

A Feature-based Solution for Kidnapped Robot Problem

by

Lei Wei

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
December 12, 2015

Keywords: Kidnapped robot problem, Robot localization, Feature matching, SIFT

Copyright 2015 by Lei Wei

Approved by

Thaddeus Roppel, Chair, Associate Professor of Electrical and Computer Engineering
John Hung, Professor of Electrical and Computer Engineering
R. Mark Nelms, Professor of Electrical and Computer Engineering

Abstract

The Kidnapped Robot problem in robotics commonly refers to a situation where an autonomous robot can't locate itself against the map. It can be caused by external force when a robot is carried to an arbitrary place, or experiencing a wake-up sequence, etc.

In this thesis, a solution is proposed and implemented using feature matching to solve the problem in an efficient and robust way. First a flood coverage algorithm is proposed to mark the building with limited number of fiducial markers and environmental features. Then a match-then-vote method is implemented based on ORB (Oriented FAST and Rotated BRIEF) and SIFT matching algorithms to recognize the place. The robot's position can then be estimated by its distance and angle to the marker. After calibration by an AMCL (Adaptive Monte-Carlo Localization) node in ROS (Robot Operating System) using feature matching, the position message will be published in the robot operating system to locate the robot in a map.

After tested in Auburn University Broun Hall, the solution proves to be a functional method in kidnapped recovery with acceptable error and speed: The average error of the distance measurement is below 1%. The average error in position estimation is about 0.31m while the angle error in the pose estimation is less than 3.5 degrees. And the recovery process with marker in sight cost less than 1.5 second.

Acknowledgments

I would first and foremost like to thank my advisor Dr. Thaddeus Roppel for his support and guidance during my time at Auburn University. Dr. Roppel gave me the opportunity to be involved into the cooperative robotics lab in EE department during my graduate studying in Auburn, and bring me in touch with a lot resources during my research. I also want to thank Dr.Nelms and Dr.Hung for serving on my thesis committee.

I also want to thank my parents, who trust me, support me and love me all these years. And finally, I want to thank all my friends, for their words and their time listening my boring ideas.

Table of Contents

Abstract	ii
Acknowledgments.....	iii
List of Figures	vii
List of Tables.....	ix
List of Abbreviations.....	x
Chapter 1: Introduction	1
1.1 Goals	2
1.2 Motivation	2
Chapter 2 Literature Review	4
2.1 Autonomous Robot and Kidnapped Robot Problem.....	4
2.2 Robot Operating System (ROS).....	6
2.3 Computer Vision in Feature Detection and Matching	7
2.3.1 Scale-Invariant Feature Transform (SIFT).....	7
2.3.2 Oriented FAST and Rotated BRIEF (ORB).....	10
2.3.3 Feature Matching	11

2.3.4	OpenCV (Open Source Computer Vision)	12
2.4	Hardware Specification	12
2.4.1	Hokuyo LIDAR	12
2.4.2	Logitech C920 Camera	13
Chapter 3 Solution Implementation		14
3.1	Mark the Building	16
3.1.1	Find Out the FOV and Angle Limitation of the Robot's View	19
3.1.2	Pick a Start Point of the Map and Start Marking Sequence	20
3.1.3	Form and Maintain a Feature Library	25
3.1.4	Mark the Rooms If Needed	26
3.2	Place Recognition	26
3.3	Position Estimation	30
3.3.1	Distance Estimation	30
3.3.2	When the Place Is a Hallway	34
3.3.3	When the Place Is a Room	35
3.4	Error Prediction	36
3.4.1	Error in Place Recognition	36
3.4.2	Error in Position Estimation	36
Chapter 4 Experiment Setup and Results		38

4.1	Marking the Map.....	39
4.2	The Experiment Recovery System Implementation	43
4.3	The Recovery Result and Analyzation.....	46
4.3.1	Random Recovery Test.....	46
4.3.2	Experiment for Error Observation	48
4.3.3	The Speed of the Recovery Program	50
Chapter 5 Conclusion and future work		51
5.1	Summary and Recommendation	51
5.2	Future Work	52
References		53

List of Figures

Figure 1.	Hallway Kidnapped Robot Example	5
Figure 2.	DOG Computing Process [22].....	8
Figure 3.	FAST corner detection [25].....	10
Figure 4.	Hallway Map Example	14
Figure 5.	Fiducial Marker Example	16
Figure 6.	Hallway Marking Example.....	20
Figure 7.	Hallway Marking Flood.....	21
Figure 8.	Hallway Marker	22
Figure 9.	Main Marking Sequence Flow Chart.....	24
Figure 10.	Camera projection model y-z plane.....	31
Figure 11.	Camera projection model x-z plain.....	32
Figure 12.	Hallway Position Illustration	34
Figure 13.	Room Position Illustration.....	35
Figure 14.	Bivariate Normal Distribution	37
Figure 15.	Broun Hall Third Floor West Wing Test Map.....	38

Figure 16. Trigger Marker Used In Experiment	39
Figure 17. FOV Test Matching	40
Figure 18. Place Candidates in Map	41
Figure 19. Message Flow in Recovery System.....	44
Figure 20. Terminal Window of the Recovery Node.....	45
Figure 21. Recovery Result Showing in RVIZ.....	46
Figure 22 Recovery Speed	50

List of Tables

Table 1. Feature Library.....	25
Table 2. Voting after object 1	28
Table 3. Voting after object 2	29
Table 4. Voting after object 3	29
Table 5. Voting after object 4	29
Table 6. Voting After object 5	30
Table 7. Feature Lib after Place A Added.....	41
Table 8. Feature Lib after Place B, D Added.....	42
Table 9. Distance Test for Candidate E and H.....	42
Table 11 Random Recovery Test 1	47
Table 13 Recovery test 3 result.....	49

List of Abbreviations

ROS	Robot Operating System
AMCL	Adaptive Monte-Carlo Localization
CRRLAB	Cooperative Robotics Research Lab
FOV	Field of View
LIDAR	Light Detection and Ranging
RVIZ	Robot Visualization Tool
SLAM	Simultaneous Location and Mapping
RFID	Radio frequency identification
DOG	Difference of Gaussian function
SIFT	Scale-invariant feature transform
SURF	Speeded-Up Robust Features
FAST	Features from Accelerated Segment Test
BRIEF	Binary Robust Independent Elementary Features
ORB	Oriented FAST and Rotated BRIEF
FLANN	Fast Library for Approximate Nearest Neighbors

Chapter 1: Introduction

Although the term ‘Robot’ was first coined by Asimov in his science fiction in 1941, the idea of automata can be traced back to many ancient civilizations, like ancient China and Ptolemaic Egypt [1]. An account in Lie Zi, a 3rd century text book, describes a humanoid automata. The automata built by Yan Shi, was presented to the king with a life-size, human-shaped figure, and filled with artificial organs made by leather and wood. [2] With years of development, robot is now a big family with many branches, range from commercial manufacture to medical agent inside a human body [3].

Today, many of the robots are equipped with legs, wheels or propeller so they can move in different terrain or space to finish their task. When a robot is capable of moving, the first problem emerges will be the localization problem. To gain accurate knowledge of robot’s position within its operating environment, many solutions have been proposed based on human interaction, computer vision, RFID or LIDAR feature. Many of them are combing with a probabilistic model [4] [5], which is using the recent acquired information and the previous robot state information to estimate its current position. This kind of model needs to keep tracking the robot past positions to give more precise estimation. What if a robot experiences a power issue or is moved by external force, and can’t locate itself? This is when the kidnapped problems comes up.

1.1 Goals

The work presented in this thesis aims to develop and implement a feature-based solution to recover the autonomous robot's location when it has been kidnapped. The system, implemented in ROS, is mainly based on feature detection using a single camera and LIDAR. The solution also proposes a method to mark building hallway and rooms, and maintain a library of objects for detection.

1.2 Motivation

Released in 2009, ROS is a flexible framework for robot operating packages. Since it is open source and compatible with multiple platforms, it has been widely used around the world [6]. Its developers and users have formed a community in which many program packages are produced and shared every day [7]. One of the most frequently used programs is ROS navigation stack. The navigation stack provided by Eitan Marder-Eppstein is a simple and efficient solution for 2D navigation [8]. However, when the program has been started up and given a task, it still needs an input of initial position to localize the robot against a map. This means a robot system with navigation stack on board is still a human-in-the-loop system.

Here, the gap between autonomous and human-in-the-loop control has been referred to as the kidnapped robot problem [9] [14]. Solutions proposed based on computer vision can be separated into two branches.

The first group of solutions use computer vision combined with markers for place recognition and localization, like in [10], [11] and [12]. In those solutions based on fiducial markers, the robot usually needs three markers to locate itself against the map [13]. These algorithms not only

require multiple pictures of robot's surroundings (at least two or more), but also need a large set of fiducial markers. Both of them will reduce the recovery speed performance.

The other branch of solutions is based on visual words, like K-means clustering algorithms combined with a vocabulary tree in [20]. These solutions will need a large set of training pictures to achieve a high accuracy.

The solution proposed in this thesis is an attempt to combine those two branches together. Instead of a large set of fiducial markers, very limited number of markers combined with a set of training objects will be used to mark the place. Instead of computing with multiple pictures, the algorithm will finish recognition with only one picture. And a voting algorithm based on fuzzy logic similar to vocabulary tree in [20] will be used to increase the speed performance.

The remainder of this thesis is organized in the following manner: Chapter 2 provides an overview of the field of autonomous robot and computer vision with focuses on kidnapped robot problem and object matching. Chapter 3 gives a specific implementation of the algorithm, followed by Chapter 4, which shows the experiment design and result to demonstrate the solution. Chapter 5 presents the conclusion, suggestions and future work.

Chapter 2 Literature Review

2.1 Autonomous Robot and Kidnapped Robot Problem

Autonomous robot is an intelligent machine equipped with computational resources, capable of performing tasks in the world without human interaction [16]. Nyagudi summarized the following rules for what an autonomous robot could do in [17]:

1. Gain information about the environment.
2. Work for an extended period without human intervention.
3. Move itself throughout its operating environment without human assistance.
4. Avoid situations that are harmful to people, property, or itself unless those are part of its design specifications.

Kidnapped problems in robotics commonly refer to a situation when an autonomous robot in action lost its location information. It can be caused by external force when a robot is carried to an arbitrary place, or experiencing a wake-up sequence, etc. No matter what situation it is, when a robot is kidnapped, it means the localization algorithm it uses cannot give its accurate position information in current environment, thus the task assigned to it will be forced to stop.

Not all the localization algorithms could solve the kidnapped problems properly. For example, the localization method used in ROS navigation stack will sometimes fail a kidnapped recovery because of its own defect. The algorithm uses a LIDAR feature matching along with a probabilistic model to localize the robot against a 2-D map. To demonstrate the kidnapped

situation, let us consider the map shown in Figure 1, which is the Auburn University Broun Hall third floor hallway map.

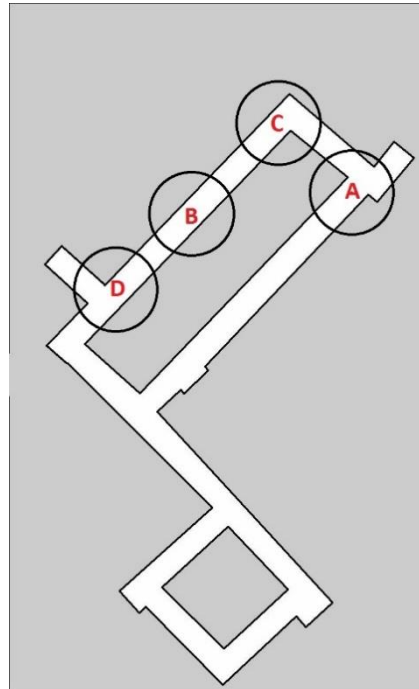


Figure 1. Hallway Kidnapped Robot Example

Assuming the robot is working autonomously at position A with a circle indicating its LIDAR detection range. Then it is kidnapped to B. We can see with the information gathered from LIDAR, the system could report a matching with several places in the map. Tested in ROS navigation stack, the kidnapped situation will trigger a recovery sequence, which means the robot will rotate to gather more LIDAR information in position B. The recovery proves to be a failure every time, simply because the LIDAR feature input at position B will never be enough to recognize the place, nor will the former position estimations in the probabilistic model be of any help.

