values, and 181 laser range sensors’ data values. Robot2 recorded 1,020 groups of data. The software Fraps recorded the robots’ camera data and took screenshots every second during the SLAM. After the SLAM process, 134 images for Robot1 and 147 images for Robot2 were recorded.

6.2.2 Image selection

Image matching requires a lot of computations. It is not efficient to conduct image matching 134*147 times. The first step of image selection is finding the positions where the robot’s movement direction is perpendicular to the wall. It is easy to find these positions by checking whether the values of sonar sensor4 and sonar sensor5 are the same. After this selection, 21 positions for Robot1 and 148 positions for Robot2 remained. All of these positions could actually be divided into several groups, and many positions were very close to each other. The next step was to delete the positions that were close to each other, which left five positions for

Robot1 and 19 positions for Robot2. The last step was to find the images that were taken at these positions.

6.2.3 Image matching

MATLAB completed the image matching process. The steps of image matching in MATLAB are: (1) extract key features, (2) compare two images' key features and match them, (3) use the algorithm MSAC[31] and exclude unmatched features, and (4) sketch the matched region. After the image selection process, the shape of the matched region should be a rectangle. The length of these rectangular sides can be calculated. 5+19 images were left after image selection. Then, MATLAB completed image matching 5*19 times. Table 1 shows the number of matched points. The best-matched pair was (3, 10). Also, the pairs (3, 11) and (3, 12) had many matched points. This means that the position where Robot1 took the third image and the positions where Robot2 took the 10th through 12th images were very close.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	322	388	523	500	394	414	274	174	544	257	258	383	296	275	195	335	320	333	284
2	320	316	383	459	444	953	662	381	761	276	252	272	297	414	362	221	258	297	288
3	872	784	643	549	500	337	275	229	132	1335	992	808	587	394	357	287	243	250	245
4	745	627	611	562	556	958	739	484	482	668	555	505	406	344	303	289	431	384	376
5	551	616	660	589	497	639	631	703	290	542	574	568	405	259	205	218	281	352	331

Table 1. Number of matched points

6.2.4 Calculate distances between 2 robots

To calculate distances between robots, first choose two pairs of matched images. These two pairs of images contain three different images: one from Robot1 and two images from Robot2. Suppose that both pairs of images were taken at the common region in the environment, where both robots have similar views. These two robots' poses are known, but the poses are actually based on two different coordinates. The goal is to compare the views and calculate the distance between Robot1 and Robot2. Figure 18 shows the schematic. Suppose that Robot1's position is A' and Robot2's positions are D and E. The goal is to find out the translations (AA' and AD) between Robot1 at position A' and Robot2 at position D. Three robots' visions were collected at these three positions. B'C' is the length of Robot1's vision, FJ is the length of Robot2's vision at position D, and GH is the length of Robot2's vision at position E. All lengths of the robots' visions are in pixels. The camera's horizontal view angle is θ. The distance between Robot2's positions D and E can be calculated from Robot2's odometry data. This distance is the real-world distance, which is measured in meters. The relationship between θ, AD, DE, B'C', FJ, GH, and the pixel length-to-meter ratio K is:

$$\tan\frac{\theta}{2} = \frac{(\mathrm{BC}/2 - \mathrm{FJ}/2)\mathrm{K}}{\mathrm{AD}}$$
$$\tan\frac{\theta}{2} = \frac{(\mathrm{BC}/2 - \mathrm{GH}/2)\mathrm{K}}{\mathrm{AD} + \mathrm{DE}}$$

Solving these two equations solves K and AD. Finally, the translations between the two robots' reference positions were found.

