SLAM for Ground Robots: Theories and Applications

by

Linjian Xiang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Control System

Department of Electrical and Computer Engineering

University of Alberta

Abstract

The technique of Simultaneous Localization and Mapping (SLAM) has been widely studied and used in autonomous vehicles. The SLAM algorithms can construct the map from an unknown environment and at the same time, estimate the robot position. These are fundamentals of the autonomous robots, for example, the navigation module can be applied on the built map to accomplish self-driving. With the growing demand for the SLAM, researchers are asked to develop high-performance SLAM solutions with respect to better accuracy, and efficiency in both computational time and space.

This thesis explains several commonly-adopted SLAM algorithms at first, including mandatory background and mathematical derivations for these SLAM algorithms. Multiple Filter-based and Graph-based SLAM algorithms are derived, simulated and compared in the thesis, including Kalman Filter (KF), Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), Particle filter and Graph-based SLAM.

Finally, the important Kidnapped Robotic Problem (KRP) is studied. The KRP occurs when the robot is deliberately moved to another place without location knowledge or it loses its location information due to malfunctions. This research introduced a modification on Augmented Monte Carlo Localization (AMCL) and Cartographer to help robots recover from KRP. Enhanced methods are tested on saved real-world data and compared in the thesis.

Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Qing Zhao for her unwavering encouragement, guidance and support. Her patience and generosity allowed me to selfishly work on a new research topic, through which I have earned tremendous growth and fulfillment for the last three years of study and research. Her expertise and inspiration have given me strong faith and back support to fight through research and real life obstacles. Also, I would like to express my gratitude towards Dr. Hong Zhang for his kindness in providing experimental data sets.

I would also like to thank Dr. Cyrill Stachniss, even though he does not know me, but I start learning SLAM from his fantastic video SLAM courses. I would encourage anyone who is interested in SLAM to go through these courses. Also, I am grateful to Dr. S. Thrun, Dr. W. Burgard and Dr. D. Fox for their book *Probabilistic Robotics*, which has laid some important foundation for this work and prepared me well through the research.

Finally, I would like to thank my family. Their love and encouragement have been my principal support during these years.

Contents

List of Tables

List of Figures

List of Symbols

w	Angular velocity, for motion model
E	Essential matrix, for relative pose transformation
y	Estimated observation
F	Fundamental matrix, for relative pose transformation
g	Inverse of observation function h
A	Jacobian of motion model
H	Jacobian of observation model
M	Map state, for feature-based map
F	Map states transformation model, for feature-based map
f	Nonlinear motion model
h	Observation function
σ	Observation noise
V	Observation noise covariance
Q	Process noise covariance
Σ, P	Process error Covariance
$\mathcal{X}$	Particles, for particle filter
R	Robot state, for feature-based map
R	Rotation, for motion transformation
z	Sensor observation
$\mathcal{X}$	Sigma points, for unscented transformation
ξ	Submap pose transformation
T_ξ	Transformation between scan frame to submap frame
t	Translation, for motion transformation
v	Velocity, for motion model

Chapter 1

Introduction

In recent years, autonomous mobile systems including autonomous robots and vehicles gradually become a reality, and their applications steadily grow, showing impacts in our daily life. As one of the key technical components, the Simultaneous Localization and Mapping (SLAM) problem plays an essential role in autonomous mobile systems. The SLAM algorithms have the ability to process limited sensor data to accurately and effectively construct a map of the unknown environment with the robot location. Over the years, SLAM algorithms are continuously enhanced to have better accuracy and computational and spatial efficiency, so that they can even run on many low-cost devices. In the near future, it is highly anticipated that autonomous robots and vehicles will be cheaper, more reliable and widely deployed.

There are many different SLAM approaches in the existing literature, and they are divided into two major categories, filter-based and graph-based SLAM. This thesis is focused on investigating SLAM in these two categories. Specifically, how SLAM operates using different sensors and incorporating real-world environment data. Different methods are reviewed and studied to understand their pros and cons. Finally the Kidnapped Robotic Problem (KRP) and its recovery is studied using SLAM approaches as it is a common and challenging problem in practical applications. In this thesis, procedures of SLAM and KRP algorithms are discussed in detail, with simulation and real-world experimental data to demonstrate and compare the performance of different approaches.

1.1 Literature Review

The past decades have seen rapid and exciting progress in SLAM related research with many implementations. Works are mainly focused on improving the accuracy of estimating the map and robot poses, and computational efficiency.

The concept of Simultaneous Localization and Mapping (SLAM) was first introduced by [57], and then the paper [36] proposed to use multiple servo-mounted sonar sensors and extended Kalman filter (EKF) to extract environment features and track the robot location. After that, the Lidar sensor came into play, and the SLAM algorithms became more robust and reliable. There are challenges for the EKF-SLAM and other KF based SLAM algorithms. First, they impose fundamental assumption on Gaussian distributions; the second challenge lies in the approximation accuracy of linearization; and the third one is due to the correspondence between sensor data and features. On the other hand, the particle filter based SLAM, which is another well-known filter-based method, is able to handle some of the challenging problems.

Different from EKF-SLAM, which utilizes features for mapping, the PF-SLAM adopts occupancy grid maps. For a SLAM problem, the particle filter [13] [31] [49] uses particle distribution to represent probability distribution. Monte Carlo Localization (MCL) [11] is a direct application of particle filters to robotic localization. It is one of the most commonly used approaches because its formulation is straightforward and it is relatively easy to implement with good performance. In [61] the mixture proposal distribution was proposed for particle resampling to make the MCL more robust and accurate. The adaptive MCL [17] [16] was developed in which particle set number is adjusted to achieve fast convergence and computational efficiency. The Augmented MCL [19] incorporating an exponential filter was applied to solving the Kidnapped Robotic Problem (KRP) for which the global localization was used to recover KRP. Moreover, PF filter using 3D liDAR data was reported in [63]. In [55] visual-based PF-SLAM was introduced, in which features from the scale-invariant feature transform (SIFT) were used. In addition to the

more conventional methods, artificial neural network (ANN) approaches have recently been applied to SLAM. For example, [1] proposed to use the convolutional neural network (CNN) to sample the particles. Although filter-based SLAM algorithms have been extensively studied, there are certain obstacles that are inherent to the approach and are not easy to overcome. For example, no matter how accurate the estimation is, filter-based SLAM schemes usually suffer from increasing errors accumulated from every iteration.

Another main category of SLAM approaches are graph-based. In this method, a graph is constructed where nodes represent robot and landmark poses, and sensor measurement connecting two nodes are called edges/constraints. Then optimization is applied to update the nodes poses by minimizing the error and reducing the contradiction between constraints. This way it helps graph-based SLAM to mitigate effects of sensor noise and accumulated estimation error. The graph-based optimization task can be converted to a (nonlinear) Least Squares (LS) problem. The work in [40] was among the first that proposed graph-based SLAM formulation and map refinement by global optimization. Then Gutmann and Konolige [23] presented a method of local registration and global correlation (LRGC), which performed loop closure detection with every new sensor input. To minimize constraint network error, Dellaert and Kaess [12] presented exploit sparse matrix factorization to solve LS optimization in SLAM, and later on Kaess etal. [30] used QR factorization to compute and solve the graph optimization. In some cases, robot motion dynamics and odometry are unknown, hence many approaches are proposed to utilize purely sensor data to estimate constraints and relative pose between nodes. The paper [14] firstly proposed to use Iterated Closest Point(ICP) [3] to estimate the relative pose between two range scans. The TrICP [8] used the Least Trimmed Squares approach along with ICP operation to achieve improved robustness and accurate motion estimation. The Random sample consensus(RANSAC) [15] is frequently applied on point clouds and image features to find the optimal motion estimation. Another important research topic is focused on reducing the time cost of detecting loop closure. For example, DBOW2 [18] applies the method of bag-of-word [56] to divide image features

into pre-trained clusters and number sequences are generated to represent the image features, since matching the number sequence is much faster than matching full image features. Similarly the work in [27] used histogram-based matching for 2D-LiDAR scans to increase the speed of loop closure detection. To achieve real-time scan matching for loop closure detection, [26] used a branch-and-bound scan (BBS) matching approach.

Currently there are multiple open-source SLAM packages made available to the general public through collaboration of developers in this area. In addition, most packages have built extension libraries for the ROS platform, which can be modified and tested conveniently by researchers. For PF-based SLAM, the Gmapping tool [20] takes 2D-LiDAR scans as inputs and generates an occupancy grid map, which can be output as a gray-scale image. The AMCL [19] package is available which uses 2D-LiDAR data as inputs and generates maps to perform global and local localization. The Cartographer [33] is capable of performing mapping and localization for 2D and 3D laser scans, and it uses a scan-matching approach for odometry estimation and BBS for loop-closure detection. For visual-based SLAM packages, the ORB-SLAM2 [43] uses ORB features for mapping and localization, and it can take both monocular and stereo camera images, as well as RGB-D camera images as inputs. The RTAB-Map [34] is another popular SLAM algorithm based on incremental appearance-based loop-closure detector and it also supports RGB-D, stereo, and LiDAR data.

1.2 Motivation and Thesis Structure

In recent development of SLAM, many new techniques have been utilized in SLAM algorithms and they join forces to deliver improved performance. There are various modifications to the more traditional approaches and the existing literature is rich. By focusing on two main important types of SLAM methodologies, this thesis provides a detailed study of SLAM theories and applications, and it also serves as a learning road map for the author.

This work first provides a detailed explanation of various background knowl-

edge used for tackling the SLAM problem. For example, Bayes filtering basics, Kalman filter, and its extensions, particle filters, least-squares, and robot motion models are reviewed. This helps to understand and prepare knowledge for most SLAM algorithms. There are many modifications to filter based SLAM methods which are developed in earlier years. This thesis demonstrates the implementation of Extended Kalman Filter (EKF) SLAM, Unscented Kalman Filter (UKF) SLAM, and Particle Filter SLAM. In addition, to better understand filter based SLAM algorithms, we provide simulation programs and simulation result to compare performance of the EKF and UKF based SLAM.

The Graph-based formulation and optimization are commonly adopted in most advanced SLAM algorithms in recent years. In this thesis, the various implementation and optimization approaches for graph-based SLAM are introduced. In addition, an application case study is performed that shows how to implement the ORB feature-based SLAM algorithm for autonomously docking a vehicle.

In the last part of the thesis, detection and recovery methods for Kidnapped Robotic Problem (KRP) are studied. The KRP occurs when a robot loses its location information. It is known that the Augmented Monte Carlo Localization (AMCL) method can detect and solve kidnapping problems by tracking the sudden drop of particles' average weight and then performing a global localization to estimate the robot's new pose. We propose a new method to assist in solving the (KRP) in the Monte Carlo localization approach. The proposed work improves the AMCL by adopting the idea of the global/static Costmap. The Costmap aided AMCL algorithm is able to recognize those absolutely wrong particles and then randomize them to enhance the speed of recovery from the localization failure. Furthermore, we implemented a KRP detector and loop-closure based KRP recovery on the Cartographer platform. Simulations are carried out on different maps and real-time data to validate and demonstrate the proposed methods' performances.

To summarize, in Chapter 2, general background on the methodologies used for SLAM problems is reviewed. Chapter 3 includes various filter-based SLAM approaches. The Graph-based SLAM and its demonstrations are described in

Chapter 4. Chapter 5 discusses the KRP and its solution. Finally, Chapter 6 closes this thesis by summarising the work presented and discussing the potential future work.

Chapter 2

Background and Preliminaries

This chapter introduces important background knowledge for understanding the SLAM problem. The following section starts with an introduction to the basic Bayes filter, the foundation of filter-based SLAM algorithms. After that, commonly used filter techniques are presented, including the Kalman Filter (KF), Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), and Particle Filter. In addition, the nonlinear Least-Squares (LS) and its solutions are also reviewed, which provide practical solutions to optimization-based SLAM. Finally, the ground robot motion model is derived.

2.1 Bayes Filter

Filter-based SLAM algorithms are built upon the concept of Bayes filter. In brief, the Bayes filter recursively updates robotic states by its prediction and values of the actual measurement. In Probabilistic Robotics [60], an example (Fig. 2.1) is given to illustrate the localization of a mobile robot using the probabilistic approach. Before any measurement, the robot position probability $bel(x)$ is assumed to be uniformly distributed; and suppose that the robot takes the first sensor measurement next to a door, then this sensor reading suggests that the robot has a higher probability $p(z|x)$ near doors. Notice that this distribution has three peaks, each corresponding to one of the doors. Then the robot *belief* is updated based on the current measurement, as it becomes the product of the measurement probability distribution and the previous belief. After that, the robot moves and takes more measurements. At

the same time, its belief iterates, and the robot gets higher probability and better confidence as to where it is.