In the situation above, the kidnapped problem could be solved by adding a program package, to hang up the former task and send an explore command. So when it moves to position C or

position D, the feature matching will give a more influential matching report, and then recover its position by a probabilistic model. But this solution could add a huge latency when the exploration process interrupts the given task of the robot for too long.

There are several other localization algorithms that are capable of solving the kidnapped robot problem, for example a semantic localization method proposed by Chuho Yi and Byung-Uk in [18]. feature based SLAM in [19], or computer vision combined with a vocabulary tree in [20]. These localization algorithms above are not primarily designed to solve the kidnapped robot problem, but after the kidnapped situation occurs, these solutions are capable of recovering the robot's position in the map.

2.2 Robot Operating System (ROS)

Robot Operating System, aka ROS, is a collection of software frameworks for robot software development. It was first implemented in Stanford AI Lab, to support Stanford AI Robot STAIR project [21]. After that, it was developed and maintained by Willow Garage, a robot research institute. Now, ROS has become an open source platform provide standard operating system services, such as hardware abstraction, low level device control, application implementation, message passing between multiple level processes, and file management, etc.

In this thesis, the solution will be implemented into a ROS node, the most common and basic level of application in ROS system. When running, this node could communicate and transfer data with other nodes by ROS messages. The message will be published with a standard format under a certain topic, so other node could subscribe to this topic and read this message.

Two main nodes will be used in this solution, one is the AMCL node, provided by ROS navigation stack. The function of this node is to localize the robot against a given map, using

LIDAR feature matching and an adaptive (or KLD-sampling) Monte Carlo localization approach. Another one is based on python language, using computer vision for place recognition and distance measurement.

2.3 Computer Vision in Feature Detection and Matching

2.3.1 Scale-Invariant Feature Transform (SIFT)

Scale-Invariant Feature Transform was proposed by D. Lowe from University of British Columbia in 1999 [24]. It is used to detect and describe local features in images.

Image features are usually the corners or blobs in the image. Such features are widely used in object detection, place recognition, etc. In SIFT, for any object in the image, interesting points can be extracted as a feature description of the object. Then the feature description can be used to detect and locate the object in another test image which contains many other objects.

Basically there are four steps for a SIFT sequence.

1. Scale-space extrema detection

This stage of the filtering process attempts to identify locations and scales that can be repeatedly assigned under different views of the same object [22]. This can be achieved using a scale-space function based on Gaussian Function shown as below:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

In the function above, $*$ is the convolution operator, $G(x, y, \sigma)$ is a variable-scale Gaussian function and $I(x, y)$ is the input image.

Then a Difference-of-Gaussian function can be described as below:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

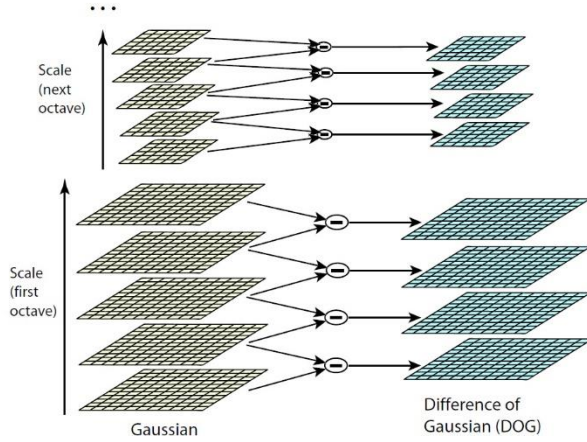


Figure 2. DOG Computing Process [22]

As shown in figure2 [23], each DOG is computed as the difference between two scaled images. So the initial image is repeatedly convolved with Gaussians to produce image in the left, and adjacent Gaussian images are subtracted to produce DOG images in the right.

Once the DOG images are produced, local extrema will be extracted by comparing each pixel with its 8 neighbors in the same scale and 9 neighbor pixels in the up and down scale images. If the value is maximum or minimum, it is an extrema.

2. Keypoint Localization

Due to the poorly defined peak in DOG function, it will introduce a strong response to long edges. And some of the edges detected are aligned by locations which can be poorly determined and unstable to noises [22]. Thus the candidate points extracted in step one will be dumped if they are poorly localized on an edge. For the same reason, points have low contrast will also be rejected.

For each point (x, y, σ) we got in step 1, \hat{x} is defined and calculated as:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2} \frac{\partial D}{\partial x}$$

If \hat{x} is below a certain threshold, as 0.03 as in [23], the keypoint will be rejected for low contrast.

For edge-located keypoints, a principal curvature will be computed at its location and scale using a 2x2 Hessian matrix. Since we already know the eigenvalue of Hessian matrix are proportional to principal curvatures of D, we can use the magnitude ratio between the largest and a smaller one to setup a criterion. If the ratio is below a certain threshold, the keypoint will be rejected.

3. Orientation Assignment

Orientation will be assigned to each keypoint in this step to make them invariable to image rotations. So for each keypoint, gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ will be calculated using pixel differences [22]:

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y + 1) - L(x, y - 1))/(L(x + 1, y) - L(x - 1, y)))$$

An orientation histogram will be formed from the gradient orientations of sample points within a region around the keypoint. Then it use the highest peak of the histogram with any other local peak within 80% of the height of this peak to create a keypoint with that orientation.

4. Keypoint descriptor

Now keypoint descriptors are created, a 16 by16 neighborhood around keypoint is taken. It is divided into 16 sub-blocks with 4 by 4 size. For each block, 8 bin orientation histogram is created. So total 128bin are available for each keypoint. It is represented as a vector to form keypoint descriptor.

After the keypoints descriptor generated in two or more images, several matching algorithms could be applied to locate object or place of interest.

2.3.2 Oriented FAST and Rotated BRIEF (ORB)

The ORB algorithm was first brought up by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in their paper ‘ORB: An efficient alternative to SIFT or SURF’ in 2011. It is basically a fusion of FAST keypoint detector and BRIEF descriptor with many modifications to enhance the performance [24].

FAST is short for Features from Accelerated Segment Test. Developed by Edward Rosten and Tom Drummond, the algorithm is used for high-speed corner detection. Consider the following image with pixels enlarged:

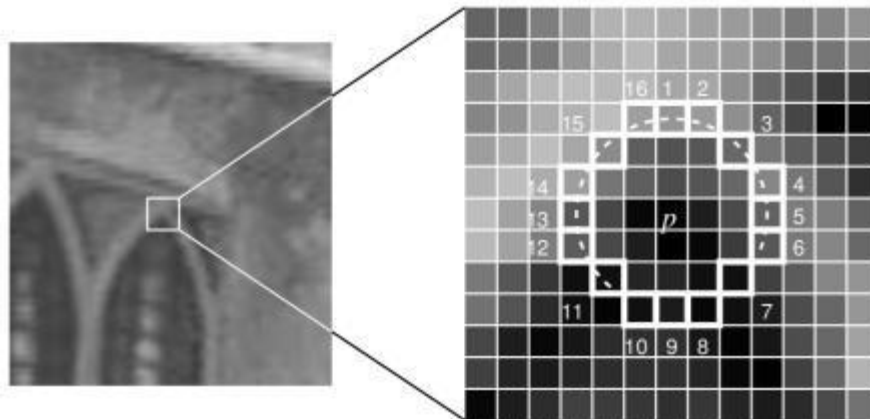


Figure 3. FAST corner detection [25]

For each pixel P , we calculate its intensity as I_p . A threshold for intensity will be chosen as t to test four pixels: 1, 5, 9 and 13. If more than two of their intensity values is all larger than $(I_p + t)$ or all smaller than $(I_p - t)$, then test the 16 pixel set around pixel P . if there exist a set with number of N pixels, whose intensity values all larger or smaller than the threshold compare to I_p , then P is a corner. [25]

After corner keypoint detection in FAST, orientation value will be added to each point in ORB algorithm. It computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation.[24]

Then ORB uses a BRIEF descriptor for the keypoints. Binary Robust Independent Elementary Features, BRIEF, is proposed by Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua, as a more robust way to modify the feature descriptors to gain better CPU performance [26]. In ORB, for any feature set with N binary tests at pixel (x_p, y_p) , a matrix with size $(2 * N)$ will be defined. The matrix will contain the coordinates of the pixel set around (x_p, y_p) . Combined with orientation found before, the matrix will be rotated by a rotation matrix to form a new matrix, which will be used as ORB's descriptor.

For performance, according to the author in [25], ORB is much faster than SIFT and SURF and ORB's descriptor works better than SURF descriptor.

2.3.3 Feature Matching

There are several matching algorithms which can be used to match two data sets. In this thesis, two of them are used.

One is Brute-Force Matcher. Its principle is quite simple; it takes the descriptor from one data set and compares with all other feature descriptors in the second set to see if there is a similar match. Distance will be calculated to evaluate the similarity between two descriptors.

The other one is FLANN based matcher. FLANN is short for Fast Library for Approximate Nearest Neighbors [27]. Basically it performs a nearest neighbor search between two data sets [28]. When it comes to a large data set, FLANN based matcher could be a lot faster than Brute-Force Matcher [29].

2.3.4 OpenCV (Open Source Computer Vision)

OpenCV is a programming library mainly aimed at real-time computer vision processing [30]. It was originally developed by Intel research center in Russia, and now is maintained by Itseez. In the early days of OpenCV, the goals of this project were described as [31]:

1. Advance vision research by providing not only open-source but also optimized code for basic vision infrastructure.
2. Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readable and transferable.
3. Advance vision-based commercial applications by making portable, performance-optimized code available for free.

Now the library has been released for several versions with modifications, and earned users all around the world. In this thesis, most of the image processing work will be based on OpenCV library.

2.4 Hardware Specification

2.4.1 Hokuyo LIDAR

The Hokuyo URG-04lx-UGO1 is a dedicated laser range finder (LIDAR) used in many robot applications and with an accuracy ± 30 mm at a 5.6 m range. The sensor have approximately 240-degree field of view, but here in this thesis, when mounting on our lab's turtlebot, only front 180 degree will be used to gather range information for feature matching.

2.4.2 Logitech C920 Camera

Logitech C920 is a HD web camera with maximum definition of 1920*1080 pixels. Its focal length is 3.67mm, with sensor at 3.60mm in height and 4.80mm in width. The camera is used in this thesis to gather image information for place recognition and robot localization.

Chapter 3 Solution Implementation

As we could see in Figure 1, when using solutions based on LIDAR feature matching and probabilistic model, kidnapped recovery process may introduce a huge latency or even fail. The latency is caused not only by the matching process, but also by an exploring operation. The exploring operation, which is used to choose one most possible position among many possible candidates, sometime may not work even after it explored the whole map. Consider the following hallway map, where the gray area will be the rooms or wall, and white area is the hallway:

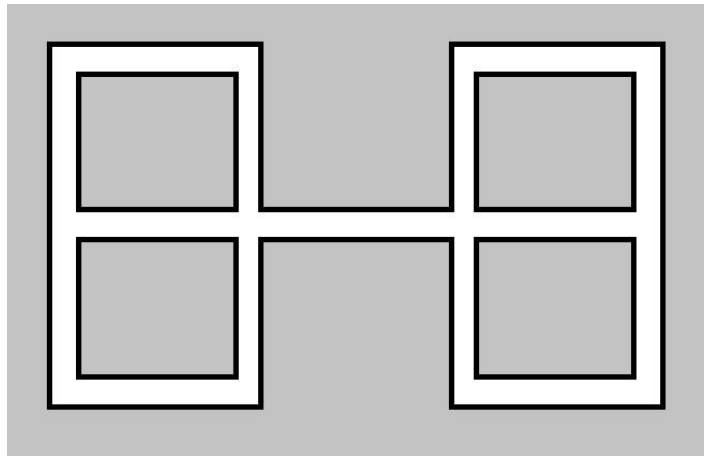


Figure 4. Hallway Map Example

In this map, if we use the solution above, the exploring process will always end up giving a twin candidates share the same possibility, making it impossible to autonomously recover its location. We all know that in the modern building layout, a symmetry room or hallway design is

widely used. This means the possibility of accruing a huge latency or recovery failure not only exist, but also could be very high.