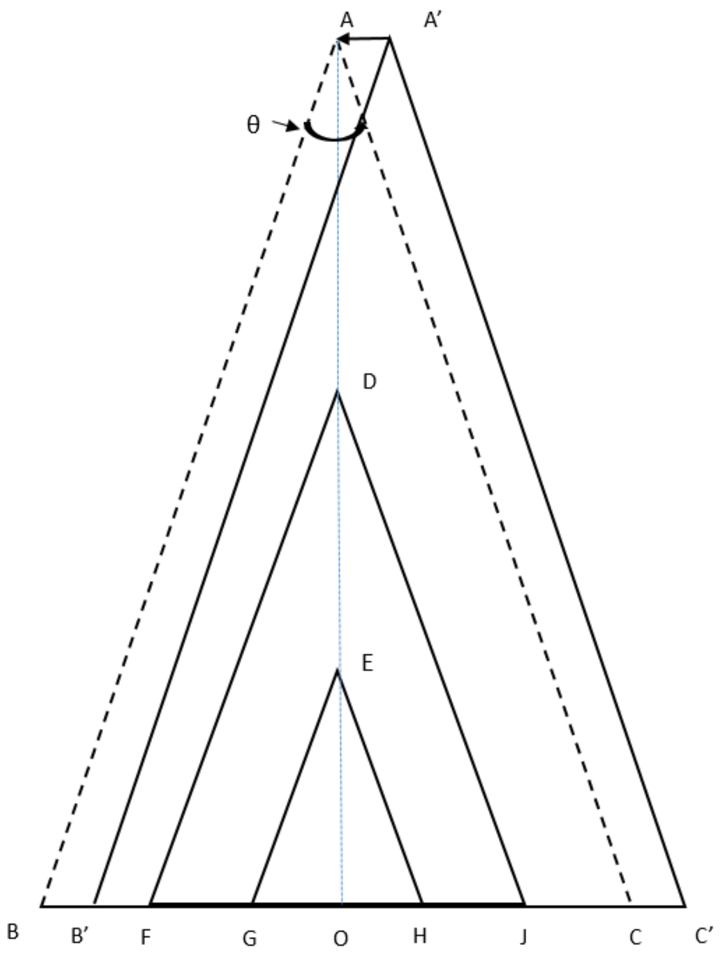

Figure 18. Schematic for calculating distance between 2 robots

6.2.5 Merge the maps

The translations between Robot1's position (x_1, y_1, θ_1) and Robot2's position (x_2, y_2, θ_2) were found in section 6.2.4. The last step is to transform one map and merge it into another. A map transformation contains a map rotation and map translation. The first step is to rotate Robot1's map. A new Robot1 map with the same orientation as Robot2's map can be transformed by multiplying Robot1's map with the rotation matrix. The rotation angle is θ_2-θ_1. Also, Robot1's odometry data needs the same rotation. Then Robot1's position (x_1, y_1, θ_1) changes to (x_1', y_1', θ_1'). Next, the translation value is the difference between Robot2's reference position and

Robot1's rotated position, plus the difference between the two robots' positions, which was calculated in section 6.2.4. In the end, the two maps merged into a single map. Figure 19 shows the map merging result.

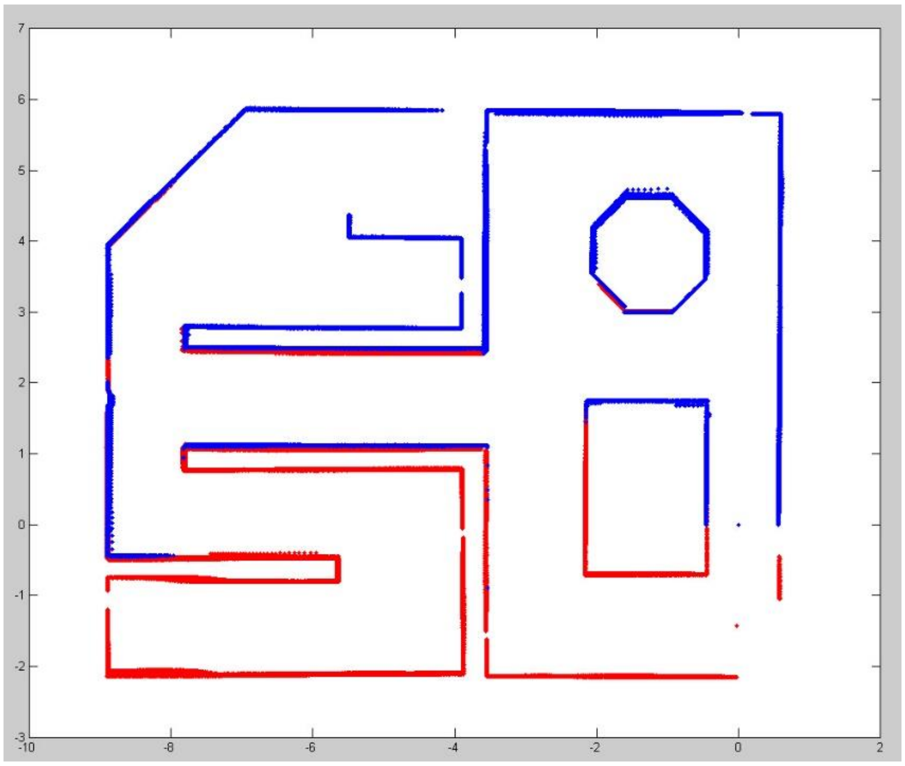

Figure 19. Map merging result

Chapter 7 Conclusion and Future Work

This thesis introduced a map merging solution. The key problem was to find the map transformation matrix. This matrix can be found if different robots' positions in their own maps and their relative locations are known. My idea was based on the hypothesis that different robots are at the same position when they have the same vision. By comparing the robots' visions, the robots' coordinate systems can be unified. A fidelity robotic simulation tool called USARSim was introduced in this thesis. USARSim and MATLAB were used to conduct the experiment. By using the image matching technique and some other calculations, the experiment successfully merged the maps.

Two main problems still exist. The first is the noise problem. Sensors always have noise. Although it is acceptable to do the SLAM without considering the sensor noise in a small environment like the experiment in this thesis, Chapter 3 explained how errors could accumulate. The noise problem will be much more serious in a big environment or a long-term SLAM. There are some solutions to minimize the SLAM errors, such as using an Extended Kalman Filter (EKF). This can be used in future work to lower the noise's influence.