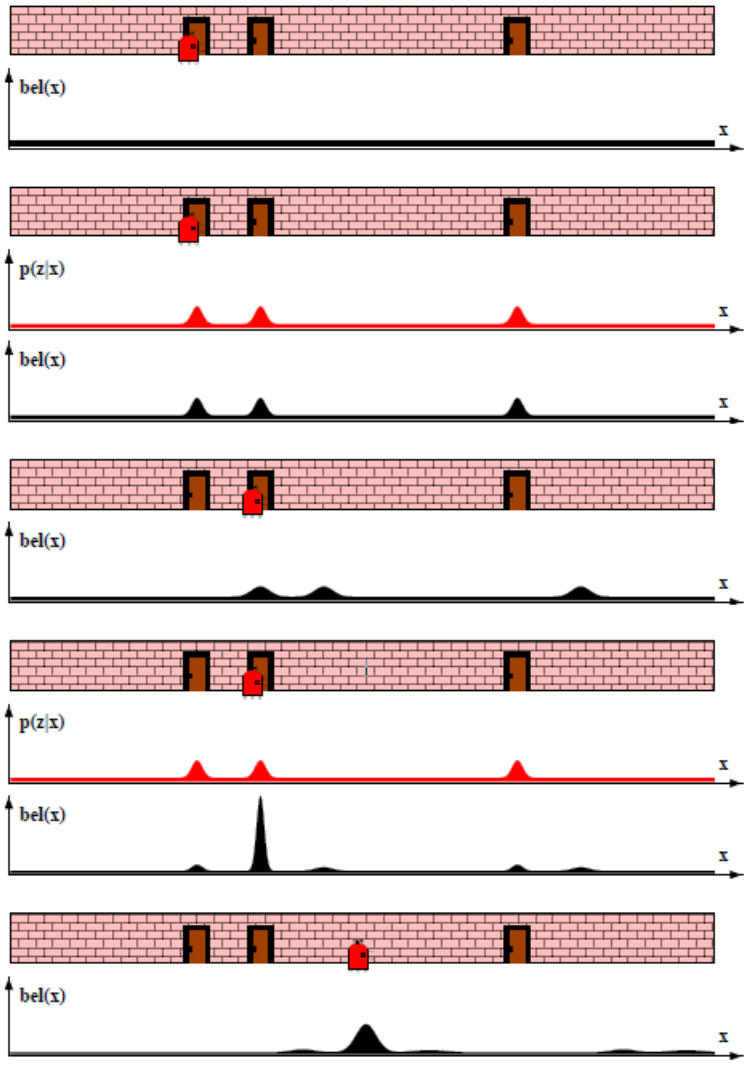

Figure 2.1: Mobile robot global localization using Markov localization, [60]

2.1.1 Basics in Probability

Gaussian distribution function

A scalar random variable x following a Gaussian distribution (i.e. normally distributed), $x \sim \mathcal{N}(\mu,\ \sigma^2)$, its distribution function is written as follows, where μ is its mean and σ^2 is the variance:

$$p(x) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\} \tag{2.1}$$

When x is a vector, $x \sim \mathcal{N}(\mu,\ \Sigma)$, where Σ is the positive definite covariance matrix, the multivariate Gaussian distribution is given by,

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\} \tag{2.2}$$

Theorem of total probability

The rule of total probability is fundamental to relate marginal probability and conditional probability. Let the marginal probability be denoted as $p(x)$ of x; the conditional probability denoted as $p(x|y)$ of x for a given variable y. The discrete and continuous expressions for the total probability theorem are shown as follows:

$$p(x) = \sum_y p(x|y)p(y) \tag{2.3}$$

$$p(x) = \int p(x|y)p(y)dy \tag{2.4}$$

Bayes' Rule

Bayes' Rule is used to update the belief with new measurements (or evident). The following equations show the discrete and continuous Bayes' rule based on total probability equations.

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\sum_x p(y|x)p(x)} \tag{2.5}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\int p(y|x)p(x)dx} \tag{2.6}$$

The conditioning Bayes' rule with additional event z can be expressed as:

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)} \tag{2.7}$$

$$p(x,y|z) = p(x|z)p(y|z) \tag{2.8}$$

2.1.2 Probabilistic Generative Laws

Probabilistic laws govern the recursively updated states and measurements. The state at time t, x_t is conditioned on all past states, measurements, and control inputs. Because of the measurement and input uncertainty, the states

are generated in a stochastic fashion, and its probability distribution can be written in the form:

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \tag{2.9}$$

where $x_{0:t-1}$ represents the past states, $z_{1:t-1}$ represents the past measurements and $u_{1:t}$ incorporates the control inputs (the current input u_t is included since the robot is assumed to first execute the control action before obtaining the measurement). It should be noted that the previous state x_{t-1} is a sufficient statistic of all previous controls and measurements till the time $t-1$. Therefore, the state probability distribution of x_t can be calculated based on the previous state x_{t-1} and the current control input u_t.

Similarly, the measurements can also be written as the probability distribution form. In particular, the current state x_t is sufficient in predicting the measurement z_t, therefore we have,

$$p(z_t|x_{1:t}, z_{1,t-1}, u_{1:t}) = p(z_t|x_t) \tag{2.10}$$

$p(x_t|x_{t-1}, u_t)$ is also called the state transition probability, which specifies how the state evolves over time with control input u_t. Also $p(z_t|x_t)$ is called the measurement probability, which is the probability transformation specifying how the measurement z_t is generated from the state x_t.

2.1.3 Belief Distributions

In reality, the robot or its environment states cannot be observed directly, but they can be referred by available data (e.g. measurements and control inputs). The *belief* is introduced to represent the internal knowledge of states, based on conditional probability distributions. Belief distribution, denoted as $bel(x_t)$, is posterior probability over states conditioned on the available data. It can be written as:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \tag{2.11}$$

where x_t is the state at time t; and $z_{1:t}$ and $u_{1:t}$ are available measurements and control inputs up to time t, respectively. As the *belief* takes the measurement

z_t at time t, we can also define the probability before incorporating z_t, denote as $\overline{bel}$, also called the prior probability.

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \tag{2.12}$$

The process of computing the prior $\overline{bel}$ through the previous posterior and control input, is usually called **prediction**. Calculating $bel(x_t)$ from the $\overline{bel}$ is called **correction** or update.

2.1.4 Derivation of the Bayes Filter

The Bayes filter is the most commonly used algorithm to recursively calculate beliefs. The steps of using Bayes filter is shown in Algorithm 1.

```
1 Bayes Filter Algorithm (bel(x_{t-1}), u_t, z_t) ;
2 for all x_t do
3 |   bel-bar(x_t) = ∫ p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} ;
4 |   bel(x_t) = η p(z_t|x_t) bel-bar(x_t)
5 end
6 return bel(x_t)
```

Algorithm 1: Bayes-filter pseudo algorithm

The line 3 of the algorithm is the prediction step, in which prior $\overline{bel}(x_t)$ is calculated by the integration of product of state transition probability in Eq.(2.9) and the previous posterior $bel(x_{t-1})$. The line 4 is the update step, where the posterior $bel(x_t)$ is calculated by product of measurement probability in Eq. (2.10) and prior $\overline{bel}(x_t)$ at time t with a normalization constant η. The detailed derivations are briefly explained in the following.

First of all, apply the theorem of total probability to $\overline{bel}(x_t)$:

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) = \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) p(x_{t-1}|z_{1:t-1}, u_{1:t}) dx_{t-1} \tag{2.13}$$

where $p(x_t|x_{1:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t)$ based on Eq. (2.9) and

$$p(x_{t-1}|z_{1:t-1}, u_{1:t}) = p(x_{t-1}|z_{1:t-1}, u_{1:t-1})$$

since x_{t-1} does not contain the information of u_t from future time (Markov assumption). Furthermore, $p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) = bel(x_{t-1})$. Finally the pre-

diction equation can be written as:

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1} \tag{2.14}$$

Then apply the conditioning Bayes rule on Eq.(2.11), the equation expends to as follow:

$$\begin{aligned} bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) = p(x_t|z_t, z_{1:t-1}, u_{1:t}) &= \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \\ &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t}) \end{aligned} \tag{2.15}$$

With Eq. (2.10) and Eq. (2.12), the posterior distribution is simplified as:

$$\begin{aligned} bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) &= \eta p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t}) \\ &= \eta p(z_t|x_t)\overline{bel}_t \end{aligned} \tag{2.16}$$

2.2 KF, EKF and UKF

For a linear or nonlinear dynamic system, assuming that the measurement and system noises are Gaussian, the Bayes filter can be extended and implemented as the *Kalman filter (KF)*, *Extended Kalman filter(EKF)* and *Unscented Kalman filter(UKF)*. The KF applies to the linear system and measurement models, while EKF and UKF can handle nonlinear system and measurement models. They are briefly reviewed in the following.

2.2.1 Kalman Filter (KF)

Given a linear state equation with states x_t at time t, the normally distributed noise $\varepsilon_t \sim \mathcal{N}(0, Q_t)$ and input u_t:

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{2.17}$$

the term $A_t x_{t-1} + B_t u_t$ is the mean of the posterior state x_t, denoted as $\bar{x}_t$. The following shows a linear measurement model with normally distributed measurement noise $\sigma_t \sim \mathcal{N}(0, V_t)$:

$$z_t = C_t x_t + \sigma_t \tag{2.18}$$

Based on the Markov assumption and Bayes filter, the well-known Kalman filter algorithm [60] can be derived. In the following $\bar{x}_t$ is the prior state estimation and $\hat{x}_t$ is the posterior state estimation. Matrix $\bar{\Sigma}_t$ and $\hat{\Sigma}_t$ are the prior and posterior error covariance matrices, which represent how accurate the state estimation is.

1 Kalman filter $(\hat{x}_{t-1}, \hat{\Sigma}_{t-1}, u_t, z_t)$:
2 $\bar{x}_t = A_t \hat{x}_{t-1} + B_t u_t$;
3 $\bar{\Sigma}_t = A_t \hat{\Sigma}_{t-1} A_t^\top + Q_t$;
4 $K_t = \bar{\Sigma}_t C_t^\top (C_t \bar{\Sigma}_t C_t^\top + V_t)^{-1}$;
5 $\hat{x}_t = \bar{x}_t + K_t(z_t - C_t \bar{x}_t)$;
6 $\hat{\Sigma}_t = (I - K_t C_t)\bar{\Sigma}_t$;
7 return $\hat{x}_t, \hat{\Sigma}_t$

Algorithm 2: Kalman filter algorithm

2.2.2 Nonlinear State Estimation

For a general nonlinear system and nonlinear measurement model,

$$x_{t+1} = f(x_t, u_t, n_t) \tag{2.19}$$

$$y_t = h(x_t) + \sigma_t \tag{2.20}$$

where f is a nonlinear function of states x_t, input u_t and the process noise n_t, which includes disturbances and modeling errors. The output y_t is the observed value, which contains measurement noise σ_t. It is assumed that all noise vectors are normally distributed and have zeros mean. For such a system, state estimation is more challenging and usually involves approximations.

Linearization

In practice, often sensors are non-ideal and perfect system models do not exist, hence the state estimation or observation of the system can be considered as an uncertainty transformation problem. Given a random vector x with mean $\bar{x}$ and covariance Σ_{xx}. The random vector y has a relationship with x as,

$$y = f(x)$$

where $f(.)$ can be taken as a transformation and the output y has the mean $\bar{y}$ and covariance Σ_{yy}. If $f(.)$ is linear, e.g. $f(x) = Ax$, and x follows Gaussian

distribution, then the output y will also follow the Gaussian distribution, and

$$\bar{y} = A\bar{x}$$

$$\Sigma_{yy} = A\Sigma_{xx}A^\top$$

However in many cases, e.g. SLAM problem, the models (e.g. a robot motion model) are mostly nonlinear. Let $f(.)$ be a nonlinear function, and $x = \bar{x} + \delta x$, where δx is zero mean Gaussian with covariance Σ_{xx}. We need to find the mean of the uncertain output y and its covariance Σ_{yy}, with an uncertain input x. We can rewrite the equation using the Taylor series expansion. For simplicity, the scalar case is considered at first,

$$f(x) = f(\bar{x} + \delta x) = f(\bar{x}) + \nabla f \delta x + \frac{1}{2}\nabla^2 f \delta x^2 + \cdots \tag{2.21}$$

Where the $\nabla^n f$ represents the nth order derivatives of the function f with respect to x. Let $E(.)$ denote the mean value. So the mean of $f(x)$ or $\bar{y}$ can be written as

$$\begin{aligned}\bar{y} = E[f(x)] &= E[f(\bar{x})] + E[\nabla f \delta x] + E[\frac{1}{2}\nabla^2 f \delta x^2] + \cdots \\ &= f(\bar{x}) + \frac{1}{2}\nabla^2 f\, \Sigma_{xx} + \cdots \end{aligned} \tag{2.22}$$

By truncating the terms with order higher than 2, the linear approximation of $\bar{y}$ is obtained as

$$\bar{y} = f(\bar{x}) \tag{2.23}$$

Then one can compute the covariance Σ_{yy}. Now we consider the general multivariate case,

$$\begin{aligned}\Sigma_{yy} &= E[(f(x) - \bar{y})(f(x) - \bar{y})^\top] \\ &= E[(\nabla f \delta x)(\nabla f \delta x)^\top] \qquad (2.24) \\ &= \nabla f \Sigma_{xx} (\nabla f)^\top \qquad (2.25)\end{aligned}$$

where ∇f is called the Jacobian. It should be noted that in many practical situations linearization introduces significant approximation errors.

Unscented Transform

The Unscented transformation is another estimation method for nonlinear transformation of random variables [29] [62] [28]. Compared with the linearization method, which only captures the first order of the Taylor series expansion, for the Unscented transform, the mean and covariance of input x and output y are corrected up to second-order , which means the Unscented transform has a better performance than the first order linearization. In short, the Unscented transform uses given mean and covariance to find a set of sigma points to represent the discrete probability distribution. Then it applies the nonlinear transformation to propagate each point, and the transformed points represent the discrete probability distribution for the transformed random variables.