To solve the robot kidnapped problem more efficiently, a solution should meet the following requirement:

1. First, the solution should be able to recover the position and direction information of the robot. And the information should accurate enough for a later task, like navigation through a map to another location.
2. Secondly, a solution should introduce as little latency as possible. So the algorithm should only use a short period of exploring time or none. Since the robot may still have a pending task to finish, it shouldn't spend a long time and distance in location recovery.
3. For a feature-based solution, the feature library used for matching should keep a small size and easy-maintained quality.
4. No solution will give a position with 100% percent accuracy, so the solution should be able to calculate the confidence value for its position estimation. Later in use, if the confidence is lower than a certain threshold, it can dump the result and set up another recovery process or report a failure.
5. A good solution should be transferable, which means it is not for a unique map. Once been slightly modified, it can be applied to another building or indoor environment.

To meet the requirement above, a feature based solution is implemented in the following sections.

This solution for robot kidnapped problem mainly contains two parts: place recognition, and position estimation. To recognize the place more efficiently, first, a limited number of fiducial markers will be used to mark the robot's operating area. Then, the robot can collect the place

information by recognizing the fiducial markers combined with the environmental objects around the marker. After this, a voting procedure is introduced to find out which place the robot is located and the confidence of the recognition result. Finally, a position estimation will be done using the marker and the LIDAR feature obtained.

In this thesis, the solution implementation is separated into three steps. So when applied to a certain building, a specific design could be formed step by step according to these steps.

3.1 Mark the Building

Since most buildings' indoor layout are using symmetric design, and these indoor places are sharing many similarities, adding fiducial markers will make a place more distinguishable, thus speeding up the recognize process.

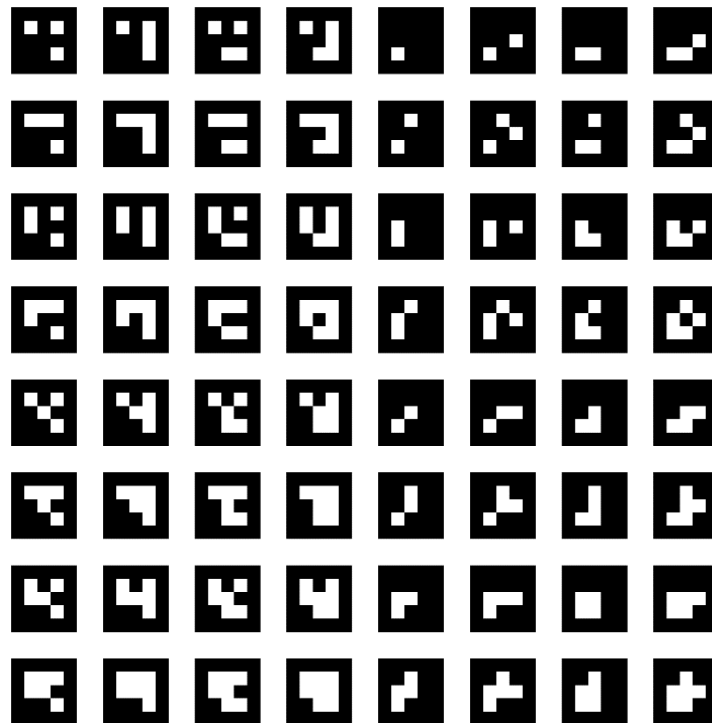


Figure 5. Fiducial Marker Example

But if we use a unique marker for each place the robot may operating at, it will be a very large marker library to use. To reduce the size and complexity of the marker library, the solution tries to follow the criterion below:

1. If the kidnapped robot does not need to move, it will take one or several pictures of the surrounding. In this scenario, the robot should see at least one marker in the pictures taken.
2. The fiducial marker combined with the objects nearby should be able to form a unique set for each place in the map.

So in this solution, the idea of a trigger marker will be introduced. As its name indicates, a trigger marker is used to initiate a recovery sequence. According to the criterion 1 above, the kidnapped robot may not take only one picture, instead it will take several pictures to find a marked place to begin the matching algorithm. So instead of matching every marker and object in every picture, checking only one or two triggers will save a huge amount of time. Usually the same trigger can be used in places sharing the same attribute, for example, places in the same floor, places are all rooms or all in the hallway, etc. So triggers can also be used to break the place candidates set down into several branches, with each branch using its own interesting objects library.

Another concept introduced here is the distance between two sets of interesting objects. As we know, for places inside a building, it is very hard to assign each place with a unique object set with unique elements which can't be found in any other place. The fact is that it is very likely to find several places sharing a similar object set, but different combinations, or only one or two object in their set are unique to other sets. To compare the similarity of two interesting object sets, a distance function is defined. Here, we use O_A and O_B for two object sets in place A and B.

NS_{AB} is the number of elements sharing between O_A and O_B , N_A and N_B is the number of elements in O_A and O_B . So the distance between O_A and O_B is defined as:

$$D(O_A, O_B) = 1 - \frac{NS_{AB}}{N_A + N_B - NS_{AB}}$$

It can be calculated that when ($NS_{AB} = 0$), which means place A and B share no interesting object, $D(O_A, O_B) = 1$. When A and B are holding the same object set, the distance will be zero. This calculation not only gives the similarity reference for two set, but also scale it into $[0,1]$, which will be easier to use later as performance reference for a feature library.

Another parameter introduced in this solution is confidence value. It is used to estimate the accuracy of a SIFT matching result. When FLANN based matcher in OpenCV is used, it will return a matched descriptor set with distance ratio for each pair of descriptors. Then a filter will be applied to dump a pair of descriptors if its distance ratio is less than a threshold. A value of 0.7 is recommended in OpenCV. After filtered, if the number of the good matches is larger than a certain number, a positive match result will be reported. Otherwise, a match failure result will be given by the program.

In the program, the number of well-matched descriptors and their distance ratio can be scaled into a confidence number. The lower the confidence is, the higher the possibility of a wrong match result it will be. Assume the distance ratio for each pair of descriptors is Dr_{match} with total matched pair number N_m , the total keypoints number in the quarry image, for example, a fiducial marker, is N_q . The confidence of a successful match is defined as:

$$Confidence_{match} = \left(r + (1 - r) * \frac{\sum_1^{N_m} Dr_{match}}{N_q} \right) * 100\%$$

The confidence of a failed match is defined as:

$$Confidence_{match} = \left(r + (1 - r) * \left(1 - \frac{\sum_1^{N_m} D r_{match}}{N_q} \right) \right) * 100\%$$

r is a scale parameter, in this thesis, r is 0.8.

In this chapter, an algorithm to mark the building hallway will be implemented first, then it could be slightly modified to mark the rooms. With the use of the triggers and distance, the following steps can be taken to mark a hallway map:

3.1.1 Find Out the FOV and Angle Limitation of the Robot's View

In this solution, the FOV is a range with boundaries as the longest and smallest distance at where the robot's camera could recognize the trigger marker and another objects. The distance could be determined by an object matching test, which involves setting up the camera, and matching a sample set of the objects and markers. If all the SIFT matching result could pass the confidence threshold, for example 85%, then the distance will be inside the FOV range. Then try to increase the distance, repeat the matching, until we have a distance long enough to apply in step 3.1.2, or the confidence is below the threshold. It will give us the upper boundary of the FOV range. Similarly, the lower boundary of the range can be found by decreasing the distance from camera to objects set.

To increase the FOV range, we can try to enlarge the size of the trigger marker. Since an ORB algorithm will be used in matching triggers, a bigger-sized trigger can increase the ORB's performance. But the size of the marker should not be too big. Since we can't increase the object's size, an oversized trigger can't give more confidence on object matching. Also it can be seen in the following section that a bigger trigger may introduce a larger distance error.

The angle limitation of the view is also required since the marker is not always facing directly to the camera. It means the angle formed by the marker plain and image plain when robot is facing directly to the marker direction. If the space indoor is too big, and the angle to too big, for example close to 90 degrees, the robot will not be able to recognize the marker no matter how many photos are taken.

3.1.2 Pick a Start Point of the Map and Start Marking Sequence

In this step, first we need to choose a start point, then a flood coverage sequence will be applied to pick place candidates around the map, mark the places and form a feature library.

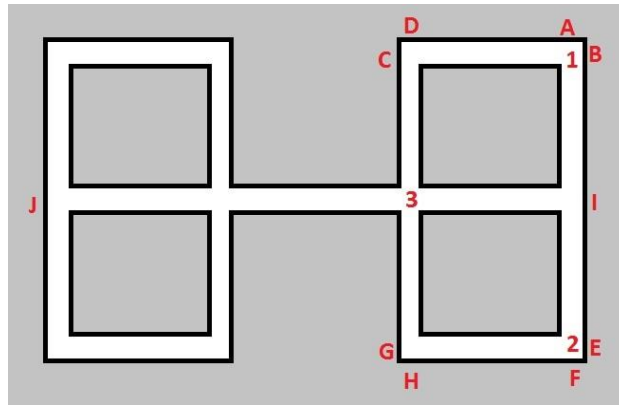


Figure 6. Hallway Marking Example

Here, the hallway example shown in Figure 4 will be used for demonstration. Each corner, which is formed by two walls, can be marked as below in Figure 6. Here, we use C_{AB} for the corner at place 1, as another four corners marked as C_{CD} , C_{EF} and C_{GH} . Similarly the hallway could be marked as H_{IJ} , H_{AF} etc. As for the starting point, the corner of the map, like place 1 or 2 is recommend. Because if a start point is chosen in the middle of the hall way, like place 3, the ‘flood’ will spread to multiple directions, and increase the complexity of calculation.

Each white pixel in the map, is a possible position for the robot. So we perform the flood sequence by changing every pixel in the white area with the RGB value of (0,0,0) into black pixel with RGB value of (255,255,255). The sequence to change all the pixels from black to white is like a flood map coverage, so it is called flood marking sequence. Thus when there is no pixel in white area, we will know the marking work has covered all possible positions of the robot in map.

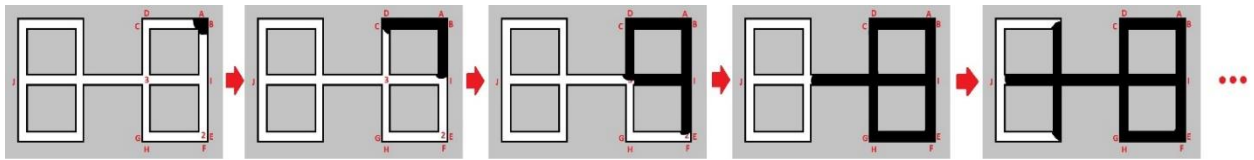


Figure 7. Hallway Marking Flood Sequence

In Figure 6, it can be seen that each corner or hallway will propose two place candidate. For instance, in corner AB, a marker can be put either on A or B. Now, let's start the flood sequence. Note that in the map, the gray area means the unknown places, while black lines mark the boundary of the hallway. The flooding procedure is shown in Figure 7.