Another problem is that the whole process of simulation still requires human judgment—it cannot be completed independently by a computer. This problem occurred during image selection. The prerequisite requires two robots' individual maps to have a common region. This means that both robots should experience a common place in the environment, but it is impossible for robots themselves to know if they are at this common region. The image matching result (Table 1) shows which pair of images are best matched images. This pair of images were taken at the common region in this experiment. But without the fiducial in common region, the best matched images may not be taken at the common region. Therefore, it is necessary to select the images that were taken in the common region by a human. Future work should find a way to solve this problem.

References

[1] Siegwart, Roland, and Illah R. Nourbakhsh. "Autonomous mobile robots." *Massachusetts Institute of Technology* (2004).

[2] Heon-Cheol Lee, Seung-Hwan Lee, Tae-Seok Lee, Doo-Jin Kim, “A survey of map merging techniques for cooperative-SLAM”, 2012 9th Internationl Conference on Ubiquitous Robots and Ambient Intelligence (URAI), daejeon, Korea, 2012

[3] Bekey, George A. *Autonomous robots: from biological inspiration to implementation and control*. MIT press, 2005.

[4] Finlayson, G. D., Hordley, S. D., Lu, C., & Drew, M. S. (2006). On the removal of shadows from images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 28*(1), 59-68.

[5] Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*. MIT press.

[6] Smith, R. C., & Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, *5*(4), 56-68.

[7] Huijuan, Zhang, and Hu Qiong. "Fast image matching based-on improved SURF algorithm." *Electronics, Communications and Control (ICECC), 2011 International Conference on*. IEEE, 2011.
[8] Mikolajczyk, Krystian, and Cordelia Schmid. "An affine invariant interest point detector." *Computer Vision—ECCV 2002*. Springer Berlin Heidelberg, 2002. 128-142.

[9] Smith, Stephen M., and J. Michael Brady. "SUSAN—a new approach to low level image processing." *International journal of computer vision* 23.1 (1997): 45-78.

[10] Canny, John. "A computational approach to edge detection." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6 (1986): 679-698.

[11] Kanopoulos, Nick, Nagesh Vasanthavada, and Robert L. Baker. "Design of an image edge detection filter using the Sobel operator." *Solid-State Circuits, IEEE Journal of* 23.2 (1988): 358-367.

[12] Ballard, Dana H. "Generalizing the Hough transform to detect arbitrary shapes." *Pattern recognition* 13.2 (1981): 111-122.

[13] Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features." *Computer vision–ECCV 2006*. Springer Berlin Heidelberg, 2006. 404-417.

[14] Lowe, David G. "Object recognition from local scale-invariant features." *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. Vol. 2. Ieee, 1999.

[15] Hartley, Richard, and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[16] Shapiro, Robert. "Direct linear transformation method for three-dimensional cinematography." *Research Quarterly. American Alliance for Health, Physical Education and Recreation* 49.2 (1978): 197-205.

[17] Tsai, Roger Y. "An efficient and accurate camera calibration technique for 3D machine vision." *Proc. IEEE Conf. on Computer Vision and Pattern Recognition, 1986*. 1986.

[18] Zhang, Zhengyou. "A flexible new technique for camera calibration." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.11 (2000): 1330-1334.

[19] Zhou, Xun S., and Stergios I. Roumeliotis. "Multi-robot SLAM with unknown initial correspondence: The robot rendezvous case." *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006.

[20] H. C. Lee et al., “Probabilistic Map Merging for Multi-Robot RBPF-SLAM with Unknown Initial Poses,” *ROBOTICA*, Vol. 30, pp. 205-220, 2012.

[21] Lee, Heon-Cheol, and Beom-Hee Lee. "Improved feature map merging using virtual supporting lines for multi-robot systems." *Advanced Robotics* 25.13-14 (2011): 1675-1696.

[22] Carpin, Stefano, et al. "USARSim: a robot simulator for research and education." *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007.
[23] Lewis, Michael, and Jeffrey Jacobson. "Game engines." *Communications of the ACM* 45.1 (2002): 27.

[24] Kaminka, Gal A., et al. "Gamebots: a flexible test bed for multiagent team research." *Communications of the ACM* 45.1 (2002): 43-45.

[25] Matlab USARSim toolbox. URL: http://robotics.mem.drexel.edu/USAR/

[26] Grisetti, Giorgio, Cyrill Stachniss, and Wolfram Burgard. "Improved techniques for grid mapping with rao-blackwellized particle filters." *Robotics, IEEE Transactions on* 23.1 (2007): 34-46.

[27] Lindeberg, Tony. "Feature detection with automatic scale selection." *International journal of computer vision* 30.2 (1998): 79-116.

[28] Lindeberg, Tony. "Scale-space for discrete signals." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 12.3 (1990): 234-254.

[29] Brown, Matthew, and David G. Lowe. "Invariant Features from Interest Point Groups." *BMVC*. No. s 1. 2002.

[30] Fitzpatrick, J. Michael, Derek LG Hill, and Calvin R. Maurer Jr. "Image registration." *Handbook of medical imaging* 2 (2000): 447-513.

[31] Torr, Philip HS, and Andrew Zisserman. "MLESAC: A new robust estimator with application to estimating image geometry." *Computer Vision and Image Understanding* 78.1 (2000): 138-156.