Given $2n+1$ sigma points in vector $\mathcal{X}^{x_i}$ with weights W_i and $\sum_{i=0}^{2n} W_i = 1$, its mean $\bar{x}$ and covariance Σ_{xx} have following properties:

$$\bar{x} = \sum_{i=0}^{2n} W_i \mathcal{X}^{x_i} \tag{2.26}$$

$$\Sigma_{xx} = \sum_{i=0}^{2n} W_i (\mathcal{X}^{x_i} - \bar{x})(\mathcal{X}^{x_i} - \bar{x})^\top \tag{2.27}$$

Sigma points and weights are chosen as following [29] [62]:

$$\begin{aligned}
\mathcal{X}_0^{x_i} &= \bar{x} \\
\mathcal{X}_i^{x_i} &= \bar{x} + (\sqrt{(n+\lambda)\Sigma_{xx}})_i \quad for \quad i = 1, \cdots, n \\
\mathcal{X}_i^{x_i} &= \bar{x} - (\sqrt{(n+\lambda)\Sigma_{xx}})_{i-n} \quad for \quad i = n+1, \cdots, 2n \\
W_{m0} &= \lambda/(n+\lambda) \\
W_{c0} &= W_{m0} + (1 - \alpha^2 + \beta) \\
W_i &= \frac{1}{2(n+\lambda)} \\
W_{mi} &= W_{ci} = \frac{1}{2(n+\lambda)}
\end{aligned} \tag{2.28}$$

where n is the dimension of x and the scaling parameter $\lambda = \alpha^2(n+\kappa) - n$. $\alpha \in (0, 1]$ is a small scaling factor for the spread of the sigma points around $\bar{x}$, and κ is a secondary scaling factor which is usually set to 0. The value of β is related to the distribution of x and $\beta = 2$ is the optimal choice for

Gaussian distribution. Furthermore, W_m and W_c denote weights for mean and covariance, and the square root of covariance matrix Σ is calculated through the Cholesky factorization. For the i_{th} sigma point, the column vector of Cholesky factorized covariance matrix is used,

$$SS^\top = \Sigma$$

Hence the subscript i of the sigma point i denotes the i_{th} column of S. The transformed output mean and covariance can be expressed as following:

$$\begin{aligned} \mathcal{X}^y &= f(\mathcal{X}^x) \\ \bar{y} &= \sum_{i=0}^{2n} W_i \mathcal{X}^{y_i} \\ \Sigma_{yy} &= \sum_{i=0}^{2n} W_i (\mathcal{X}^y - \bar{y})(\mathcal{X}^y - \bar{y})^\top \end{aligned} \tag{2.29}$$

2.2.3 Extended Kalman Filter

The EKF adopts linearization to orignal KF algorithm and the general EKF algorithm [60] is shown in Algorithm 3:

1 Extended Kalman filter $(\hat{x}_{t-1}, \hat{\Sigma}_{t-1}, u_t, z_t)$:
2 $\bar{x}_t = f(\hat{x}_{t-1}, u_{t-1})$;
3 $A_t = \frac{\partial f(x,u)}{\partial x}|_{x=\hat{x}_{t-1}}$;
4 $H_t = \frac{\partial h(x)}{\partial x}|_{x=\bar{x}_t}$;
5 $\bar{\Sigma}_t = A_t \hat{\Sigma}_{t-1} A_t^\top + Q_t$;
6 $K_t = \bar{\Sigma}_t H_t^\top (H_t \bar{\Sigma}_t H_t^\top + V_t)^{-1}$;
7 $\hat{x}_t = \bar{x}_t + K_t(z_t - h(\bar{x}_t))$;
8 $\hat{\Sigma}_t = (I - K_t H_t)\bar{\Sigma}_t$;
9 return $\hat{x}_t, \hat{\Sigma}_t$

Algorithm 3: Extended Kalman filter algorithm

For the line 2 of the algorithm, the states mean $\bar{x}_t$ is predicted using the state transition function f with the mean $\bar{x}_{t-1}$. For the prediction of the covariance $\bar{\Sigma}_t$ from line 5, it depends on previous covariance mean $\bar{\Sigma}_{t-1}$ and the linearized motion dynamic A_t. Similar as the state updating, line 7, the non-linear observation equation is also linearized such that H_t is obtained for output mean transformation. In EKF, due to the first order linearization at the local estimated states, such approximation errors may lead to sub-optimal

solutions and sometimes divergence, for example, initial state estimation is not close to the true value.

2.2.4 Unscented Kalman Filter (UKF)

The UKF method is a modification of EKF, which replaces the first-order linearization with the Unscented tranformation. The first step is to choose sigma points around states and their weights using Eq.(2.28), for example, the state of parameter x at time *t-1*, with 3 sigma points,

$$\mathcal{X}_{t-1}^{x} = [\bar{x}_{t-1} \quad \bar{x}_{t-1} \pm \sqrt{(n+\lambda)\Sigma_{t-1}}]$$

Assume there are $2n$ sigma points used for each transformation, the UKF algorithm 4 can be modified from EKF as following steps:

1. The state prediction in line 2 of EKF becomes

$$\mathcal{X}_{t}^{\bar{x}} = f(\mathcal{X}_{t-1}^{x}, u_t) \tag{2.30}$$

   where $\mathcal{X}_{t-1}^{x}$ is the sigma points for system states, where

$$\bar{x}_t = \sum_{i=0}^{2n} W_i^m \mathcal{X}_{i,t}^{\bar{x}} \tag{2.31}$$

2. The covariance prediction in line 5 becomes

$$\bar{\Sigma}_t = \sum_{i=0}^{2n} W_i^c (\mathcal{X}_{i,t}^{\bar{x}} - \bar{x}_t)(\mathcal{X}_{i,t}^{\bar{x}} - \bar{x}_t)^\top + Q_t \tag{2.32}$$

   Where Q_t is the process noise covariance.

3. Then for the estimation step, the output equation will be updated as

$$\mathcal{X}_{t}^{y} = h(\mathcal{X}_{t}^{x}) \tag{2.33}$$

   and its mean

$$y_t = \sum_{i=0}^{2n} W_i^m \mathcal{X}_{i,t}^{y}$$

   the matrix H_t is the output transfer matrix,the output covariance is

$$\Sigma_{yy} = H_t \bar{\Sigma}_t H_t^\top = \sum_{i=0}^{2n} W_i^c (\mathcal{X}_{i,t}^{y} - \bar{y}_t)(\mathcal{X}_{i,t}^{y} - \bar{y}_t)^\top + V_t \tag{2.34}$$

4. In the line 6, we are able to rewrite $\bar{\Sigma}_t H_t^\top$ as

$$\bar{\Sigma}_t H_t^\top = E[(x-\bar{x})(x-\bar{x})^\top]H_t^\top = E[(x-\bar{x})((x-\bar{x})H_t^\top)] = E[(x-\bar{x})(y-\bar{y})] = \Sigma_{xy}$$

Then for the UKF, Σ_{xy} becomes

$$\Sigma_{xy} = \sum_{i=0}^{2n} W_i^c(\mathcal{X}_t^{\bar{x}} - \bar{x}_t)(\mathcal{X}_t^y - y_t)^\top \tag{2.35}$$

Using the Eq.(2.34) and Eq.(2.35) the UKF gain can be written as:

$$K_t = \Sigma_{xy}\Sigma_{yy}^{-1} \tag{2.36}$$

5. The UKF state update equation is

$$x_t = \bar{x}_t + K_t(z_t - y_t) \tag{2.37}$$

6. For the UKF state covariance update, we start with the EKF case

$$\begin{aligned}
\Sigma_t &= (I - K_t H_t)\bar{\Sigma}_t \\
&= \bar{\Sigma}_t - K_t H_t \bar{\Sigma}_t \\
&= \bar{\Sigma}_t - K_t(\Sigma_{xy})^\top \\
&= \bar{\Sigma}_t - K_t(\Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yy})^\top \\
&= \bar{\Sigma}_t - K_t(K_t\Sigma_{yy})^\top \\
&= \bar{\Sigma}_t - K_t\Sigma_{yy}K_t^\top
\end{aligned} \tag{2.38}$$

Finally the UKF algorithm is obtained.

1 Unscented Kalman Filter $(\hat{x}_{t-1}, \hat{\Sigma}_{t-1}, u_t, z_t)$:
2 $\mathcal{X}_{t-1}^u = [\bar{u}_{t-1} \quad \bar{u}_{t-1} \pm \sqrt{(n+\lambda)\Sigma_{t-1}^u}]$
3 $\mathcal{X}_t^{\bar{x}} = f(\mathcal{X}_{t-1}^{\hat{x}}, \mathcal{X}_t^u)$
4 $\bar{\Sigma}_t = \sum_{i=0}^{2n} W_i^c(\mathcal{X}_{i,t}^{\bar{x}} - \bar{x}_t)(\mathcal{X}_{i,t}^{\bar{x}} - \bar{x}_t)^\top + Q_t$
5 $\mathcal{X}_t^y = h(\mathcal{X}_t^{\bar{x}}), \quad \bar{y}_t = \sum_{i=0}^{2n} W_i^m \mathcal{X}_{i,t}^y$
6 $\Sigma_{yy} = \sum_{i=0}^{2n} W_i^c(\mathcal{X}_{i,t}^y - \bar{y}_t)(\mathcal{X}_{i,t}^y - \bar{y}_t)^\top + V_t$
7 $\Sigma_{xy} = \sum_{i=0}^{2n} W_i^c(\mathcal{X}_t^{\bar{x}} - \bar{x}_t)(\mathcal{X}_t^y - y_t)^\top$
8 $K_t = \Sigma_{xy}\Sigma_{yy}^{-1}$
9 $\hat{x}_t = \bar{x}_t + K_t(z_t - \bar{y}_t)$
10 $\hat{\Sigma}_t = \bar{\Sigma}_t - K_t\Sigma_{yy}K^\top$
11 return $\hat{x}_t, \hat{\Sigma}_t$

Algorithm 4: Algorithm of Unscented Kalman filter

2.3 Particle Filter (PF)

The particle filter has been used to solve SLAM problems [60], it uses a set of particles (samples) to represent the posterior distribution of certain stochastic process given noisy observations. The particles can represent a much broader category of distributions (e,g, non-Gaussian) than those with parametric forms. Hence PF can be considered as an alternative non-parametric implementation of the Bayes filter. In the SLAM problems, it has been widely used in mapping and Monte Carlo localization.

In the particle filter, particles x represent samples on posterior distributions. The particle filter algorithm is shown in Algorithm 5, where there are M numbers of particles in particle set $\mathcal{X}$ at time t:

$$\mathcal{X}_t = x_t^{[1]}, x_t^{[2]}, \cdots, x_t^{[M]} \tag{2.39}$$

```
1  Particle-Filter (X_{t-1}, u_t, z_t) :
2  X̄_t = X_t = ∅ ;
3  for m = 1 to M do
4  |   sample x_t^[m] ~ p(x_t | u_t, x_{t-1}^[m]) ;
5  |   w_t^[m] = p(z_t | x_t^[m]) ;
6  |   X̄_t = X̄_t + < x_t^[m], w_t^[m] > ;
7  end
8  for m = 1 to M do
9  |   draw i with probability ∝ w_t^[i] ;
10 |   add x_t^[i] to X_t
11 end
12 return X_t
```

Algorithm 5: Particle filter pseudo algorithm

1. In line 4, the particle $x_t^{[m]}$ at time t is generated by the state transition distribution $p(x_t|u_t, x_{t-1}^{[m]})$ based on the previous time particles $x_{t-1}^{[m]}$ and control input u_t. As the density distribution of the $\mathcal{X}_{t-1}$ can be seen as the approximation of $bel(x_{t-1})$, the prior probability distribution $\overline{bel}(x_t)$ can be represented by the density distribution of $\bar{\mathcal{X}}_t$. This step can be referred to as prediction step of Bayes filter.

2. Line 5 calculates the *importance factor* or called weight of each particle, denoted by $w_t^{[m]}$. Weights incorporate the measurement information into

the particle set, precisely, the weight of a particle $x_t^{[m]}$ equals to the probability of obtaining measurement z_t under this particle.

3. The second **for** loop (Line 8 - Line 11) is called *resampling* or *importance sampling* of particles, which implements the approximation of update step to get the distribution close to posterior distribution $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$. In order to change the distribution of particles from $\overline{bel}(x_t)$ to $bel(x_t)$, we transform particles in $\bar{\mathcal{X}}$ to another particle set $\mathcal{X}$ with same size according to posterior distribution. The detailed procedures are discussed in next section.

2.3.1 Resampling

The resampling step can be seen as an update step of the Bayes filter, it uses a set of particles and their weights to generate a new set of particles with equal weights. The new particle set is used to represent the posterior distribution. Assume that we attempt to convert the probability distribution from g to f. The distribution function f is called *target distribution* to represent the desired distribution, and the g is called *proposal distribution*. In particle filter, f corresponds to the posterior belief $bel(x_t)$ and g is the prior $\overline{bel}(x_t)$, while the probability distributions are approximated by particle density distributions. Given a particle set $\mathcal{X}_t$ with M number of particles, $x^1, \cdots, x^M$, the approximation of proposal distribution g can be performed using $\mathcal{X}_t$. Specifically the cumulative probability of any subset A of the distribution g (area under g within the range) is approximated by the particle percentage in the range. If $M \to \infty$ the probability of particles fall in set A equals to the integral of g.

$$\frac{1}{M}\sum_{m=1}^{M} I(x^{[m]} \in A) \to \int_A g(x)dx \tag{2.40}$$

Furthermore, to get the transmission between f and g, every particle can incorporate a weight w

$$w^{[m]} = \frac{f(x^{[m]})}{g(x^{[m]})} \tag{2.41}$$

Then combining these two equations, the set of weighted particles in A is approximated to the integration of f in A

$$\left[\sum_{m=1}^{M} w^{[m]}\right]^{-1} \sum_{m=1}^{M} I(x^{[m]} \in A) w^{[m]} \rightarrow \int_A f(x)dx \tag{2.42}$$

The transmission does not change the density of particle set, but assigns weight on each particle. We can also rewrite the Eq.(2.42) as

$$\begin{aligned}
&\left[\sum_{m=1}^{M} w^{[m]}\right]^{-1} \sum_{m=1}^{M} I(x^{[m]} \in A) w^{[m]} \\
=&\left[\sum_{m=1}^{M} w^{[m]}\right]^{-1} \sum_{m=1}^{M} I(x^{[m]} \in A) w^{[m]} M M^{-1} \\
=&\left[\sum_{m=1}^{M} w^{[m]}\right]^{-1} \sum_{m=1}^{M} w^{[m]} M I(x^{[m]} \in A) M^{-1}
\end{aligned} \tag{2.43}$$

From the Eq.(2.43), the factor M^{-1} can be seen as an average weight assigned to each particle and $w^{[m]}M$ number of particles are used as the weighted particles. Hence the number of particles after resampling is proportional to the weight. In the following, two common resampling schemes are introduced.

Naive sampler

To implement resampling, naive resampling method [60] provides a straightforward solution. It firstly generates a random number between $[0, 1]$ for every particle, then within the normalized cumulative weight c, searches for the weight range that the random value is placed at.

```
Naive-sampler (X̄_t, w_t) :
X_t = ∅; c^[0] = 0;
for m = 1 to M do
    c^[m] = c^[m−1] + w_t^[m] ;
end
for m = 1 to M do
    r = rand(0; 1) ;
    i = 0 ;
    while c^[i] < r and c^[i+1] > r do
        i = i + 1 ;
    end
    X_t = X_t+ < x̄_t^[i] >
end
return X_t
```

Algorithm 6: Naive sampler pseudo algorithm

Low variance sampler

The *low variance sampling* method [60] is more commonly used for resampling, Algorithm 7 depicts an implementation of a low variance sampler.

```
Low-variance-sampler (X̄_t, w_t) :
X_t = ∅ ;
r = rand(0; M^−1) ;
c = w_t^[1] ;
i = 1 ;
for m = 1 to M do
    u = r + (m − 1) ∗ M^−1 ;
    while u > c do
        i = i + 1 ;
        c = c + w_t^[i]
    end
    X_t = X_t+ < x̄_t^[i] >
end
return X_t
```

Algorithm 7: Low Variance Sampling pseudo algorithm

2.4 Least-Squares Estimation

The least-squares approach is widely used in various estimation problems by minimizing the sum of the squared errors (MSE).