When we first start the flood sequence in map shown in Figure 6 at position 1, we can simply choose a place nearby inside the FOV. Here we choose A and put a trigger marker. Then, we choose the objects beside the trigger, which is used to form a set of interesting objects for place A.

Then, for every new pixel been flooded through, we do the following test. Firstly, within the FOV and angle criterion, does there already exist a place with a trigger marker? If the answer is yes, then move on to the next position. If the answer is no, then we check the hallway in which the robot is located. As in the Figure 6, we begin in position 1, the first set of flooded positions

should be able to locate themselves at hallway H_{BC} and H_{AF} , the hallway will help to locate four candidate places, A,B,C and F.

Consider position a, b, c and d in Figure 8, let's mark locate them one by one. For position b, check the FOV first. Because it is too close to A and B, assume it can't recognize the marker in place A, and also won't recognize the possible marker in B. So we are left with place candidates

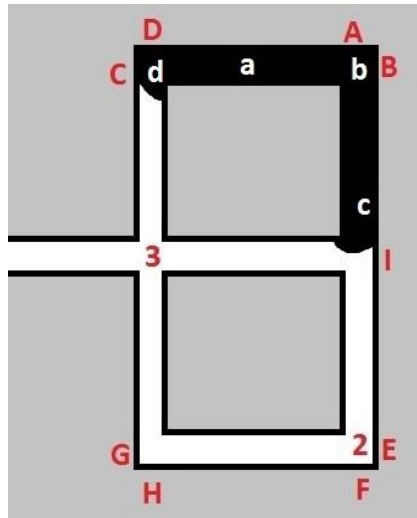


Figure 8. Hallway Marker Assignment

C and F. Then we consider the FOV again, is place F covered in FOV when the robot is at position b? If not, candidate F will be dumped. The same test will be done on place C. If both answers are yes, then an interesting object set will be formed for C and F. Then the candidate set, which is O_C and O_F , will be used to calculate Hausdorff distance to every set already existing in the library. In this case, it will be set O_A . For each distance calculated, it will be first compared to a threshold. If the distance is below the threshold, the new candidate set will be rejected. If more than one candidate set passes the threshold test, the total distance (dissimilarity) will be calculated as:

$$TotalDistance_{candidate} = \sum_1^{Size(library)} D(O_{candidate}, O_{library})$$

After the calculation, candidate holding the least total distance value will be chosen as a new member in the feature library.

In the same way, the place is chosen for position a, c and d. One situation that may happen is that the FOV of a robot is so limited, that it can't detect any hallway end. For example, in position c, if a robot's FOV cannot reach place A and F, the algorithm will fail. In this situation, the first thing to do is trying to increase the FOV. This could be done by using a camera with higher resolution or enlarging the trigger's size. If the FOV still can't reach any hallway end, we can hang the trigger image under the ceiling to solve the problem.

When feature library comes into a large scale, another situation may occur that for one position, all its place candidates can't pass the distance threshold test. This could also happen when each candidate for a place holds an exactly same object set as another set in library. In this case, a fiducial marker will be added to each candidate place. The distance test will be performed all over again too choose the most suitable candidate. If the threshold still can't be reached, we could add a second marker, a third, and so on.

During the process above, more than one fiducial marker may be added to the library to satisfy the distance test. When this happens, every next time we need to add a fiducial marker, we first pick from the markers which are already been used before. Only if they all fail the distance test will we add a new fiducial marker to the library. This principle will help to reduce the total fiducial marker used.

The main marking sequence could also be described as a flow chart shown in Figure 9.

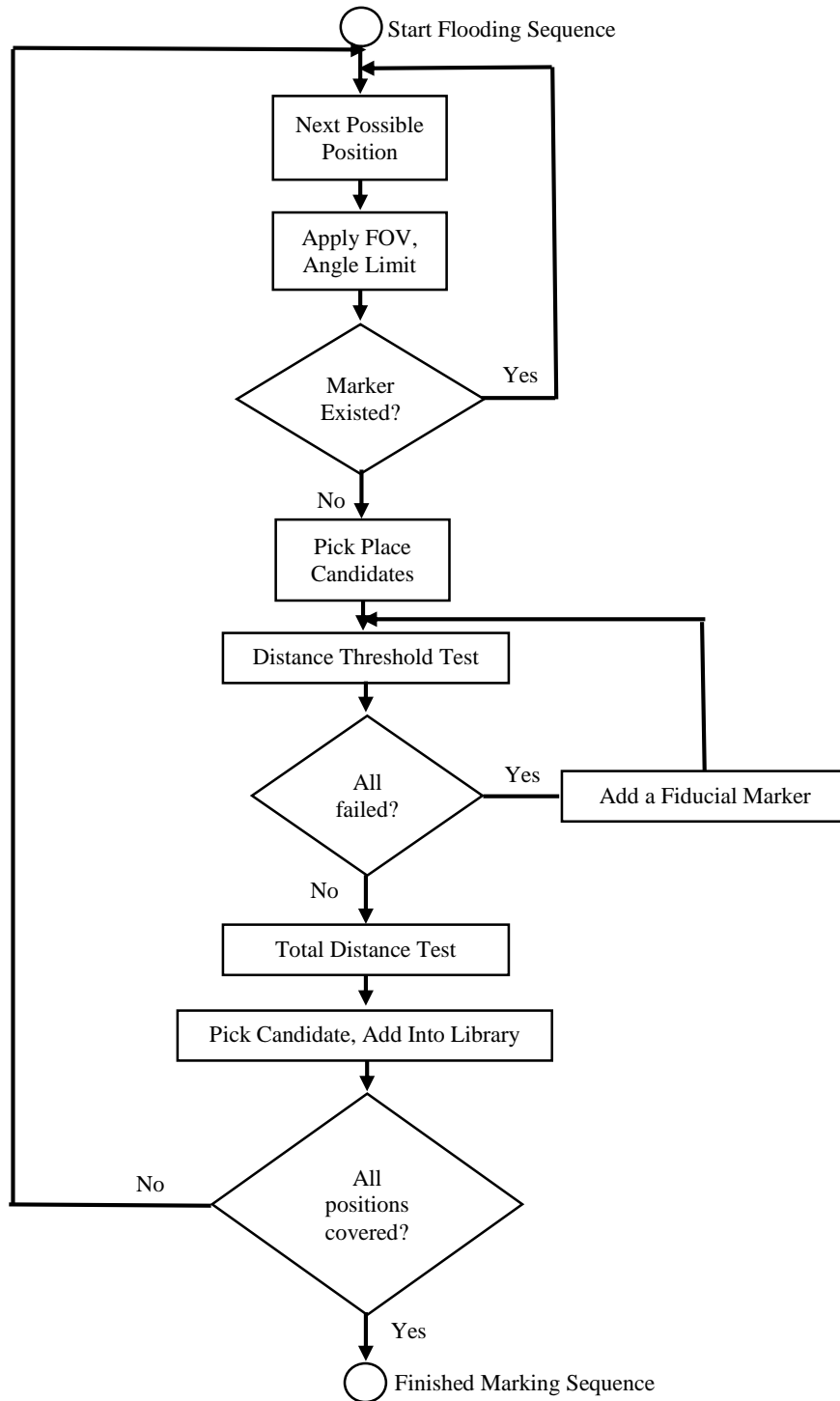


Figure 9. Main Marking Sequence Flow Chart

Another fact that needs to be mentioned is that different start points for flooding sequence may give different libraries. Because different places may be chosen, different marker allocation may be used. Here, the performance of a library can be evaluated by calculating average distance between each pair of place sets in it.

3.1.3 Form and Maintain a Feature Library

For each place newly added to the library, it will be numbered, as well as the objects in the new set. So a following table could be formed to represent the library. ‘1’ means the object is included in the set of a place, while ‘0’ means not included. The column shows that for each place, which object is included in its set. And the row could tell us for each object, which place it belongs to.

	Place A	Place C	Place B	Place F	Place G	Place I
Trigger 1	1	1	1	1	1	1
Object 1	1	1	0	0	0	1
Object 2	1	0	1	1	0	1
Object 3	0	1	1	0	1	0
Object 4	0	0	1	0	1	1
Object 5	0	0	0	1	0	1

Table 1. Feature Library

Table 1 shows one possible marking result for the right wing of the hallway in Figure 6. Using places as columns and object as rows, the computer could simply keep the library as a matrix. To maintain a library in operation, for example, if a new object is added into one or several existed

set, we can add a new row in the matrix. Similarly each time we add a new place, a new column will be added into the matrix.

3.1.4 Mark the Rooms If Needed

The steps to mark the rooms will be similar to the steps to mark the hallway. Choose one room to start, or use room number to mark them one by one. For a room with small size and non-blocked space, only one trigger is needed. If a room is too big to be reached by the robot FOV, place candidates will be picked similar like the hallway, and proceed with the distance test.

Also, a feature library will be formed. Here, if the trigger used in the room is the same as the hallway, we can add these two libraries together into one matrix. If not, it is recommended to store them in different matrices to improve the query performance later.

A major difference for marking the rooms is for every trigger, another marker will be assigned beside them with a certain distance. This marker will be used for every room and will not change with the triggers, meaning all the room could share a same marker. The latter marker will not be in the library. The marker is added for position estimation, which will be illustrated in Section 3.3.3.

3.2 Place Recognition

Once we finished marking the map, the feature library matrix can be imported into the recovery program in a robot's computer, and the kidnapped robot recovery process can start.

When the robot is kidnapped or powered-on, it will start a recovery program. First a picture will be taken by the onboard camera. The picture will be matched with the trigger marker using ORB detector. If there is no match report, the robot will rotate a certain angle to take another

picture. The procedure will be repeated until there is a match report. Although the ORB detector is not good at precision compared with SIFT, it can still provide a reliable match report with speed several times faster than SIFT.

Once the trigger is recognized, the robot will know that the place it is looking at is one of the places in the feature library. When several triggers are used, the robot will also know which feature library the place belongs to. Then the next step to take is to decide which place it is. To achieve this, robot needs to match the picture with objects in the feature library. Since every place in the library holds a unique combination of objects, a unique result should be reported after a full library check. Due to the precision quality of the SIFT detector, it is used in the feature library matching. So when a trigger marker is detected, the algorithm will take the object one by one from the library to match with the picture using SIFT. SIFT matching will report two kinds of result, either a successful match along with a confidence ratio, or a match failure with a confidence ratio. But SIFT is much slower than the ORB algorithm. If we run the test for the whole feature library, it will introduce a huge amount of latency, especially when we have a large number of place candidates. To speed up the process, a voting process based on fuzzy logic will be performed right after every object match test.

First let's consider a voting with result in true or false. When an object is in the picture, we vote for true. It means each place holding a set including this object will gain one vote, while each place without this object will lose one vote. The vote cannot be less than zero. When the object is not in the picture, any place holding a set with this object inside will lose one vote. Note that no set will gain a positive vote in this situation, because it will reduce the accuracy of the estimation later.

But when SIFT algorithm reports a match, it won't be a 100% match or match failure. Instead a confidence number in percentage will be reported, telling us the probability of the object existing or not existing in the picture. Here, we use R_c to represent the ratio of confidence. Since we know $0 \leq R_c \leq 1$, instead of voting 1 or 0, we can split the vote into R_c and $(1 - R_c)$. For each object that reports a match with confidence of R_c , we assign R_c v for true, and $(1 - R_c)$ for false. Thus a place with this object will receive $(2 * R_c - 1)$ vote in total, and a place without this object will receive $(-R_c)$ vote. Similarly, for the failure report with R_c , place with the object will receive $(1 - 2 * R_c)$ vote, while place without the object will receive $(R_c - 1)$ vote.

After each object matching, we will update the vote for each place, and then calculate the weighting number for each place's vote in the total vote. If one place got a leading vote, and it is the only one in the leading position, we will break the matching loop and report a successful recognition. A threshold test can also be introduced here. It is an optional test, when applied to the voting, the winning place not only need to have a leading vote, but also a weighting above threshold. The threshold can force the matching algorithm to match for more object than needed, thus to increase the accuracy of the recognition.

Here, let's demonstrate the voting process using feature library in Table 1. Assume the robot is kidnapped in the hallway showed in Figure 8, at position c, facing place A. The confidence ratio for each detection is 0.9.

After the test for object 1, program will report a match, and voting result is shown in Table 2:

	Place A	Place C	Place B	Place F	Place G	Place I
vote	0.8	0.8	0	0	0	0.8
weighting	0.3333	0.3333	0	0	0	0.3333

Table 2. Voting after object 1

After the test for object 2, the program will report a match, with result in Table 3:

	Place A	Place C	Place B	Place F	Place G	Place I
vote	1.6	0	0.8	0.8	0	1.6
weighting	0.3333	0	0.1667	0.1667	0	0.3333

Table 3. Voting after object 2

After the test for object 3, program will report a match failure, with result in Table 4:

	Place A	Place C	Place B	Place F	Place G	Place I
vote	1.5	0	0	0.7	0	1.5
weighting	0.4054	0	0	0.1892	0	0.4054

Table 4 Voting after object 3

After the test for object 4, program will report a match failure, with result in Table 5:

	Place A	Place C	Place B	Place F	Place G	Place I
vote	1.4	0	0	0.6	0	0.7
weighting	0.5185	0	0	0.2222	0	0.02593

Table 5. Voting after object 4

After this round of voting, we can see place A holds a leading vote weighting number as 0.5185. So we can call an end to the matching process now. In this case, the voting process will reduce the time cost by skipping the matching operation on the object 5. However, if the program do the match for more objects, it will increase the accuracy of the voting result. This can be done by setting a weighting threshold larger than 0.5185. So the algorithm will take object 5 for match.

After the test for object 5, program will report a match failure, with result in Table 6:

	Place A	Place C	Place B	Place F	Place G	Place I
vote	1.3	0	0	0	0	0
weighting	1	0	0	0	0	0

Table 6. Voting After object 5

After this round, the weighting number is 1, which is above the threshold for sure, we can break the further possible loop and report a recognition.

However, the possibility may exist that with all object matched, the leading weighting number is still below the threshold. This means the threshold setting has a flaw. This could be solved either by using a lower threshold, or by adding more markers to increase the distance between two similar sets.

3.3 Position Estimation

Once a robot has recognized the place it is facing, the next step is to get the position information against a known map. Here, the distance from the robot to the trigger marker will be calculated first. Then the position of the robot will be estimated by one of the two algorithms depending on which kind of place it is located: a room, or a hallway.

3.3.1 Distance Estimation

In a sample camera projection model shown in Figure 10, distance d is the distance between fiducial marker and camera projection center along z axis.

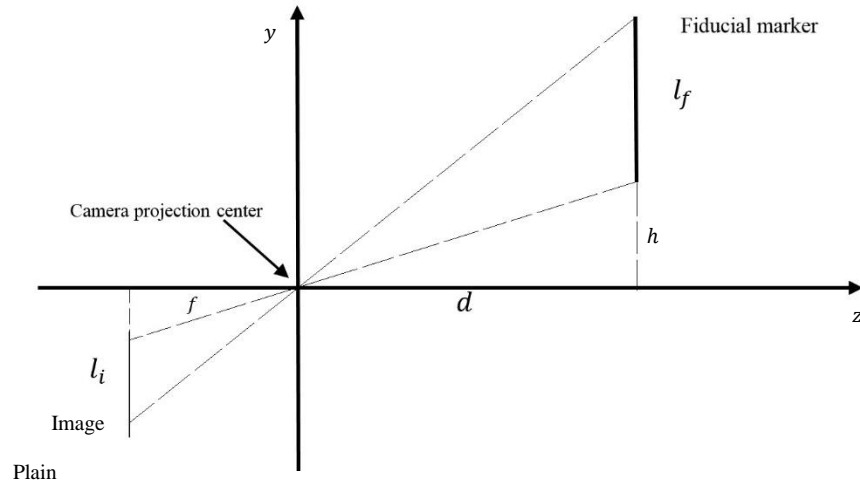


Figure 10. Camera projection model y-z plane

In the model above, the camera's y axis and fiducial marker is always perpendicular to the ground, and the fiducial marker's center is located at y-z plain. Assume the height of the fiducial marker is l_f , and it is l_i when projected into the image plain. When the focal distance is f , we have:

$$\frac{d}{f} = \frac{l_f}{l_i}$$

But the marker's center is not always in the y-z plain of the camera coordinate. When it is not, as shown in Figure 11 at camera's x-z plain, the projection triangles in Figure 10 will become a line with d and f' . Assume the line will form an angle of ϑ . It is not hard to see that d can be calculated as below:

$$d = \frac{l_f}{l_i} * \frac{f}{\cos \vartheta}$$

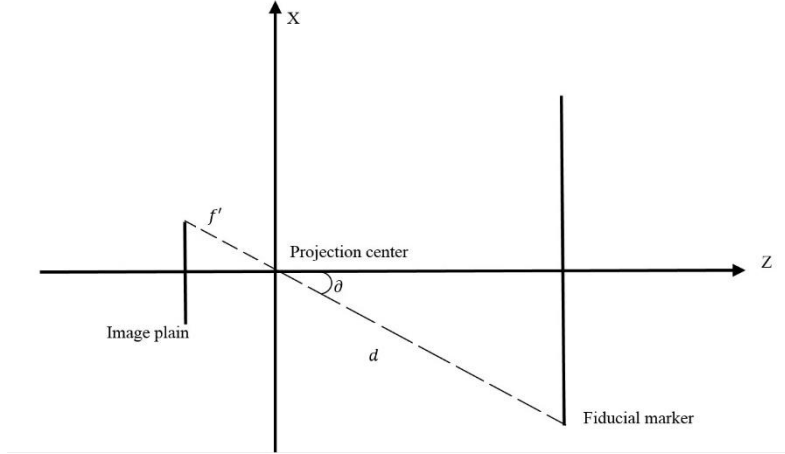


Figure 11 Camera projection model x-z plain

Since we know that l_f is proportional to the pixel number P_{fid} in the fiducial marker image file, and l_i is proportional to the pixel number P_{cam} of the marker in camera image, the distance can be calculated as

$$d = r * \frac{P_{fid}}{P_{cam} * \cos \theta}$$

In the equation above, r is a parameter that can be estimated by experiment. So we don't have to know the camera technical parameter when camera is changed. For θ estimation, we can use the Taylor series. In figure 11, we can see θ is not proportional to the marker-to-image-center offset, but $\tan \theta$ does. For a camera with limited FOV, for example less than $[-30,30]$ degrees, it is accurate enough to estimate θ in $[0,30]$ or $[-30,0]$ using two stage Taylor series. The parameter of the Taylor series can be calculated by sample points in camera image with θ number manually measured.

When applied to the computer program, SIFT detector will be used to find a pair of matched points between the fiducial marker and the camera image, then calculate their coordinate

difference along camera's y axis to solve the distance. To pick a good pair of matched point, here, two filters will be used.

The first filter is used to reduce the error of distance estimation. During the estimation, error is introduced mainly from two sources. The first one is the estimation of l_i . l_i is calculated by the number of pixels. Since the number is a discrete integer value, l_i will always come with a certain error. However, the error can be reduced by increasing the number of total pixels along y axis. It means to pick a pair of the matched keypoints with the biggest pixel distance along y axis. The second source of the error is caused by the perspective transformation. A pair of points reported may have a difference along x coordinate of the image plain and the marker plain. So when the points have been transformed from maker plane into image plane, this difference will cause a change in their distance along image y axis. Assume the lower point is in the height of h in Figure 10. Then the z axis forms an acute angle with the marker plane as ∂ , and the distance of the points in the fiducial marker in image x coordinate is Δx . According to [32], the ratio of the error to length l_f as r_{error} can be given in:

$$r_{error} = \frac{h\Delta x \cos \partial}{dl_f}$$

So to reduce this error, we can lower the position of the fiducial marker and choose the points will smaller Δx . Usually for each fiducial marker, there will be about 3 to 16 points reported as matches. So for each pair of the points coordinates, we set up a criterion as C , calculated as:

$$C = \frac{\Delta x}{l_i^2}$$

The pair of points with the smallest C value will be chosen to estimate the distance.

The second filter is used to eliminate a wrong match. During the recognition process, there is a possibility that the SIFT detector will report a match with the point outside the fiducial marker. The wrong-match point can easily have a leading position in the first filter's test, cause an estimation failure.

During the test, observation reveals that wherever the wrong-matched point is, it is usually located outside the fiducial marker's keypoints cluster. If we calculate the geometric center of all matched points and its distance to each point, it can be seen that the wrong-matched point usually have the longest distance to the geometric center. So here, an average distance to the center will be calculated, then a ratio between distance of one point to average distance can be found. If the ratio exceeds a certain threshold, it will be dumped before the first filter taken into action.

With the distance calculated, position can be found according to the type of the place. There stands two possibilities: hallway, or a room.

3.3.2 When the Place Is a Hallway

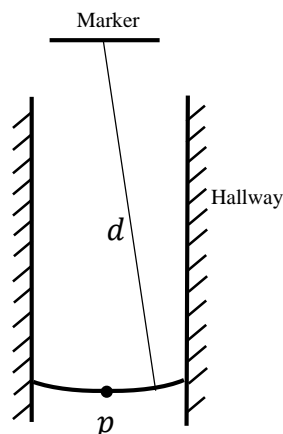


Figure 12. Hallway Position Illustration

When the place is a hallway, due to the angle and FOV we got in 3.1.1, the robot position can always be located at an arc shown in figure 12.

Since the place with the marker can be found in the map with a coordinate reference, the center of the arc can be calculated as p in figure 12. Then position will then be input into the AMCL node in ROS. Based on LIDAR feature matching using the wall of the hallway, AMCL will publish a position estimation using TF publisher in ROS.

3.3.3 When the Place Is a Room

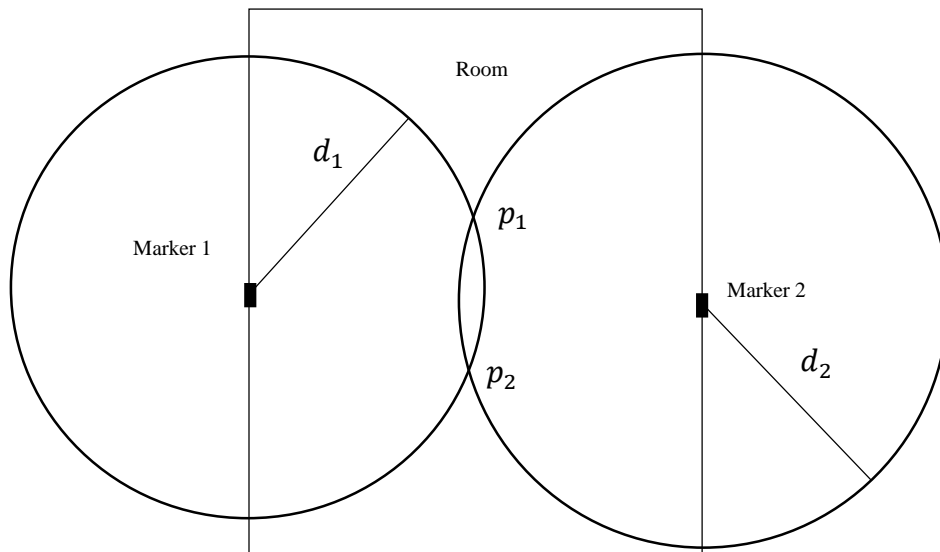


Figure 13. Room Position Illustration

When the place is in a room. The LIDAR features may have a confusing layout because of the indoor objects. So here a second marker is been used to estimate the position. The marker is located with certain distance or angle with the trigger marker. For accuracy consideration, it is recommended to put this marker in a place where the robot camera can't acquire it in the image frame with the trigger. So after the robot recognize the room and the marker, it needs to rotate a

certain angle to find the second marker and its distance information, similar as the process to locate the trigger. When both markers' distance are calculated, program can find two possible positions shown in figure 13.