2.4.1 Linear Least Squares

Consider a system of linear equations written in a vector form, $y = Mx + n_d$ where $\hat{y} = Mx$ is the ideal output vector, x represents system states, n_d is output error vector and M is the transformation matrix. With the given measurement y, and an information matrix S containing weights of uncertainty, the goal is to calculate the best estimation of states x^*, respect to the cost function J_y.

$$x^* = \arg\min_x J_y \tag{2.44}$$

where J_y is the sum of the weighted square of errors,

$$J_y = \frac{1}{2}(y - \hat{y})^\top S(y - \hat{y}) \tag{2.45}$$

To achieve an optimal estimation, the derivative of cost function at the optimal point equals zero and function needs to be convex up.

$$\frac{\partial J_y}{\partial x}|_{x^*} = 0$$
$$\frac{\partial^2 J_y}{\partial x^2}|_{x^*} > 0$$

The optimal solution for given output y can be readily calculated as follows,

$$\begin{aligned}
& \frac{\partial J_y}{\partial x}|_{x^*} = 0 \\
\Rightarrow \quad & \frac{1}{2}\left[-M^\top Sy - (y^\top SM)^\top + M^\top SMx^* + (x^* M^\top SM)^\top\right] = 0 \\
\Rightarrow \quad & -M^\top Sy + M^\top SMx^* = 0 \\
\Rightarrow \quad & x^* = (M^\top SM)^{-1} M^\top Sy
\end{aligned} \tag{2.46}$$

To test the convexity we have

$$\frac{\partial^2 J_y}{\partial^2 x}|_{x^*} = M^\top SM > 0 \tag{2.47}$$

2.4.2 Nonlinear Least Squares

Given a system of nonlinear equations with unknown states x, n number of nonlinear functions $f_i(.), i = 1, \ldots n$, and also n number of predicted measurements $\hat{y}_i$ and real measurement y_i

$$\hat{y}_i = f_i(x) \tag{2.48}$$

For example, the state x can be the robot's position which needs to be estimated, and for a 2-D case, the states contain position and angle information $x \in \mathbb{R}^3$. $\hat{y}_i$ can be the measurement from the robot position to the i_{th} landmark, which at least has the information of angle and distance, $\hat{y}_i \in \mathbb{R}^2$. y_i is the real transformation between the robot and the landmark, which can be calculated from GPS or other localization tools, in this case, $y_i \in \mathbb{R}^2$. The error function can be expressed as

$$e_i(x) = y_i - f_i(x) \tag{2.49}$$

And the cost function is the sum of squared error, which is

$$e_i'(x) = e_i(x)^\top S_i e_i(x) \tag{2.50}$$

$$F(x) = \sum_i e_i'(x) = \sum_i e_i(x)^\top S_i e_i(x) \tag{2.51}$$

Where $F(x)$ is the global cost, S_i is the information matrix, $e_i(x)$ is a scalar denotes the squared error of $\vec{e}_i(x)$. In order to find the best states estimation x^*, the following optimization needs to be solved:

$$x^* = \arg\min_x F(x) \tag{2.52}$$

Because of the nonlinearity, a good initial guess (x_0) is necessary to avoid a local minimum after the optimization procedure. An iterative local linearization is used to linearize error terms around the current guess. The first-order approximation of the error function is given as,

$$e_i(x + \Delta x) \approx e(x) + J_i(x)\Delta x \tag{2.53}$$

where $J_i(x)$ is the Jacobian,

$$J_i(x) = \frac{\partial e_i(x)}{\partial x} \tag{2.54}$$

Substitute Eq. (2.53) into Eq. (2.50) we get

$$\begin{aligned} e_i'(x) &\approx (e(x) + J_i(x)\Delta x)^\top S_i (e(x) + J_i(x)\Delta x) \\ &= e(x)^\top S e(x) + \left(\Delta x^\top J_i(x)^\top S_i e(x)\right)^\top + e(x)^\top S_i J_i(x)\Delta x \\ &+ \Delta x^\top J_i(x)^\top S_i J_i(x)\Delta x \\ &\approx e(x)^\top S e(x) + \Delta x^\top J_i(x)^\top S_i J_i(x)\Delta x + 2e(x)^\top S_i J_i(x)\Delta x \\ &\approx C_i + 2b_i^\top \Delta x + \Delta x^\top H_i \Delta x \end{aligned} \tag{2.55}$$

where,

$$C_i = e(x)^\top S e(x) \tag{2.56}$$

$$H_i = J_i(x)^\top S_i J_i(x) \tag{2.57}$$

$$b_i^\top = e(x)^\top S_i J_i(x) \tag{2.58}$$

Then subbing in Eq. (2.55) of global cost function to obtain:

$$\begin{aligned} F(x+\Delta x) &= \sum_i e_i'(x) \\ &= \sum_i C_i + 2b_i^\top \Delta x + \Delta x^\top H_i \Delta x \\ &= \sum_i C_i + 2(\sum_i b_i^\top)\Delta x + \Delta x^\top (\sum_i H_i)\Delta x \end{aligned} \tag{2.59}$$

For further simplicity, we use

$$C = \sum_i C_i$$

$$H = \sum_i H_i$$

$$b = \sum_i b_i^\top$$

$$F(x+\Delta x) = C + 2b^\top \Delta x + \Delta x^\top H \Delta x \tag{2.60}$$

Finally, by applying the simplified global cost, and setting its first order derivative to zero, we obtain

$$\begin{aligned} \frac{\partial F(x+\Delta x)}{\partial \Delta x} &= 2b + 2H\Delta x = 0 \\ H\Delta x &= -b \\ \Delta x^* &= -H^{-1}b \end{aligned} \tag{2.61}$$

Finally, the optimal state estimation is equal to

$$x^* = x + \Delta x^* = x - H^{-1}b \tag{2.62}$$

where x is the estimated value from the last iteration when implemented.

2.5 Robot Motion Model

In this section, the kinetic model of a two wheel ground robot is introduced. This dynamic model is used in subsequent chapters for various tests and simulations. Assume that a robot moves from a pose $A = \begin{bmatrix} x_t & y_t & \theta_t \end{bmatrix}$ to $B = \begin{bmatrix} x_{t+1} & y_{t+1} & \theta_{t+1} \end{bmatrix}$, see Fig. 2.2. Pose A and B share a center C, around which the robot rotates. Let the distance between the wheels be D and the distance from C to the center of the robot be R. Finally, two wheels have the same angular velocity w with respect to C, which is also the robot motion angular velocity. From above, we obtain the following equations,

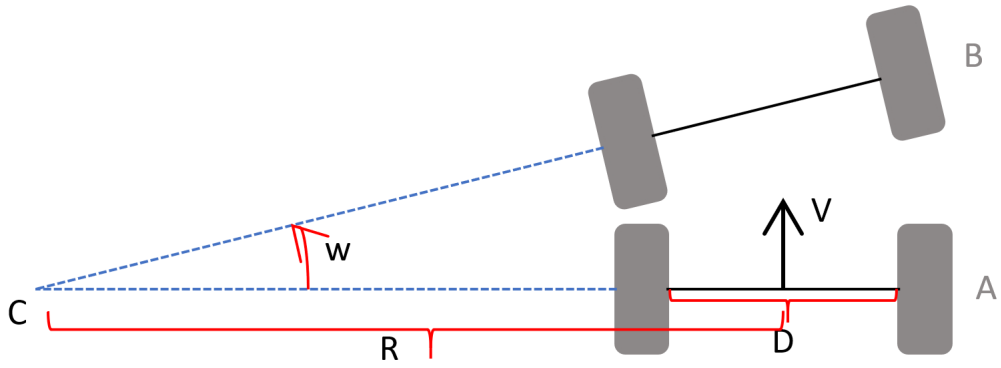

Figure 2.2: Two wheels ground robot motion

$$\begin{cases} w(R - \frac{D}{2}) = v_l \\ w(R + \frac{D}{2}) = v_r \end{cases} \tag{2.63}$$

Where v_l and v_r denote the linear velocities of the left and right wheel, respectively. Let v denote the linear velocity of the robot. We can rewrite Eq.(2.63) as the following,

$$\begin{aligned} v_l(R + \frac{D}{2}) &= v_r(R - \frac{D}{2}) \\ R &= \frac{D(v_l + v_r)}{2(v_r - v_l)} = \frac{v}{w} \end{aligned} \tag{2.64}$$

where

$$\begin{aligned} v_r - v_l &= wD \quad \text{or} \\ w &= \frac{v_r - v_l}{D} \end{aligned} \tag{2.65}$$

and

$$v = wR = \frac{v_l + v_r}{2} \tag{2.66}$$

When the distance from A to B is short, the transformation between pose A and B can be approximated by:

$$\begin{aligned} \Delta x &\approx v \cos\theta_t \Delta t \\ \Delta y &\approx v \sin\theta_t \Delta t \end{aligned} \tag{2.67}$$

where θ_t is the angle. Then the approximated motion model can be expressed as:

$$\begin{aligned} \begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} &= \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} v\cos\theta_t \Delta t \\ v\sin\theta_t \Delta t \\ w\Delta t \end{bmatrix} \\ w = \frac{v_r - v_l}{D}, &\quad v = \frac{v_l + v_r}{2} \end{aligned} \tag{2.68}$$

The approximation in Eq.(2.67) only valid when the robot motion satisfies the small signal assumption, meaning that a small change of velocity causes a small change of distance, and a better assumption has to be made if the approximation does not stand. In this case, R and C are constant, which can be used to calculate the position of the center point C:

$$\begin{aligned} C_x = x_t - R\sin\theta_t, &\quad d_{x_t} = x_t - C_x = R\sin\theta_t \\ C_y = y_t + R\cos\theta_t, &\quad d_{x_t} = y_t - C_y = -R\cos\theta_t \end{aligned}$$

d_{x_t} and d_{y_t} denote the positions along the coordination system where origin is at C. While the angular velocity w is the input, the pose B can be computed by the rotation transformation:

$$\begin{bmatrix} d_{x_{t+1}} \\ d_{y_{t+1}} \end{bmatrix} = \begin{bmatrix} \cos w\Delta t & -\sin w\Delta t \\ \sin w\Delta t & \cos w\Delta t \end{bmatrix} \begin{bmatrix} d_{x_t} \\ d_{y_t} \end{bmatrix} \tag{2.69}$$

Finally, we obtain the motion model Eq. (2.70) with inputs as the velocity v and angular velocity w:

$$
\begin{aligned}
\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} &= \begin{bmatrix} \cos w\Delta t & -\sin w\Delta t & 0 \\ \sin w\Delta t & -\cos w\Delta t & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_{x_t} \\ d_{y_t} \\ \theta_t \end{bmatrix} + \begin{bmatrix} C_x \\ C_y \\ w\Delta t \end{bmatrix} \\
&= \begin{bmatrix} \cos(w\Delta t)R\sin(\theta_t) + \sin(w\Delta t)R\cos(\theta_t) \\ \sin(w\Delta t)R\sin(\theta_t) - \cos(w\Delta t)R\cos(\theta_t) \\ \theta \end{bmatrix} + \begin{bmatrix} x_t - R\sin\theta_t \\ y_t + R\cos\theta_t \\ w\Delta t \end{bmatrix} \\
&= \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} -\frac{v}{w}\sin\theta_t + \frac{v}{w}\sin(\theta_t + w\Delta t) \\ \frac{v}{w}\cos\theta_t - \frac{v}{w}\cos(\theta_t + w\Delta t) \\ w\Delta t \end{bmatrix}
\end{aligned}
\tag{2.70}
$$

There are other motion dynamics, such as the four wheel model, etc. In practice, we can directly adjust motor current or voltage to control its speed. To estimate the robot motion, the motor encoder values are usually used to compute the robot odometry. The encoder data contains the motor or wheel's rotation angle, and with the use of the rotation angle and wheels dimensions, we can estimate how far and which direction the robot has traveled and compute the new robot location.

Chapter 3

Filter-based SLAM

There are three main features in SLAM, namely mapping, localization, and planning. When a robot drives in an unknown environment, the mapping algorithm uses sensor collected environment data to generate different maps, for example, topological map or grid map. The localization algorithm is for the robot to estimate its position in a given map. The robot uses combined information from localization and mapping as the foundation for navigation. When the map and locations are known, the navigation algorithm calculates the best path to the destination. In this chapter, map representations and measurement models are at first given in the 2-D case, and then how they are used in filter based SLAM algorithms is discussed. EKF, UKF, and particle filter based SLAM are introduced. Finally MATLAB simulation results of EKF and UKF SLAM are reported.

3.1 The Map

In a 2-D scenario, the mathematical representation of the map and robot position are introduced in two ways. One approach is to combine the robot position and detected landmarks in one map vector. This representation is commonly used when object features are used for sensor measurement. Because of the limited feature number (much less than sensor observations), it is possible to use all of features during the calculation. Another approach is to use occupancy grid map, in which each grid cell is associated with the probability of the cell being occupied.