An extra position can be dumped according to the rotation of the camera. For example if the camera is been rotated counterclockwise and marker 2 is in the first frame, marker 1 is in one of the followed frames, then p_2 is the robot position, p_1 will be dumped. The heading direction of the robot can be estimated by the second marker's position in the camera frame. Then a full position estimation can be published in ROS topics.

3.4 Error Prediction

3.4.1 Error in Place Recognition

For place recognition, error accrues when a wrong place is recognized and reported. In this part, the weighting number of the final vote can also be used to calculate a confidence reference. For place A in library, assume we do a voting with all matching result at 100% confidence, a winning vote weighting can be calculated as V_{binary} . If place A is the winner in a voting process, with the final vote weighting V_p , the confidence for the recognition is defined as:

$$confidence_{place} = \frac{V_p}{V_{binary}} * 100\%$$

This confidence can be used to evaluate the possibility of a wrong recognition result reported.

3.4.2 Error in Position Estimation

For position estimation. When in the hallway, error will come from both distance calculation and AMCL feature matching. AMCL in ROS will usually give a result with fix covariance for

reference. The error in distance calculation could be estimated use a normal distribution as below, with distance value as mean value μ , and σ given through a distance test. The distance test program will take a known marker, calculate the distance for hundreds of times, and calculate σ against a known distance.

$$f(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

When the robot is in the hallway, we can estimate error in a similar way as above. With two distance in two normal distribution sharing same σ , a bivariate normal distribution can be described as follow [34]

$$f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-\mu_x)^2 + (y-\mu_y)^2}{4\sigma^4}}$$

Basically it will locate a circle of possible positions with distribution similar to figure 14.

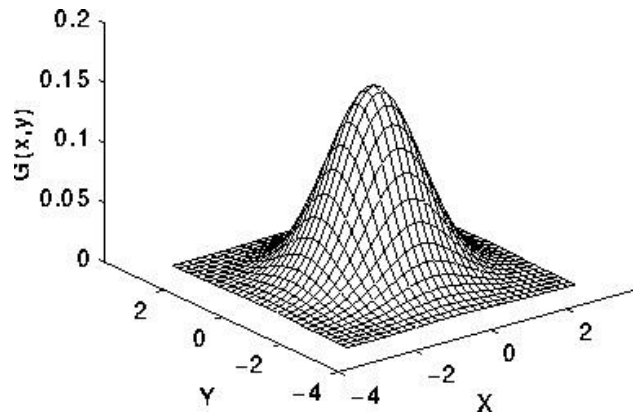


Figure 14. Bivariate Normal Distribution

Chapter 4 Experiment Setup and Results

In this thesis, the west wing of Auburn University Broun Hall Third floor along with room 368 will be used to form a map for test. First, we use a flood sequence to mark the map and form up a feature library. Then test the recovery program at multiple positions to see the result and accuracy of the recovery solution.

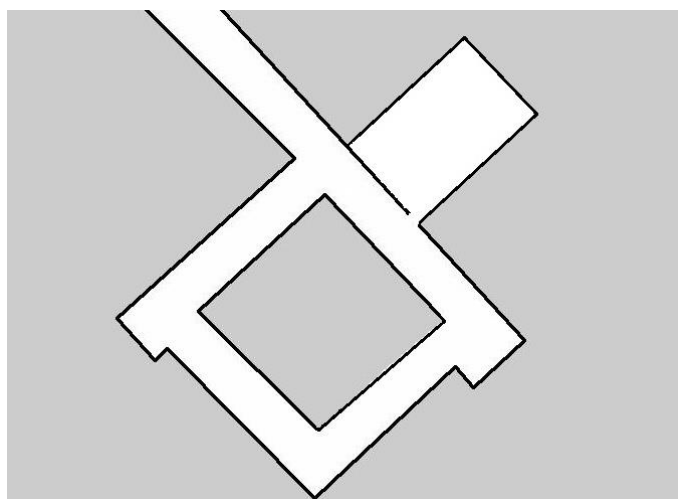


Figure 15. Broun Hall Third Floor West Wing Test Map

Since the kidnapped recovery sequence only need a camera and a LIDAR, we don't need to test the robot in a fully functional mode. So the robot will be remotely controlled, move to a certain position of the map, and a recovery program will be started to test the performance. The rotation of the robot can be achieved by remote control or manually, both of them won't affect the recovery sequence.

Before the camera is mounted on the robot, it will be calibrated for radial and tangential distortions. It can be done by using the functions provided in OpenCV based on an algorithm proposed in [36].

With a calibrated camera. First the FOV and camera angle will be calculated, then the map will be marked with a feature library in 4.1. With a marked map, the recovery result will be given in 4.2. Finally, the performance of the system will be analyzed in 4.3.

4.1 Marking the Map

The first step is to measure the FOV and angle limitation of the camera. The FOV can be measured by a trigger maker. In this case, the fiducial marker shown in the figure 16 is been used. The marker is chosen as the trigger because it has a good height-to-width ratio, and a good size of the keypoint set in SIFT, which will help reduce the error in distance calculation later.

In this test, for every frame taken by the camera, it will be matched with the marker in figure 16 using ORB detector. A certain threshold of the matched descriptors will be set. If number of

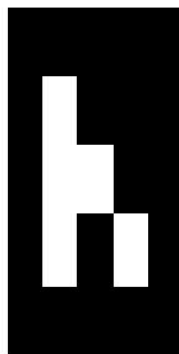


Figure 16. Trigger Marker Used In Experiment

the matched descriptors exceeds the threshold, it will be shown as follow in picture 15, means we have a match.

The FOV will be reported as a range of the distance. The lower boundary is where the camera will lose the fiducial marker and other objects in the image. The Higher boundary is where the ORB detector can't report a match when facing directly to the trigger marker. After tested, the Logitech C920 camera with trigger marker in figure 16 got a FOV from about 2 meters to approximately 14 meters in the hallway, 1 to 14 meters in a room, and the angle of the FOV is about 60 degrees. The FOV in the room will have a lower boundary to the FOV in a hallway, because the objects in the rooms are mostly chairs, desks, vase or bookshelf, which are in a lower height level compare to the posters and fire alarms in the hallway.

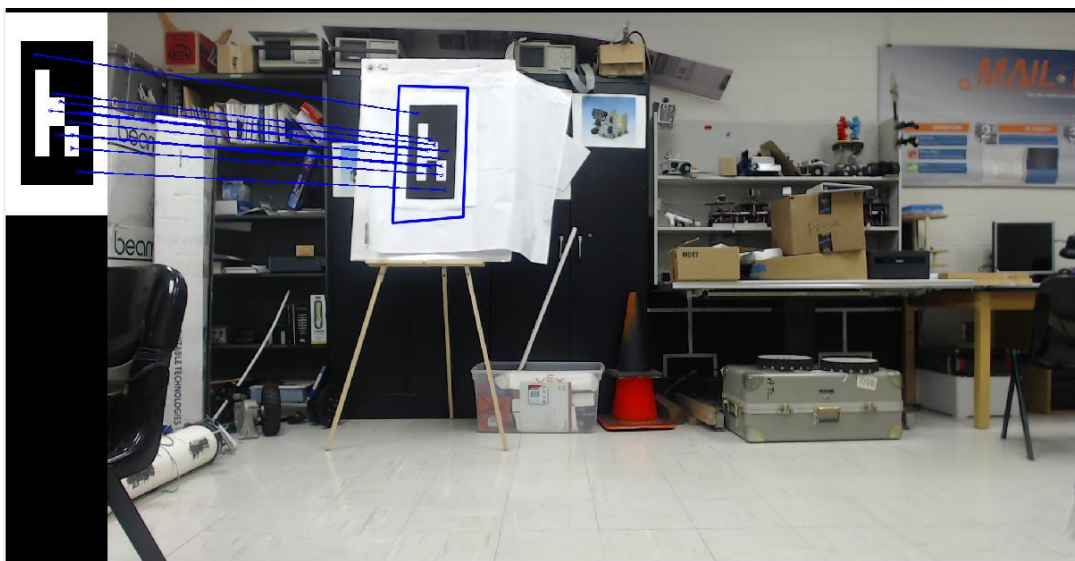


Figure 17. FOV Test Matching

Now, we can start marking the map in figure 15. First, all the place candidates will be marked as in figure 18. Since the size of room 368 shows in the map is a small one, it will be marked as one place. The number of the place candidates in the map use is below 26, so they will be marked alphabetically to help the illustration. For same reason, sample positions will be marked

form 1 to 6. In this map, distance threshold between two sets in feature library is 0.6. So the least distance between two places in library is 0.6.

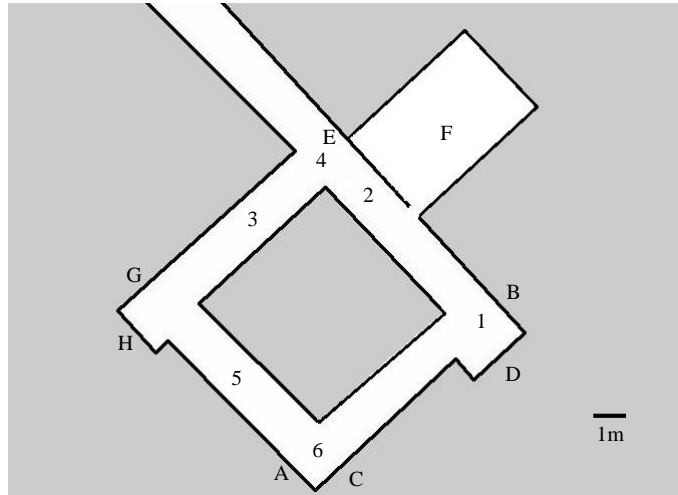


Figure 18. Place Candidates in Map

Here, we choose the bottom left corner in figure 18, which is corner C_{BD} to start the flooding sequence. So the first place to added in the library will be place A, since at position 1, near C_{BD} , marker at BD is out of the range in FOV.

Now the feature library will be like the table below, after we choose the objects. Here, a fiducial marker will be used because place A only have one object, which gives us a low distance number when adding another place candidate.

	Place A
Trigger marker	1
Fiducial marker 1	1
Fire alarm	1

Table 7. Feature Lib after Place A Added

With the flood sequence going on to position 2 and 6, B and D will be marked, with the library updated as table 8.

	Place A	Place B	Place D
Trigger marker	1	1	1
Fiducial marker 1	1	0	0
Fire alarm	1	1	0
Fire extinguisher	0	1	0
Door	0	1	1
Poster 1	0	0	1

Table 8. Feature Lib after Place B, D Added

When position 3 is flooded, two new candidates will be reported as place E and H. Place E have fire alarm and a poster different from poster 1, while place H have a door and a fire alarm. In this condition, distance will be calculated as table 9.

	Distance to A	Distance to B	Distance to D	Total
Place E	0.75	0.6	0.8	2.15
Place H	0.6667	0.5	0.75	1.9167

Table 9. Distance Test for Candidate E and H

As can be seen in table 9, place H not only failed the threshold test, but also hold a lower total distance. So here, place E will be added into the feature library.

When position 5 is flooded, place C and G will be chosen as candidate. C have a door in object set, while G have only a fire alarm. It is not hard to see that they will both fail the distance

test. So fiducial marker 1 is added to C and G for a new round of distance test. The result is G fail the distance test, while C passed all the test. So C is added to the library.