3.1.1 Feature Map

Let a 2-D map M contain the robot and n landmarks states, and the robot state has its location x, y, and orientation θ. The i_{th} landmark states include its location L_x^i with respect to the x-axis and L_y^i to the y-axis. The following are vector representations of the map states M and its covariance matrix P

$$M = \begin{bmatrix} R \\ L \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \\ L_x^1 \\ L_y^1 \\ \cdots \\ L_x^n \\ L_y^n \end{bmatrix} \tag{3.1}$$

The error covariance matrix P has dimension of $(3 + 2n) \times (3 + 2n)$ and is written as

$$P = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xL_x^1} & \sigma_{xL_y^1} & \cdots & \sigma_{xL_x^n} & \sigma_{xL_y^n} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} & \sigma_{yL_x^1} & \sigma_{yL_y^1} & \cdots & \sigma_{yL_x^n} & \sigma_{yL_y^n} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} & \sigma_{\theta L_x^1} & \sigma_{\theta L_y^1} & \cdots & \sigma_{\theta L_x^n} & \sigma_{\theta L_y^n} \\ \sigma_{L_x^1 x} & \sigma_{L_x^1 y} & \sigma_{L_x^1 \theta} & \sigma_{L_x^1 L_x^1} & \sigma_{L_x^1 L_y^1} & \cdots & \sigma_{L_x^1 L_x^n} & \sigma_{L_x^1 L_y^n} \\ \sigma_{L_y^1 x} & \sigma_{L_y^1 y} & \sigma_{L_y^1 \theta} & \sigma_{L_y^1 L_x^1} & \sigma_{L_y^1 L_y^1} & \cdots & \sigma_{L_y^1 L_x^n} & \sigma_{L_y^1 L_y^n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sigma_{L_x^n x} & \sigma_{L_x^n y} & \sigma_{L_x^n \theta} & \sigma_{L_x^n L_x^1} & \sigma_{L_x^n L_y^1} & \cdots & \sigma_{L_x^n L_x^n} & \sigma_{L_x^n L_y^n} \\ \sigma_{L_y^n x} & \sigma_{L_y^n y} & \sigma_{L_y^n \theta} & \sigma_{L_y^n L_x^1} & \sigma_{L_y^n L_y^1} & \cdots & \sigma_{L_y^n L_x^n} & \sigma_{L_y^n L_y^n} \end{bmatrix}$$

or simply

$$P = \begin{bmatrix} P_{RR} & P_{RL} \\ P_{LR} & P_{LL} \end{bmatrix} \tag{3.2}$$

3.1.2 Occupancy Grid Map

It is sometimes difficult to use the feature map to represent objects. First of all, features of a uniform structure tend to be complex. Furthermore, the four most important factors cannot be effectively handled by the feature map [60]. First, when the robot is exposed to a broader environment, it is more challenging and slower to maintain the map. Second, the considerable noises from sensors or the environment affect the result significantly. The third factor is the perceptual ambiguity, the map should have more confidence when the

sensor data matches the map information. Lastly, the accumulated odometry error can generate a wrong map. The occupancy grid map can address these four problems well. The basic idea of the occupancy grid map is to arrange the map into a number of grids and use probability to present the occupied grid's confidence. The occupancy grid map representation is very convenient for post SLAM processes. For example, after the EKF SLAM, we can use the maximum likelihood of grids to optimize the robot pose. Also, some SLAM algorithms directly use the inverse sensor model to update the occupancy grid map, for example, Monte Carlo localization [11], etc.

The probability of an occupancy grid map M is computed as the sum of the probability from each grid cell M_i. The probability of a grid been hit p_{hit} or missed p_{miss} can be computed by the inverse of the measurement model. Another way is to assign the probability p_{hit} or p_{miss} to grid points been hit or missed. For example, Fig. 3.1 demonstrates the hits and misses for a 2D-scan. For every hit, the closest grid cell is inserted to the hit set, and then missed grids are those intersecting the rays between the scan origin and the hit point.

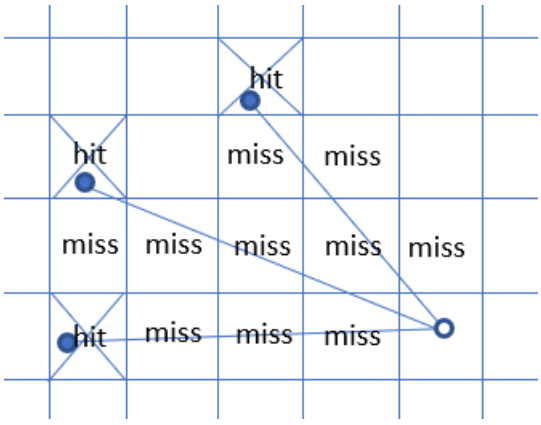

Figure 3.1: A scan and grid map associated with hits and misses

The grid cells probability is updated by the following steps:

$$\text{odds}(p) = \frac{p}{1-p} \tag{3.3}$$

$$M_{new}(x) = \text{clamp}(\text{odds}^{-1}(\text{odds}(M_{old}(x)) \cdot \text{odds}(p_{hit}))) \tag{3.4}$$

where the grid point x has been observed and we update the old grid probability $M_{old}(x)$ by the odds for hits or misses, and the function clamp(x) does the following:

$$\text{if } (x > p_{max}), \quad x = p_{max}; \quad \text{if } (x < p_{min}), \quad x = p_{min}$$

3.2 EKF-SLAM

One common solution to the 2D SLAM problem is based on EKF , see Algorithm 3. In this section, EKF based SLAM for the 2D case is discussed. The feature map is used to implement EKF-SLAM, and it is assumed that the data association is known for landmarks, and every landmark is independent.

3.2.1 Initialization and New Landmark Observation

Let the robot is initially placed at the origin of the map with no uncertainty, where the robot states R and covariance matrix P_{RR} are initialized as zero vector and matrix.

$$R_0 = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} = \begin{bmatrix} \emptyset \\ \emptyset \\ \emptyset \end{bmatrix} \quad P_{RR_0} = \begin{bmatrix} \emptyset \end{bmatrix} \tag{3.5}$$

Assume landmarks' positions are unknown initially, so the landmarks' states and the covariance between the robot and landmark P_{RL} and P_{LR} are set to zeros. Also, the landmark auto-covariance matrix is set to be infinite identity and covariance between landmarks to be zero because landmarks are assumed to be independent of each other, i.e.

$$L = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}^\top \quad P_{LL} = \begin{bmatrix} \infty & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \infty \end{bmatrix} \tag{3.6}$$

Landmarks are not observed initially. When a new landmark is observed, a fixed covariance can be assigned to the landmark. Another way to incorporate a new landmark is to recalculate the landmark states and covariance upon its observation. Also, the map and covariance matrix dimensions expand whenever a new landmark is observed. For the robot considered in this chapter,

when the sensor detects a new landmark, it returns values of the range and the angle. Received range and angle can be used to calculate map states. The landmark observation function can be written as:

$$\hat{y} = h(M) \tag{3.7}$$

where the Map M includes the robot states and landmark states: $M = [R^\top \quad L^\top]^\top$. The new landmark is computed from the reverse of the observation function, which can be expressed as,

$$L_n = g(R, \hat{y}) \tag{3.8}$$

After the n_{th} new landmark is added to M then the new map states become,

$$M_{new} = \begin{bmatrix} M_{old} \\ L_n \end{bmatrix} \tag{3.9}$$

The new covariance matrix becomes,

$$P_{new} = \begin{bmatrix} P_{old} & P_{X_n L_n} \\ P_{L_n X_n} & P_{L_n L_n} \end{bmatrix} \tag{3.10}$$

The auto-covariance of landmark states $P_{L_n L_n}$ and the covariance between new landmark sates and existing states $P_{L_n M_{old}}$ are appended to the existing matrix.

For convenience, let the $X_n = [R^\top \ \hat{y}^\top]^\top$, where R contains the robot states and we use sensor measurement z here to approximate $\hat{y} \approx z$. The landmark auto-covariance can be computed by following

$$P_{L_n L_n} = E[(g(X_n) - \bar{g}(X_n))(g(X_n) - \bar{g}(X_n))^\top]$$

$\bar{g}(.)$ is the mean of the output function g. By using the first-order approximation of the Taylor's expansion, we can compute the landmark location and its mean as:

$$L_n = g(X_n) = g(\bar{X}_n + \delta X_n) = g(\bar{X}_n) + \nabla g(\bar{X}_n)\delta X_n + \cdots$$

$$\bar{L}_n = \bar{g}(X_n) = E[g(\bar{X}_n) + \nabla g(\bar{X}_n)\delta X_n + \cdots] \approx g(\bar{X}_n)$$

The covariance matrix can be rewritten and approximated as:

$$\begin{aligned} P_{L_nL_n} &= E[(g(X_n) - \bar{g}(X_n))(g(X_n) - \bar{g}(X_n))^\top] \\ &\approx E[\nabla g(\bar{X}_n)(\delta X_n(\delta X_n)^\top)\nabla g(\bar{X}_n)^\top] \\ &= \nabla g(\bar{X}_n) P_{X_nX_n} \nabla g(\bar{X}_n)^\top \end{aligned} \tag{3.11}$$

The gradient of X_n is

$$\nabla g(\bar{X}_n) = \begin{bmatrix} \frac{\partial g}{\partial R} & \frac{\partial g}{\partial y} \end{bmatrix} \tag{3.12}$$

The covariance matrix $P_{X_nX_n}$ can be extend to:

$$P_{X_nX_n} = \begin{bmatrix} P_{RR} & P_{Ry} \\ P_{yR} & P_{yy} \end{bmatrix} \tag{3.13}$$

Furthermore, assume that new landmark's observation and robot states are independent, which means $P_{Ry} = P_{yR} = 0$. Also, the covariance for the measurement P_{yy} can be assigned as the measurement error covariance V. The covariance $P_{L_nL_n}$ for new landmark states becomes

$$\begin{aligned} P_{L_nL_n} &= \nabla g(\bar{X}_n) P_{X_nX_n} \nabla g(\bar{X}_n)^\top \\ &= \begin{bmatrix} \frac{\partial g}{\partial R} & \frac{\partial g}{\partial y} \end{bmatrix} \begin{bmatrix} P_{RR} & \\ & P_{yy} \end{bmatrix} \begin{bmatrix} \frac{\partial g}{\partial R} \\ \frac{\partial g}{\partial y} \end{bmatrix} \\ &= \frac{\partial g}{\partial R} P_{RR} \frac{\partial g}{\partial R}^\top + \frac{\partial g}{\partial y} V \frac{\partial g}{\partial y}^\top \end{aligned} \tag{3.14}$$

Finally, the covariance between the new landmark and robot is:

$$P_{L_nX_n} = \frac{\partial g}{\partial R} P_{RL} \tag{3.15}$$

$$P_{X_nL_n} = P_{L_nX_n}^\top \tag{3.16}$$

3.2.2 Prediction

Consider that landmarks are stationary, then the map M_t at time t is:

$$M_t = F(M_{t-1}, u_t, n_t) = \begin{bmatrix} f(R_{t-1}, u_t, n_t) \\ L_t \end{bmatrix}$$

$$y_t = h(M_t) + \sigma_t$$

where u, n, and σ are the input, process noise, and the measurement noise, respectively. The EKF prediction step from the Algorithm 3 is expressed as follows,

$$\bar{M}_t = F(M_{t-1}, u_t) \tag{3.17}$$
$$\bar{P}_t = A_t P_{t-1} A_t^\top + N_t Q_t N_t^\top$$

The linearized transformation model A is invariant to landmarks and can be written as:

$$A_t = \frac{\partial F(M,u,n)}{\partial R}\bigg|_{M_{t-1}} = \begin{bmatrix} \frac{\partial f(R,u,n)}{\partial R} & 0 \\ 0 & \mathbf{I} \end{bmatrix}\bigg|_{R_{t-1}} \tag{3.18}$$

where Q is covariance matrix of the perturbation n and

$$N_t^\top = \frac{\partial F(M)}{\partial n} = \begin{bmatrix} \frac{\partial f(R,u,n)}{\partial n} \\ 0 \end{bmatrix}$$

Finally, substitute A_t, P_{t-1} and N_t into the error covariance matrix,

$$\begin{aligned}
\bar{P}_t &= A_t \; P_{t-1} \; A_t^\top + N_t Q_t N_t^\top \\
&= \begin{bmatrix} \frac{\partial f(R,u,n)}{\partial R} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \begin{bmatrix} P_{RR} & P_{RL} \\ P_{LR} & P_{LL} \end{bmatrix} \begin{bmatrix} \frac{\partial f(R,u,n)}{\partial R} & 0 \\ 0 & \mathbf{I} \end{bmatrix}^\top + \begin{bmatrix} \frac{\partial f(R,u,n)}{\partial n} \\ 0 \end{bmatrix}^\top Q \begin{bmatrix} \frac{\partial f(R,u,n)}{\partial n} \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial f(R,u,n)}{\partial R} P_{RR} \frac{\partial f(R,u,n)}{\partial R}^\top & \frac{\partial f(R,u,n)}{\partial R} P_{RL} \\ (\frac{\partial f(R,u,n)}{\partial R} P_{LR})^\top & P_{LL} \end{bmatrix} + \left(\frac{\partial f(R,u,n)}{\partial n} \right)^\top Q_{RR} \frac{\partial f(R,u,n)}{\partial n}
\end{aligned} \tag{3.19}$$

3.2.3 Estimation

Use estimation step from EKF Algorithm 3,

$$K_t = \bar{P}_t H_t^\top (H_t \bar{P}_t H_t^\top + V_t)^{-1}$$

$$M_t = \bar{M}_t + K_t(z_t - h(\bar{M}_t))$$

$$P_t = (I - K_t H_t)\bar{P}_t$$

where the Jacobian $H_t = \frac{\partial h(M)}{\partial M}|_{M=\bar{M}_t}$, V is the covariance matrix for measurement noise and K is the Kalman gain. z_t represents the sensor measured value and $\hat{y}_t = h(\bar{M}_t)$ is the estimated measurement computed by observation model.

For SLAM problems, assume there are n numbers of different observations z_t

are collected and n number of measurement estimation $h(M)$ are computed. Therefore, the EKF filter updates n landmarks one by one during an update process. For example, when the i_{th} landmark is observed, the Jacobian matrix H becomes,

$$H = \begin{bmatrix} \frac{\partial h(M)}{\partial R} & 0 & \cdots & \frac{\partial h(M)}{\partial L^i} & \cdots & 0 & \cdots & 0 \end{bmatrix} \tag{3.20}$$

3.3 UKF-SLAM

In this section, the implementation of UKF for SLAM is discussed. Similarly, the feature map representation is used.