After the hallway been marked, place F will be added as a room place candidate. So when the marking processes been finished, the library will be as table 10.

	Place A	Place B	Place C	Place D	Place E	Place F
Trigger marker	1	1	1	1	1	1
Fiducial marker 1	1	0	1	0	0	0
Fire alarm	1	1	0	0	1	0
Fire extinguisher	0	1	0	0	0	0
Door	0	1	1	1	0	1
Poster 1	0	0	0	1	0	0
Poster 2	0	0	0	0	1	0
Road Block	0	0	0	0	0	1
Chair 2	0	0	0	0	0	1
Turbot	0	0	0	0	0	1

Table 10 Final feature library

4.2 The Experiment Recovery System Implementation

This section is the initiation of the recovery system. Before start up the recovery system, the following parameter and data will be feed to recovery program or published around ROS to prepare for the recovery sequence.

1. The reference point of the trigger marker in each place. Each trigger's position in the map is recorded by saving its top left corner's coordinate in an array. It is used to calculate the robot's position in map coordinate.
2. The parameter used in distance calculation. First is the estimation of θ , as discussed before, a set of image point with offset and angle already measured is used to form a Taylor series. Then parameter r in the distance function can be estimated by measure a set of sample points with distance already know. With both θ and r settled, we can calculate the covariance of distance estimation by taken a large set of sample distance. After calculation, the covariance is $43.376mm$.
3. The map is provided by ROS map server, with a resolution at $0.05m/pixel$, under the frame named 'map'.
4. The LIDAR information is provided by ROS URG node. Using hokuyo LIDAR, it will publish scanned LIDAR features under the 'laser' frame in topic 'scan'.
5. Static transform publisher, to publish the transform form frame 'laser' all the way to 'odom', so the AMCL node can receive the message form hokuyo LIDAR and work functionally.

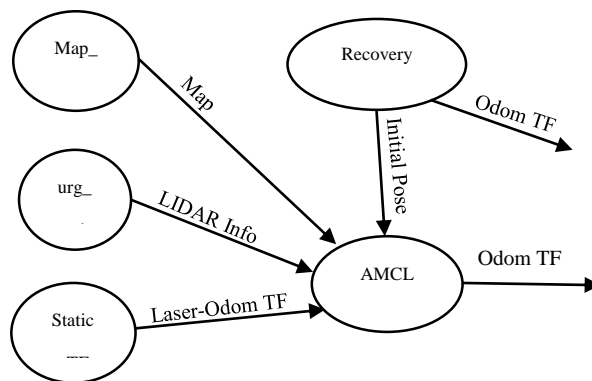
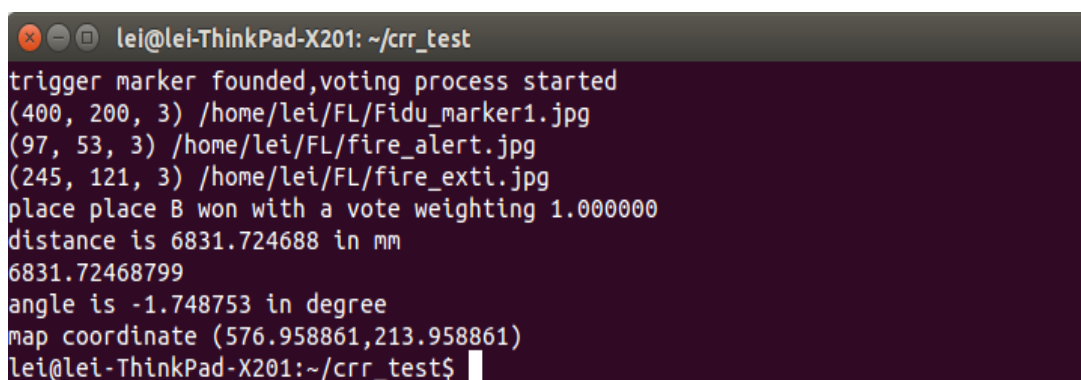


Figure 19. Message Flow in Recovery System

The message flow among the whole system is shown in figure 19. In this experiment, the input of the recovery program is the camera pictures, the output will be the initial pose of the robot published in the ROS topic, and a transform from map frame to 'Odom' frame. The second transform can be enabled optionally to replace the initial pose setting of the AMCL node. But the main function of the initial pose send by recovery node is a filter threshold, so if the AMCL

A terminal window with a dark background and light text. The window title is 'lei@lei-ThinkPad-X201: ~/crr_test'. The output text is as follows:

```
trigger marker founded,voting process started
(400, 200, 3) /home/lei/FL/Fidu_marker1.jpg
(97, 53, 3) /home/lei/FL/fire_alert.jpg
(245, 121, 3) /home/lei/FL/fire_exti.jpg
place place B won with a vote weighting 1.000000
distance is 6831.724688 in mm
6831.72468799
angle is -1.748753 in degree
map coordinate (576.958861,213.958861)
lei@lei-ThinkPad-X201:~/crr_test$
```

Figure 20. Terminal Window of the Recovery Node

report an initial pose too far from the one in recovery node, it will be filtered.

After a successful recovery, the recovery node will have an output in its terminal window as in figure 20. We can see that after the trigger been detected, a match-then-vote sequence will begin according to the feature library. The voting sequence stop when one place become the only leading winner among the place candidates. Then the distance will be calculated, followed by the initial pose in the map coordinate. After calculated by AMCL node, the final recovery result will be like figure 21. As we can see in the image section in the bottom left figure, text will be printed on the image showing the matched objects. The right part is the estimated pose array calculated by AMCL node. The array is shown as a set of arrows indicates the robot possible

positions. The black line is part of the hallway map, while the white line showing the laser feature in the surrounding environment.

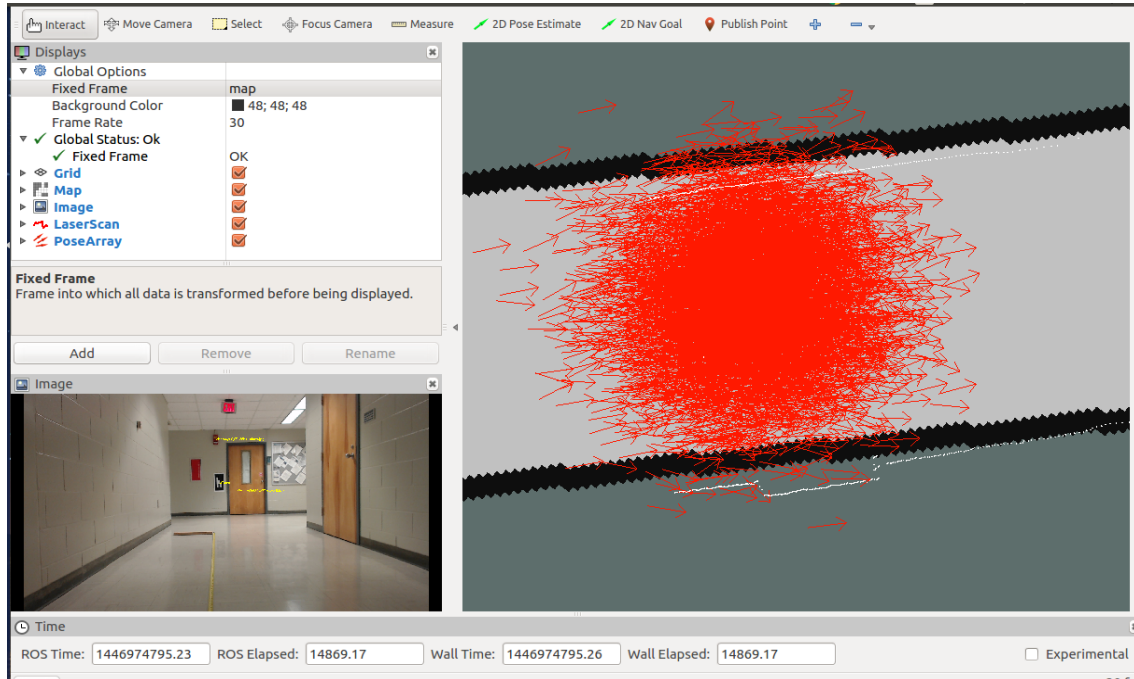


Figure 21. Recovery Result Showing in RVIZ

4.3 The Recovery Result and Analysis

4.3.1 Random Recovery Test

The first table below shows some random positions recovery results in the hallway. Here, the real coordinate is manually measured and calculated. Since the map been used have a resolution of 0.05m/pixel, and is drawn by a SLAM 'gmapping' node in ROS, error in the measured coordinates in the table may not be very accurate. But the map hold a distance error measured less than 1%, so the map and recovery result is still acceptable by ROS navigation stack.

Place	Vote weighting	Confidence	Distance/mm	Distance measured/mm	Angle Error/degree	Coordinate	Coordinate Measured	Coordinate Error/mm
A	0.5053	100%	5773.88	5760.0	1.52	(478.532, 107.642)	(481, 105)	176.78
A	0.4926	98.52%	7420.58	7442.0	2.51	(501.935, 130.230)	(498, 139)	490.43
A	0.4938	98.76%	8988.92	9000.0	1.22	(524.122, 153.125)	(527, 151)	176.14
B	1.0	100%	3429.02	3467.0	1.67	(525.109, 161.817)	(525, 162)	11.18
B	1.0	100%	6820.32	6806.0	1.75	(440.609, 80.709)	(438, 74)	359.34
B	1.0	100%	8046.75	8110.5	3.00	(463.17, 100.800)	(471, 94)	518.52
C	0.9077	90.77%	5401.23	5422.0	1.07	(357.618, 114.382)	(360, 115)	130.00
C	0.9062	90.62%	6650.39	6671.0	1.66	(339.95, 132.045)	(344, 134)	225.85
D	1.0	100%	5672.96	5702.0	0.89	(518.772, 229.214)	(522, 235)	340.66
D	1.0	100%	10099.56	10005.0	3.12	(456.460, 291.538)	(482, 302)	1380.71
D	1.0	100%	6740.22	6812.0	1.50	(478.532, 107.642)	(481, 105)	180.35
D	Failure	Failure	N/A	15000.0	N/A	N/A	N/A	N/A
E	1.0	100%	4229.89	4242.0	0.65	(370.746, 311.193)	(372, 313)	109.97
E	1.0	100%	6082.34	6100.0	1.27	(343.237, 284.987)	(343, 288)	150.47
E	1.0	100%	7829.85	7888.0	2.7	(319.281, 260.380)	(322, 268)	404.52
F	1.0	100%	3758.74	3757.0	1.5	(543.015, 384.458)	(542, 382)	132.68
F	1.0	100%	5055.97	5032	2.3	(528.467,363.877)	(527, 361)	161.47

Table 11 Random Recovery Test 1

From the results shown in table 11, we can see:

1. Within the FOV, all the places in the map can be recognized with acceptable confidence.
2. The distance measurement in table 11 have a convenience in 30.12mm. The maximum error in the test is 98.56mm at a distance about 10000mm, which is still below 1%.

3. The coordinates can be used to estimate the error distance in recovered position. The maximum error distance is 1.38m, with an average error at about 0.309m.
4. It can be seen that when the distance number is larger, the error in the distance measurement will also increase.
5. The recovery angle will be slightly larger when distance increase. But we can see all the angle errors are below 5 degrees.
6. Even with the trigger detected, the place may still fail the recognition, as the example shown in table 11. Although the trigger can be detected at 15 meters to start a voting process. The place still cannot be recognized with multiple match failures.

4.3.2 Experiment for Error Observation

To get more information about the system's performance, two experiments are set to observe the error in position estimation and recognition failure.