3.3.1 New Landmark Observation

Similar to EKF-SLAM, the observation model, formulation of states and landmark map can be obtained as in Eq. (3.8)-(3.10). In the following, we discuss how to compute the covariance matrices by using the unscented transformation. Applying the unscented transformation, the covariance $P_{L_nL_n}$ can be computed through the Eq.(2.27) for each landmark L_n,

$$\bar{P}_{L_nL_n} = \sum_{i=0}^{2n} W_i^c (\mathcal{X}_i^{L_n} - \bar{L}_{i,n})(\mathcal{X}_i^{L_n} - \bar{L}_{i,n})^\top \tag{3.21}$$

where

$$\begin{aligned} \mathcal{X}^{L_n} &= g(\mathcal{X}^{X_n}) \\ \bar{L}_n &= \sum_{i=0}^{2n} W_i^m \mathcal{X}_i^{L_n} \\ \mathcal{X}^{X_n} &= [\bar{X}_n \quad \bar{X}_n \pm (\sqrt{(n+\lambda)P_{X_nX_n}})_1 \ \cdots \ \bar{X}_n \pm (\sqrt{(n+\lambda)P_{X_nX_n}})] \end{aligned} \tag{3.22}$$

Given the measurement noise covariance Q, and assume the independency between robot and observation, the covariance $P_{X_nX_n}$ can be written as

$$P_{X_nX_n} = \begin{bmatrix} P_{RR} & \emptyset \\ \emptyset & Q \end{bmatrix} \tag{3.23}$$

To calculate the $\bar{P}_{X_nL_n}$, start with equations:

$$\bar{P}_{XL_n} = \sum_{i=0}^{2n} W_i^m (\mathcal{X}_i^{X_n} - \bar{X}_{i,n})(\mathcal{X}_i^{L_n} - \bar{L}_{i,n})^\top \tag{3.24}$$

where

$$\mathcal{X}^{L_n} = [\bar{L}_n \quad \bar{L}_n \pm (\sqrt{(n+\lambda)P_{L_nL_n}})_1 \ \cdots \ \bar{L}_n \pm (\sqrt{(n+\lambda)\bar{P}_{L_nL_n}})n] \quad (3.25)$$

Where the $\bar{P}_{L_nL_n}$ is calculated from Eq.(3.21).

3.3.2 Prediction

Assume landmarks are not moving all the time, the prediction step will only update the robot states and leave landmarks states not changed. Use the robot covariance matrix P_{RR} to generate sigma states

$$\mathcal{X}^{R_{t-1}} = [\bar{R}_{t-1} \quad \bar{R}_{t-1} \pm (\sqrt{(n+\lambda)P_{RR}})_1 \ \cdots \ \bar{R}_{t-1} \pm (\sqrt{(n+\lambda)P_{RR}})_n]$$

where n is the size of covariance matrix, and for the robot states $n = 3$. The weights are calculated through Eq.(2.28). The prior states are computed by the following equation:

$$\mathcal{X}^{R_t} = f(\mathcal{X}^{R_{t-1}}, u_t) \quad (3.26)$$

Its average is

$$\bar{R}_t = \sum_{i=1}^{n} W_{m,i}\mathcal{X}^{R_t} \quad (3.27)$$

Then apply the Eq.(2.32) and add the control input noise to predict the UKF covariance $\bar{P}_{RR}$:

$$\bar{P}_{RR} = \sum_{i=0} W_c(\mathcal{X}^{R_t} - \bar{R}_t)(\mathcal{X}^{R_t} - \bar{R}_t)^\top + \sum_{i=0} W_c(\mathcal{X}^{u_t} - u_t)(\mathcal{X}^{u_t} - u_t)^\top + Q_t \quad (3.28)$$

Where for convenience we combine the input noise with the process noise to estimate $Q = \sum_{i=0} W_c(\mathcal{X}^{u_t} - u_t)(\mathcal{X}^{u_t} - u_t)^\top + Q_t$ The P_{RL} and P_{LR} can be computed in two ways. First approach simply set P_{RL} and P_{LR} to be zeros. The better one is to apply UT to calculate P_{RL} and P_{LR}.

$$\bar{P}_{RL} = E[(\mathcal{X}^{R_t} - \bar{R}_t)(\mathcal{X}^{L} - \bar{L})^\top] \quad (3.29)$$

Lastly, the landmarks covariance are kept same

$$\bar{P}_{LL} = P_{LL} \quad (3.30)$$

3.3.3 Estimation

During the estimation step, the observation equation takes the sigma points from all changing states (robot and observed landmark states). The computed prior covariance matrix will be used for sigma points calculation.

$$\mathcal{X}^{M_t} = [\bar{M}_t \quad \bar{M}_t \pm (\sqrt{(n+\lambda)\bar{P}})_1 \ \cdots \ \bar{M}_t \pm (\sqrt{(n+\lambda)\bar{P}})_n]$$

The sigma points of the observation are,

$$\mathcal{X}^y = h(\mathcal{X}^{M_t})$$

and its mean is

$$\bar{y} = \sum_{i=0}^{2n} W_m \mathcal{X}^y$$

the output error covariance can be computed as

$$P_{yy} = \sum_{i=0}^{2n} W_c(\mathcal{X}^y - \bar{y})(\mathcal{X}^y - \bar{y})^\top + V$$

where V is the measurement noise covariance matrix. Next, the covariance P_{yM_t} between the prior map states $\bar{M}_t$ and measurement y.

$$P_{yM_t} = \sum_{i=0}^{2n} W_c(\mathcal{X}^y - \bar{y})(\mathcal{X}^{M_t} - \bar{M}_t)^\top$$

$$K = P_{yM_t} P_{yy}^\top$$

Lastly, the updated map M_t and covariance P_t becomes:

$$M_t = \bar{M}_t + K(z_t - \bar{y}) \tag{3.31}$$

$$P_t = \bar{P} + K P_{yy} K^\top \tag{3.32}$$

3.4 Simulation of EKF/UKF-SLAM

The simulation for EKF-SLAM and UKF-SLAM is developed in Matlab. Numbers of landmarks are generated, and a robot drives in a circular motion using

the two-wheel ground robot motion model described in Chapter 2. During the simulation, robot paths and landmarks are estimated using EKF, UKF and the pure motion model. In addition, the trajectory and landmark error are computed and presented. The Matlab program can be founded at `https://github.com/linjianxiang/SLAM_simulations`.

3.4.1 Two Wheel Ground Robot

In the simulation, the two-wheel ground robot motion model given in Eq. (2.70) is used. This subsection describes how robot model is integrated into the feature map. With control input $u_t = (v_t \ w_t)^\top$, the motion model $f(x_{t-1}, u_t)$ is given as,

$$R = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = f(x_{t-1}, u_t) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} -\frac{v_t}{w_t}\sin(\theta_{t-1}) + \frac{v_t}{w_t}\sin(\theta_{t-1} + w_t\Delta t) \\ \frac{v_t}{w_t}\cos(\theta_{t-1}) - \frac{v_t}{w_t}\cos(\theta_{t-1} + w_t\Delta t) \\ w_t\Delta t \end{bmatrix}$$

The map M is represented by the following equation,

$$M_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \\ L^1_{xt} \\ L^1_{yt} \\ \vdots \\ L^n_{xt} \\ L^n_{yt} \end{bmatrix} = F(M_{t-1}, u_t) = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \\ L^1_{xt-1} \\ L^1_{yt-1} \\ \vdots \\ L^n_{xt-1} \\ L^n_{yt-1} \end{bmatrix} + F_x \begin{bmatrix} -\frac{v_t}{w_t}\sin(\theta_{t-1}) + \frac{v_t}{w_t}\sin(\theta_{t-1} + w_t\Delta t) \\ \frac{v_t}{w_t}\cos(\theta_{t-1}) - \frac{v_t}{w_t}\cos(\theta_{t-1} + w_t\Delta t) \\ w_t\Delta t \end{bmatrix}$$

In this case, we assume all landmarks are stationary, but only robot states change with time. To reduce computation burden involving matrix calculations, the matrix F_x is introduced to ensure only changed states are used for calculations, which saves a significant amount of computation time,

$$F_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix} \tag{3.33}$$

Furthermore, let the estimated output observation be $\hat{y}_t = h(M_t) = [r \quad \phi]^\top$ and the sensor measurement equation is shown as follow:

$$y_t = h(M_t) + \sigma_t$$

Initialization is conducted according to Eq. (3.5) and (3.6).

Prediction for Two-Wheel Ground Robot

To perform the EKF prediction in Eq.(3.17) and (3.19), we need to calculate the Jacobian matrix A_t and covariance matrix. From two-wheel robot motion kinetic model,

$$\begin{aligned} A_t = \frac{\partial f(R,u,n)}{\partial R} &= \frac{\partial}{\partial (x,y,\theta)^\top}\left(\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} -\frac{v}{w}\sin(\theta) + \frac{v}{w}\sin(\theta + w\Delta t) \\ \frac{v}{w}\cos(\theta) - \frac{v}{w}\cos(\theta + w\Delta t) \\ w\Delta t \end{bmatrix}\right) \\ &= \mathbf{I} + \begin{bmatrix} 0 & 0 & -\frac{v}{w}\cos(\theta) + \frac{v}{w}\cos(\theta + w\Delta t) \\ 0 & 0 & -\frac{v}{w}\sin(\theta) + \frac{v}{w}\sin(\theta + w\Delta t) \\ 0 & 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & -\frac{v}{w}\cos(\theta) + \frac{v}{w}\cos(\theta + w\Delta t) \\ 0 & 1 & -\frac{v}{w}\sin(\theta) + \frac{v}{w}\sin(\theta + w\Delta t) \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3.34}$$

For simplicity, we redefine the motion noise covariance as

$$Q = \left(\frac{\partial f(R,u,n)}{\partial n}\right)^\top Q_{RR} \frac{\partial f(R,u,n)}{\partial n} \tag{3.35}$$

The prediction step can then be performed as shown in Eq.(3.17) and (3.19) and EKF Algorithm3.

Estimation for Two-Wheel Ground Robot

The sensor model observes the i_{th} landmark at the time t, i.e.

$$y_t^i = [r_t^i \quad \phi_t^i]^\top$$

The first time the sensor observes the i_{th} landmark (range and angle), landmark states (navigation frame location) can be calculated by the reverse observation function:

$$\begin{bmatrix} L_x^i \\ L_y^i \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \end{bmatrix} + \begin{bmatrix} r_t^i \ \cos(\phi_t^i + \theta_t) \\ r_t^i \ \sin(\phi_t^i + \theta_t) \end{bmatrix} \tag{3.36}$$

In the Eq.(3.36), r_t^i and ϕ_t^i are measured range and angle value, position x_t, y_t and orientation θ_t are estimated robot states at time t. The estimated output for i_{th} landmark at time t can be calculated by:

$$h(M_t) = \begin{bmatrix} r_t^i \\ \phi_t^i \end{bmatrix} = \begin{bmatrix} \sqrt{(L_x^i - x_t)^2 + (L_y^i - y_t)^2} \\ \arctan((L_y^i - y_t)/(L_x^i - x_t)) - \theta_t \end{bmatrix} \tag{3.37}$$

After having the measurement model, the output Jacobian matrix H can be calculated,

$$H = \begin{bmatrix} \frac{\partial h(M)}{\partial R} & 0 & \cdots & \frac{\partial h(M)}{\partial L^i} & 0 & \cdots & 0 \end{bmatrix}$$

where

$$\frac{\partial h(M_t)}{\partial R} = \frac{\partial h(M_t)}{\partial (x_t, y_t, \theta_t)} = \begin{bmatrix} -\frac{(L_x^i - x_t)}{\sqrt{(L_x^i - x_t)^2 + (L_y^i - y_t)^2}} & -\frac{(L_y^i - y_t)}{\sqrt{(L_x^i - x_t)^2 + (L_y^i - y_t)^2}} & 0 \\ \frac{L_y^i - y_t}{(L_x^i - x_t)^2 + (L_y^i - y_t)^2} & -\frac{L_x^i - x_t}{(L_x^i - x_t)^2 + (L_y^i - y_t)^2} & -1 \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{(L_x^i - x_t)}{r_t^i} & -\frac{(L_x^i y - y_t)}{r_t^i} & 0 \\ \frac{L_y^i - y_t}{(r_t^i)^2} & -\frac{L_x^i - x_t}{(r_t^i)^2} & -1 \end{bmatrix} \tag{3.38}$$

$$\frac{\partial h(M_t)}{\partial L^i} = \frac{\partial h(M_t)}{\partial (L_x^i, L_y^i)} = \begin{bmatrix} \frac{(L_x^i - x_t)}{\sqrt{(L_x^i - x_t)^2 + (L_y^i - y_t)^2}} & \frac{(L_y^i - y_t)}{\sqrt{(L_x^i - x_t)^2 + (L_y^i - y_t)^2}} \\ -\frac{L_y^i - y_t}{(L_x^i - x_t)^2 + (L_y^i - y_t)^2} & \frac{L_x^i - x_t}{(L_x^i - x_t)^2 + (L_y^i - y_t)^2} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{(L_x^i - x_t)}{r_t^i} & \frac{(L_y^i - y_t)}{r_t^i} \\ -\frac{L_y^i - y_t}{(r_t^i)^2} & \frac{L_x^i - x_t}{(r_t^i)^2} \end{bmatrix} \tag{3.39}$$

Substitute the Eq.(3.38) and Eq.(3.39) into H

$$H = \begin{bmatrix} -\frac{(L_x^i - x_t)}{r_t^i} & -\frac{(L_x^i y - y_t)}{r_t^i} & 0 & \cdots & \frac{(L_x^i - x_t)}{r_t^i} & \frac{(L_x^i y - y_t)}{r_t^i} & \cdots \\ \frac{L_y^i - y_t}{(r_t^i)^2} & -\frac{L_x^i - x_t}{(r_t^i)^2} & -1 & \cdots & -\frac{L_y^i - y_t}{(r_t^i)^2} & \frac{L_x^i - x_t}{(r_t^i)^2} & \cdots \end{bmatrix}$$

To reduce the computational time, the modified algorithm uses a transformation matrix F_x to only update the robot and current observed landmark states,

$$F_x = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \end{bmatrix}^\top$$

The matrix F_x has dimension of $2n \times 5$ and the i_{th} landmark observation is located at the $(3 + 2(i-1))_{th}$ row of the matrix. (i starts from 0)

$$HF_x = \begin{bmatrix} -\frac{(L_x^i - x_t)}{r_t^i} & -\frac{(L_x^i y - y_t)}{r_t^i} & 0 & \frac{(L_x^i - x_t)}{r_t^i} & \frac{(L_x^i y - y_t)}{r_t^i} \\ \frac{L_y^i - y_t}{(r_t^i)^2} & -\frac{L_x^i - x_t}{(r_t^i)^2} & -1 & -\frac{L_y^i - y_t}{(r_t^i)^2} & \frac{L_x^i - x_t}{(r_t^i)^2} \end{bmatrix}$$

Up to this point, equations for applying EKF-SLAM algorithm to motion model (Eq.(2.70)) are derived.