The first experiment is a recovery test at similar distances around one place. The recovery node give the position estimation in the center of the arc as shown in figure 12, where AMCL take this pose to give a more accurate estimation. Here, in this test, the robot is moved roughly along an arc like in figure 12 with different rotation angles. The recovery results are shown as table 13.

It can be seen that in the same position, increasing the angle of rotation will also increase the distance value. This is because the ϑ angle used in distance calculation is estimated by a two stage Taylor series. So when angle increased, the Taylor series approximation cannot well follow the ϑ value, thus introduced more error.

Place	Distance to the arc center/mm	Rotation along map x-axis	Distance/mm	Distance measured/mm	Angle Error/degree	Coordinate	Coordinate measured
B	5.0	0 degree	8097.85	8083.0	1.80	(461.915, 98.918)	(462, 98)
B	5.0	8.2 degree	8152.87	8083.0	2.8	(460.624, 97.618)	(462, 98)
B	212	0 degree	8060.87	8086.0	1.2	(462.034, 99.037)	(465, 95)
B	212	5.8 degree	8082.5	8086	2.1	(461.951, 98.932)	(465, 95)
B	543	0 degree	8063.02	8099.0	1.94	(462.023, 99.045)	(469, 91)
B	543	-7 degree	8072.0	8099.0	2.2	(461.988, 98.965)	(469, 91)
B	1035	0 degree	8032	8110.0	2.6	(463.786, 101.234)	(471, 94)
B	1035	5.7 degree	8046.75	8110.0	3.00	(463.17, 100.800)	(471, 94)

Table 13 Recovery test 3 result

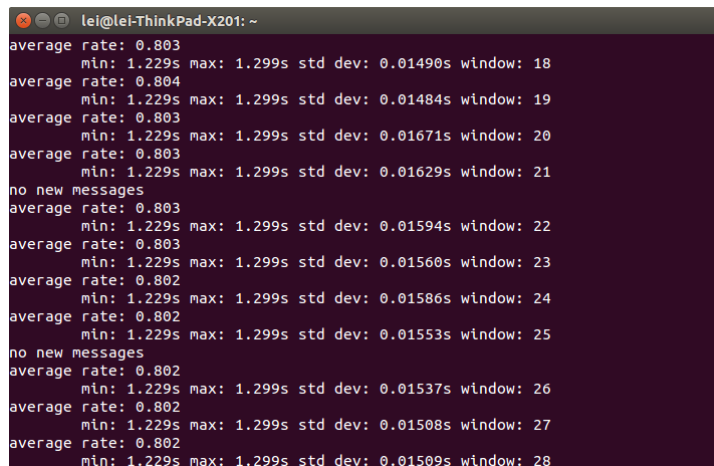
Another fact shown in table 13 is the distance from robot to the center line of the hallway will also affect the pose estimation result. The further the distance is, the larger the error will be. This is because the recovery node is using an arc for estimation. When robot is getting further from the center line of the hallway, it will still be treated as it was in the center. So the error will increase. Also we can see, the AMCL node cannot correct this error very effectively.

The second experiment been done is to test is how a wrong match will affect the recovery result. If the robot can't recognize the trigger, there is no doubt the recovery will fail. After the trigger been detected, the match process will go through the feature library. During the experiment in 4.3.1, it can be observed that when a place wins a voting test, some objects in its feature set may still remain untested. Assume place B is being tested, as shown in figure 20. After the matching test on fire extinguisher, which gives a positive result, place B will be recognized. So in this case, object 'door' is a redundant object.

The second experiment is implemented to recognize place A to E with one object covered by white paper. The experiment in 4.3.1 shows that place B is the only one among A to E who has a redundant object. And result in this test shows B is the only place who got a chance to be recognized with one covered object. It can also be proved by simulating a voting using feature library. As the result shows, with one object mismatched, a place with redundant object may still be recognized, while a place without redundant object will fail the recognition for sure.

Another fact need to mention is that during the test, with no object been blocked deliberately like above, place F is most likely to experience a recovery failure. Since the object in the room is been moved or blocked frequently.

4.3.3 The Speed of the Recovery Program



```
lei@lei-ThinkPad-X201: ~
average rate: 0.803
min: 1.229s max: 1.299s std dev: 0.01490s window: 18
average rate: 0.804
min: 1.229s max: 1.299s std dev: 0.01484s window: 19
average rate: 0.803
min: 1.229s max: 1.299s std dev: 0.01671s window: 20
average rate: 0.803
min: 1.229s max: 1.299s std dev: 0.01629s window: 21
no new messages
average rate: 0.803
min: 1.229s max: 1.299s std dev: 0.01594s window: 22
average rate: 0.803
min: 1.229s max: 1.299s std dev: 0.01560s window: 23
average rate: 0.802
min: 1.229s max: 1.299s std dev: 0.01586s window: 24
average rate: 0.802
min: 1.229s max: 1.299s std dev: 0.01553s window: 25
no new messages
average rate: 0.802
min: 1.229s max: 1.299s std dev: 0.01537s window: 26
average rate: 0.802
min: 1.229s max: 1.299s std dev: 0.01508s window: 27
average rate: 0.802
min: 1.229s max: 1.299s std dev: 0.01509s window: 28
```

Figure 22 Recovery Speed

The speed of the recovery node can be evaluate using the publish rate of the initial pose message. As shown in picture 20. After tested for all the places, for each recovery procedure, when the camera is capable of taking a picture with trigger in sight, the maximum recovery time is 1.332s, the minimum time is 1.199s.

Chapter 5 Conclusion and future work

5.1 Summary and Recommendation

This thesis proposed a solution to the kidnapped robot problem. First a set of fiducial markers combined with environmental features are used to mark a building. Then a voting algorithm is implemented based on object matching to recognize the place, followed by a position estimation based on marker's distance and angle.

As can be seen in the experiment result, using the algorithms above and a map marked with feature library, the robot can recover its position with an acceptable speed and error. The average error of the distance measurement is below 1%, while the average error in position estimation is about 0.31m. And the angle in pose estimation has an error less than 3.5 degrees. It is an error which can be calibrated later with the LIDAR feature and probabilistic model in AMCL node.

However, we can see this solution may still failed to recover robot's position at a certain situation. So the following recommendations is provided.

1. When marking the building, each place is recommend to have one or more redundant object, so the algorithm can be more robust and reliable.
2. The algorithm works more efficiently in a hallway than in a room, since objects in a room is more likely to be obscured.
3. More accurate estimation method for angle θ is recommend to reduce the total error.

5.2 Future Work

One of the problem emerged in this solution is: when applied in room recognition, the object in the room may be blocked or rotated, cause a recovery failure. So here, a machine learning algorithm can be used and make the recognition process more robust.

In a more broad scope, the library building process can also be replaced by computer program. A program can be implemented to detect the trigger autonomously, pick the object and store it in to a library. This will make the solution more user friendly.

References

- [1] Currie Adam. “The History of Robotics”. July 2006.
URL:<https://web.archive.org/web/20060718024255/http://www.faculty.ucr.edu/~currie/roboadam.htm>
- [2] Needham Joseph, “Science and Civilization in China: Volume 2, History of Scientific Thought”, Cambridge University Press. ISBN 0-521-05800-7.
- [3] CMU robotics institute, “2005-2010 research guide”,
URL: https://www.ri.cmu.edu/research_guide/medical_robotics.html.
- [4] Maybeck, P.S., “The Kalman filter: An introduction to concepts”, 1990, pp 194-204.
- [5] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. “Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)”. The MIT Press, 2005.
- [6] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: ICRA Workshop on Open Source Software. 2009.
- [7] “ROS - Powering the world's robots”. 2014. URL: <http://www.ros.org/>.
- [8] *navigation stack* - ROS Wiki. 2014. URL: <http://wiki.ros.org/navigation>.
- [9] Negenborn Rudy, “Robot localization and kalman filters”. Institute of Information and Computing Sciences, Utrecht University. 2003.
- [10] Hyon Lim, “Real-time single camera SLAM using fiducial markers”, Aug 2009.
- [11] Sivic, J., Okutomi, M., Pajdla, T. “Visual Place Recognition with Repetitive Structures”, March, In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* (Volume:37, Issue: 11).
- [12] Luke Ross, Dr Karen Bradshaw, “Fiducial Marker Navigation for Mobile Robots”, Oct, 2011
- [13] Luke Ross, “Fiducial Marker Navigation for Mobile Robots”, Aug, 2012
- [14] Alan Mutka, Damjan Miklic, Ivica Draganjac, Stjepan Bogdan, “A low cost vision based localization system using fiducial markers”, July 2008
- [15] Qingxiang Wu. “Rough computational methods on reducing cost of computation in Markov localization for mobile robots”. In: *Intelligent Control and Automation*, 2002. Proceedings of the 4th World Congress. pp. 1226–1230. ISBN 0-7803-7268-9
- [16] Roland Siegwart and Illah R. Nourbakhsh. “Introduction to Autonomous Mobile Robots”. Scituate, MA, USA: Bradford Company, 2004.
- [17] G.A. Bekey. “Autonomous Robots: From Biological Inspiration to Implementation and Control”. In: *Intelligent robotics and autonomous agents*. MIT Press, 2005.
- [18] Nyagudi, Nyagudi Musandu. “Humanitarian Algorithms: A Codified Key Safety Switch Protocol for Lethal Autonomy”, Sep 2015.
- [19] Chuho Yi, Byung-Uk Choi, “Detection and Recovery for Kidnapped-Robot Problem Using Measurement Entropy”, 2012

- [20] Jung, I.Lacroix,s., “Simultaneous localization and mapping with stereovision”, In: *Proceedings of the 11th International Symposium Robotic Research, Siena, Italy,2005*
- [21] Nister, D.Stewenius, H., “Scalable recognition with a vocabulary tree”, 2006
- [22] Morgan Quigley, Eric Berger, Andrew Y, “STAIR: Hardware and Software Architecture”, Stanford University, 2007
- [23] David G. Lowe,“Distinctive image features from scale-invariant keypoints”, In: *International Journal of Computer Vision*, 2004.
- [24] David G. Lowe, “Object recognition from local scale-invariant features”, In: *International Conference on Computer Vision, Corfu, Greece, Sep1999*.
- [25] Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski “ORB: An efficient alternative to SIFT or SURF”, 2011
- [26] Edward Rosten, Tom Drummond “Machine learning for high-speed corner detection”, 2006
- [27] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua, “BRIEF: Binary Robust Independent Elementary Features”, In: *11th European Conference on Computer Vision, Heraklion, Crete, Greece, Sep 2010*
- [28] Marius Muja and David G. Lowe: “Scalable Nearest Neighbor Algorithms for High Dimensional Data”. *Pattern Analysis and Machine Intelligence*” 2009
- [29] Marius Muja and David G. Lowe: “Fast Matching of Binary Features”. *Conference on Computer and Robot Vision (CRV) 2012*.
- [30] Marius Muja and David G. Lowe, “Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration”, In: *International Conference on Computer Vision Theory and Applications (VISAPP'09)*, 2009
- [31] OpenCV. 2014. URL: <http://opencv.org/>
- [32] OpenCV History, URL: https://en.wikipedia.org/wiki/OpenCV#cite_note-Itseez-1
- [33] The Geometry of Perspective Projection, URL: <http://www.cse.unr.edu/~bebis/CS791E/Notes/PerspectiveProjection.pdf>
- [34] D. P. Bertsekas and J. N. Tsitsiklis, “Introduction to Probability”, 2nd edition (2008)
- [35] Roland Siegwart, Illah R.Nourbakhsh, David Scaramuzza, “Introduction to Autonomous Mobile Robots”, second edition, 2011, pages 195-227
- [36] Z.Zhang. “Flexible Camera Calibration by Viewing a Plane from Unknown Orientations”. In: *International Conference on Computer Vision (ICCV'99)*, Corfu, Greece, pages 666-673, Sep 1999.