3.4.2 Parameters

There are many parameters to be tuned in the simulation and some important ones are listed as follows.

- **iteration**: number of iterations that the robot runs
- **dt**: time interval
- **map.map_length**: length of the squared map
- **map.landmark_number**: number of landmarks
- **map.random_landmark**: randomize or fixed landmarks
- **q**: control noise standard deviation
- **v**: measurement noise standard deviation
- **u_v**: robot velocity
- **u_w**: robot angular velocity

3.4.3 Results

To test and compare performance of the EKF and UKF schemes, simulation is performed in two experiments, and results are obtained.

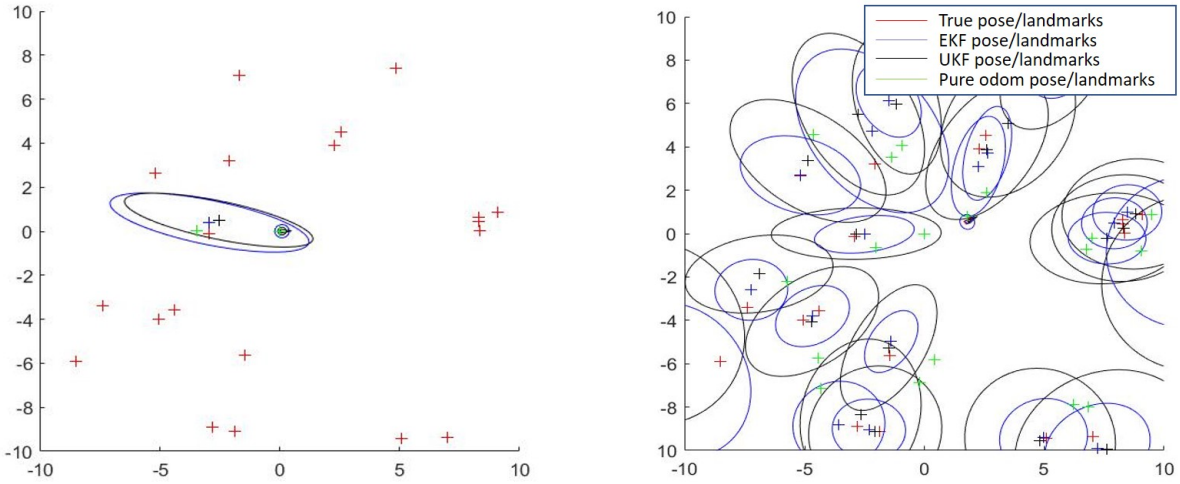

Figure 3.2: The plot on the left shows estimation results at the time when only one landmark is observed. The right one shows the iteration when landmarks are observed.

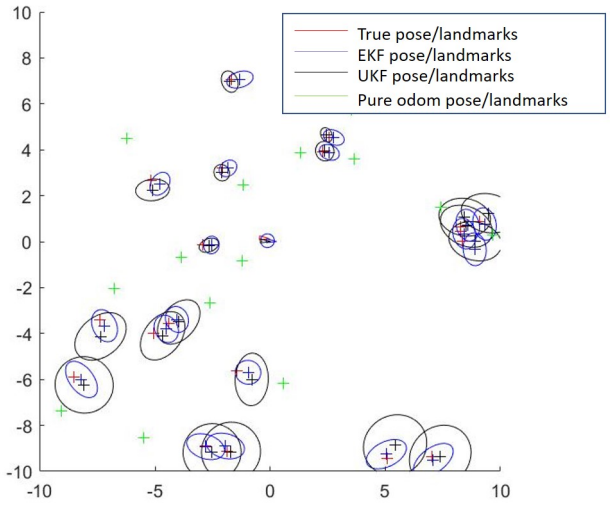

Figure 3.3: Robot and landmark position after 200 iteration

Experiment 1

The first experiment runs 200 iterations and the robot input speed $u\_v = 1$, angular velocity $u\_w = \pi/10$. Also, there are 20 landmarks used, and noise standard deviations are set as $q = [0.1; \pi/18]$, $v = [2, \pi/18]$ for control and measurement respectively. In the beginning, no landmarks are observed.

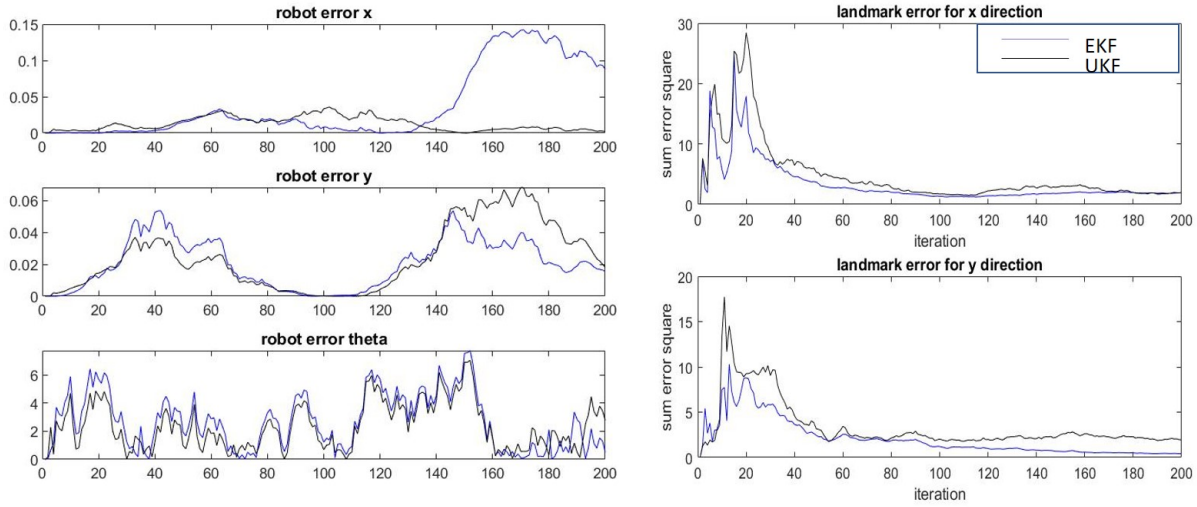

Figure 3.4: Estimated robot position and landmark position error

For each iteration, one landmark is added to the states until all of them are detected. For example, the left of the Fig.3.2 shows the beginning of the SLAM process when only one landmark is observed. The right image from the Fig.3.2 also shows how confident the landmark location is in the early stage. The ellipse represents the covariance P from EKF and UKF algorithm.

After 200 iterations, the landmarks' locations converge and are shown in Fig.3.3. By pure motion model result, the estimated landmarks (green) are far away from the true landmarks (red). Furthermore the EKF or UKF SLAM achieves similar performance. This can also be seen from the robot and landmark Mean Squared Error(MSE) plot in Fig.3.4.

Experiment 2

In this experiment, we compare the performance of EKF-SLAM and UKF-SLAM. The input noise is doubled to $q = 2 \times [0.1; \pi/18]$, which is about 20% of the input magnitude. After 200 iterations, the SLAM estimated map and the recorded robot path are shown in Fig.3.5. Fig.3.6 shows the MSE of robot position and landmarks. It is clear that the UKF-SLAM in general is better than EKF-SLAM with less estimation errors in both robot poses and landmarks. It is clear that for EKF-SLAM the relative large error can build

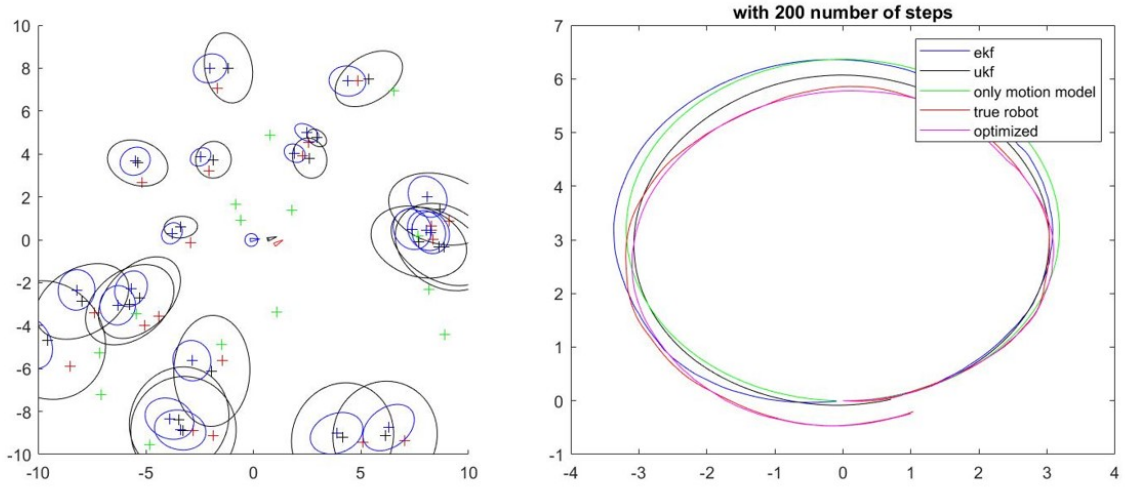

Figure 3.5: The left image shows the location of the estimated robot and landmark location. The right image shows the robot's true and estimated path.

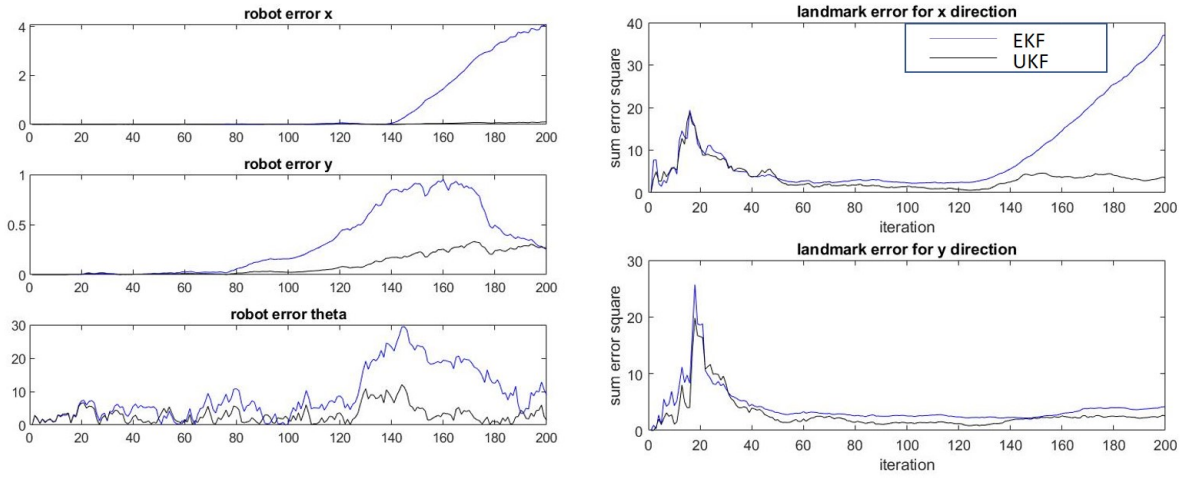

Figure 3.6: Robot and landmark error

up and cause divergence as shown in Fig.3.5 and Fig.3.6.

3.5 Particle Filter Based SLAM

In this section, the application of particle filters to SLAM problem is discussed. The particle based SLAM algorithm is one of the most popular SLAM algorithms in robotics since it is easy to implement, and most importantly it handles non-Gaussian distributions. Furthermore the algorithm has better

robustness to uncertainties and errors.

From the particle filter algorithm given in Algorithm 5, the core idea of particle based SLAM is that each particle $x_t^{[i]}$ carries a grid map and a robot state (pose and orientation). Then the measurement model is used to evaluate particles and reallocate 'bad' particles accordingly. Finally, the low variance sampler (Algorithm 7) is used for the resampling step. In addition, during the localization processes, each particle represents only the robot state.

There are many differences between the particle filter based SLAM and Kalman filter based SLAM. The former can handle non-Gaussian noises in either system or sensor measurements, and thanks to the grid map and beam measurement model, it does not require information on landmarks and landmarks correspondence. This makes the particle filter SLAM algorithm easier to implement and faster than the Kalman filter based ones, especially when the built map is getting bigger.

3.5.1 Measurement Model

In order to generate proper weights for each particle, the measurement needs to return the possibility of the current particle being correct. Most sensors generate more than one measurement value at each time. For example, when using a range-bearing laser beam sensor, each measurement contains a number of range and angle values generated. Assume there are K number of laser beam values from each measurement z_t at time t.

$$z_t = \{z_t^1, \cdots, z_t^K\} \tag{3.40}$$

Then the measurement probability can be calculated as the product of the probability for each laser value

$$p(z_t|x_t, M) = \prod_{k=1}^{K} p(z_t^k|x_t, M) \tag{3.41}$$

Beam model of range finder

No matter how good a sensor is, its measurements usually are subject to various errors. For a range sensor, measurement noise, unexpected object, ran-

dom unknown sources of noises, and detection failure are four common types of possible measurement errors. They should be considered when calculating $p(z_t|x_t, M)$

- Due to measurement noises, the observation is not 100% match of the constructed grid map, where the observed hit point may not be located at the same occupied gird cell of the object. To evaluate the possibility $p_{hit}(z_t^k|x_t, M)$ of a laser beam hit the object, a narrow Gaussian distribution with standard deviation σ_{hit} and mean $\bar{z}_t$ is assumed. $\bar{z}_t$ is the mean location of the closest object to the robot along the laser beam. To calculate $\bar{z}_t$, the ray casting is used, which is very computationally expensive (to accelerate the process, this step is typically pre-computed and saved for after use). The measurement probability can be expressed as:

$$p_{hit}(z_t^k|x_t, M) = \begin{cases} \eta\mathcal{N}(z_t^k, \bar{z}_t^k, \sigma_{hit}^2), & if \quad 0 \leq z_t^k \leq z_{max} \\ 0, & otherwise \end{cases} \tag{3.42}$$

  where z_{max} is the maximum sensor range, and variance σ_{hit}^2 need to be tuned. The normal distribution follows the equation

$$\mathcal{N}(z_t^k, \bar{z}_t^k, \sigma_{hit}^2) = \frac{1}{\sqrt{2\pi\sigma_{hit}^2}} e^{-\frac{(z_t^k - \bar{z}_t^k)^2}{2\sigma_{hit}^2}} \tag{3.43}$$

  And the normalization factor η is the inverse of the area under the distribution

$$\eta = \left(\int_0^{z_{max}} \mathcal{N}(z_t^k, \bar{z}_t^k, \sigma_{hit}^2) \right)^{-1} \tag{3.44}$$

- While running the SLAM algorithm, there can be an *unexpected object* moving around. These objects can block sensing and cause wrong measurements. In fact, the closer the object is, the smaller the likelihood of having an unexpected object, and for this reason an exponential distribution is used to tackle this situation. However, unexpected or moving objects are not considered in our model.

- A laser sensor shoots out a laser beam and calculates the range from the returned beam. When the map is too large for the sensor to re-

ceive returning beams, or a light absorbing objects may also weaken the returning beams, the sensor in this case generates a *maximum range measurement* value. In this thesis, we only consider the exceeding maximum range case and model this as a narrow uniform distribution centered at z_{max}

$$p_{max}(z_t^k|x_t, M) = \begin{cases} 1, & if \quad z_t^k \approx z_{max} \\ 0, & otherwise \end{cases} \tag{3.45}$$

- The range sensor may generate *random noise* in measurements, we model is as a uniform distribution spread over the entire measurement range

$$p_{rand}(z_t^k|x_t, M) = \begin{cases} \frac{1}{z_{max}}, & if \quad 0 < z_t^k < z_{max} \\ 0, & otherwise \end{cases} \tag{3.46}$$

Then, combine these four types of distributions with normalized weight parameters to get the measurement probability

$$w_{hit} + w_{unexpect} + w_{max} + w_{rand} = 1 \tag{3.47}$$

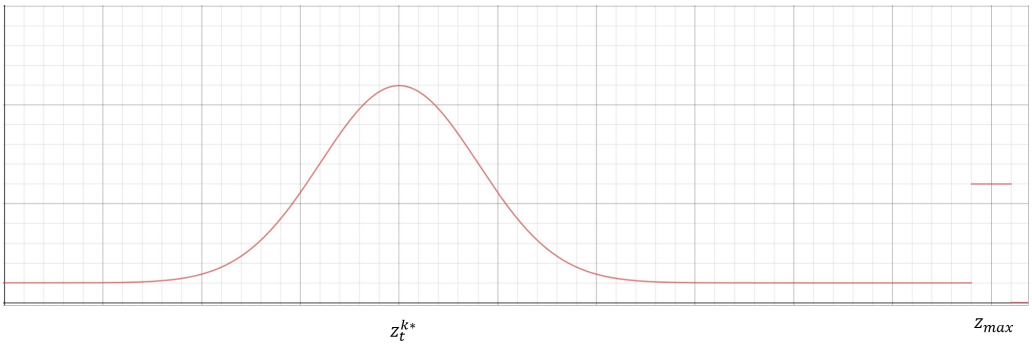

Figure 3.7: Beam model probability distribution

Then the measurement model can be implemented by the following beam range finder model algorithm [60], and its distribution shows in the Fig.3.7

Algorithm beam model (z_t, x_t, M) :
$p = 1$;
for k = *1 to* K **do**
  $p_k = w_{hit} * p_{hit}(z_t^k|x_t, M) + w_{unexpect} * p_{unexpect}(z_t^k|x_t, M) + w_{max} * p_{max}(z_t^k|x_t, M) + w_{rand} * p_{rand}(z_t^k|x_t, M)$;
  $p = p * p_k$
end
return p

Algorithm 8: Range finder model pseudo algorithm

3.6 Discussions

In this chapter, several common filter based SLAM algorithms are described and discussed. The Kalman Filter based SLAM algorithms were developed earlier, and have been studied and improved over the years. For small number of features they do not need much computational resource to achieve online SLAM. However, they have some obvious disadvantages compared with the more recent particle filter (PF) based SLAM scheme. First, KF SLAM can only apply a limited number of features/landmarks because it needs to run the KF update step for all sensor observations, which is time-consuming. For example, a standard LiDAR sensor collects more than thousands of laser measurements for each scan and the sensor takes many scans within a second. It is computational exhaustive and difficult to run in real-time. In addition, the number of sensor measurements also bring out the problem of correspondence. All of the features/landmarks information is saved in a state vector, and each state corresponds to a specific map feature/landmark. While the robot moves, it needs to identify whether the sensor value is for a new feature or a specific existing feature. This step is complicated using only raw measurement data, such as range and bearing value or images. The KF based SLAM algorithms usually involves a data pre-processing step to extract features from a cluster of data points and find corresponds to the existing states. On the other hand, the particle filter directly takes laser scans to update the occupancy grid map.

When compared to the PF-SLAM, another disadvantage is that the KF-SLAM only tackles Gaussian distributed noises. Also, it needs to apply approximation algorithms (linearization or UT) to nonlinear systems to ensure the Gaussian assumption. On the other hand, PF-SLAM can handle complex multi-model probability distributions, which provides a much better estimation of the robot pose. Because of the characteristics of Kalman filter, a large change of observation error would significantly affect the SLAM result. Lastly, the KF-SLAM cannot handle the global localization problem and recover from localization failures, for example, the kidnapped robot problem (to be discussed in Chapter 5). Conversely, the PF-SLAM algorithms can solve

these problems. For example, by applying the Adaptive and Augmented Monte Carlo Localization (AMCL) algorithm, which uniformly distributes particles to the map while processing the global localization, the localization failure can be detected. This work is discussed in Chapter 5 in details.

Finally, the cumulative error from each filter step is a problem for both of the algorithms. To handle that, the Graph-based SLAM is introduced to apply minimize the error using all the past robot position (path), which will be addressed in the next Chapter.

Chapter 4

Graph-based SLAM

The filter based SLAM recursively estimates state values by using only its value from the last step. Differently, the Graph-based SLAM solves *full SLAM problem*, which calculates a solution for past poses and all features on the map simultaneously. The graph-based SLAM problem can be separated into the front and back end process. The front-end is sensor related, which collects sensor data and computes a relatively accurate map and robot pose. The front-end process combines a number of algorithms, including feature detection, data association, data fusing, and filters, etc. Then it constructs an abstract representation of the map called a graph, which has two key components, nodes and constraints (edges). Nodes contain landmarks and robot poses information. Constraints connect nodes (robot poses and landmarks) by robot motion dynamics and encode the sensor measurement between two nodes while the robot is moving. However, the error between expected node positions and constraints keep growing because of the sensor and motion noises. Therefore, the graph SLAM back-end process is designed to periodically reconfigure nodes to maximize the graph consistency with the measurements. In other words, the back-end process optimizes the node positions to get minimized graph errors. Fig.4.1 shows the structure of the graph SLAM process.

In this Chapter, basics of graph-based SLAM are introduce at first. Then the graph-based SLAM using visual data is addressed, and a case study of the visual SLAM is included.

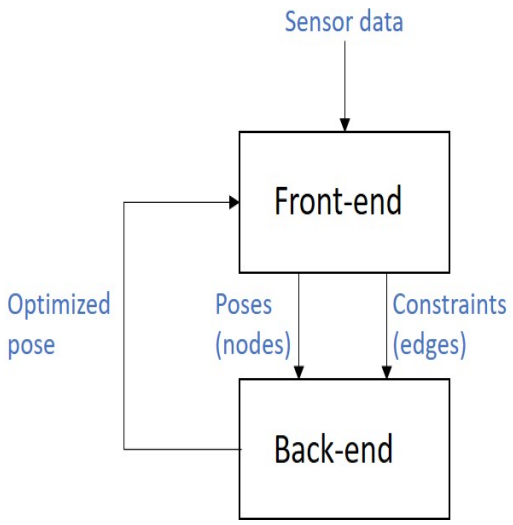

Figure 4.1: The relationship between front-end and back-end of the SLAM.

4.1 The Pose Graph

For the graph-based SLAM, the robot and landmark poses are modeled by nodes, and then edges are constructed between two nodes, which can be thought of a rope tying all nodes together. This section only addresses the pose-graph, where only robot pose is modeled for a SLAM process. Each edge consists of a probability distribution over the rigid-body relative transformations between the two poses, and a covariance associated to the transformation. For example Fig.4.2 shows the pose-graph representation of a SLAM process,[22]. Every node in the graph corresponds to a robot pose here, and nodes are linked by edges computed from motion estimation and loop-closure constraints.

For a pose-graph, a predicted node pose is calculated using the motion model and the previous node position. Because of the noise, the predicted location is not equal to the predicted one. Therefore, the back-end of graph-based SLAM attempts to minimize the total error between predicted measurement and observed measurement for all tied nodes.

Let $\mathbf{x} = [\mathbf{x}_1^\top \dots \mathbf{x}_T^\top]^\top$ be a vector of nodes from time 1 to T, where the i_{th}

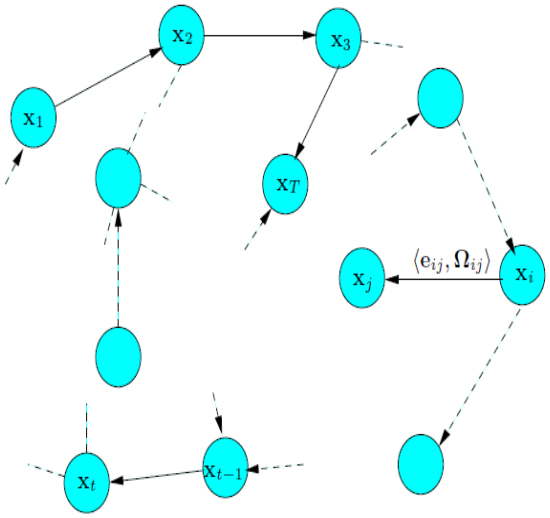

Figure 4.2: A pose-graph representation of a SLAM process, the arrow between nodes $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ is a regular constraint. The edge between $\mathbf{x}_i$ and $\mathbf{x}_j$ is a loop-closure constraint, [22]

node $\mathbf{x}_i$ contains the robot location t_i and orientation θ_i.

$$\mathbf{x}_i = [t_i, \theta_i]^\top$$

The Fig.4.3 shows the relationship between the graph components, and we use $\mathbf{x}_i$ and $\mathbf{x}_j$ as an example. Let z_{ij} and Ω_{ij} denote the observation mean and information matrix between node i and j, respectively. In this case, the measurement z_{ij} is a transformation that makes the observation from i to j to have the maximum overlap, which is computed through environment matching algorithms. Also, $\hat{z}_{ij}$ is the predicted measurement between nodes, which is normally computed through relative transformations (motion model and odometry, etc.). For example, by relative transformation that can be computed from an adjacent pose, the pose of node $\mathbf{x}_i$ and $\mathbf{x}_j$ are able to be computed sequentially. The error between two measurements can be computed as:

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = z_{ij} - \hat{z}_{ij} \tag{4.1}$$

To minimize the sum of error function for all $< i, j > \in \mathbb{C}$, where $\mathbb{C}$ contains

indices for an existent constraint (observation) z. The global cost function can be written as:

$$F(\mathbf{x}) = \sum_{<i,j>\in\mathbb{C}} \mathbf{e}_{ij}^{\top} \Omega_{ij}^{-1} \mathbf{e}_{ij} \tag{4.2}$$

and the optimal solution for the nodes is:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} F(\mathbf{x}) \tag{4.3}$$

which can be solved by using the non-linear least-squares method.

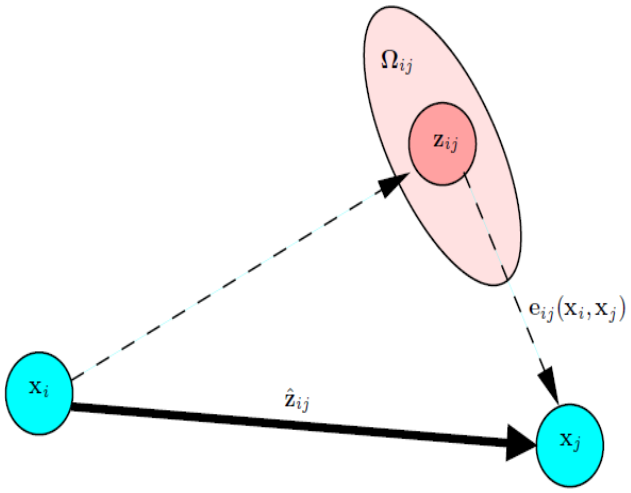

Figure 4.3: Aspects of an edge connecting node i and node j. The error $\mathbf{e}_{ij}$ depends on the displacement between the expected and real observation, [22]

To make the calculation of the relative transformation easier, we convert from Cartesian coordinates to Homogeneous coordinates at first. Let f be the function transform from Cartesian to Homogeneous. Then the Eq.(4.1) can be rewritten as:

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = f^{-1}(Z_{ij}^{-1}(\mathbf{X}_i^{-1}\mathbf{X}_j)) \tag{4.4}$$

where $Z_{ij} = f(z_{ij})$, $\mathbf{X}_i = f(\mathbf{x}_i)$, $\mathbf{X}_j = f(\mathbf{x}_j)$ and

$$f(\mathbf{x}_i) = \begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(x_i^{\theta}) & -\sin(x_i^{\theta}) & x_i^x \\ \sin(x_i^{\theta}) & \cos(x_i^{\theta}) & x_i^y \\ 0 & 0 & 1 \end{bmatrix}$$

where R is the rotation matrix, t is the translation between x and y coordinate. Then the error function for pose-pose and pose-landmark constraints are obtained:

Pose-Pose edge:

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \begin{bmatrix} R_{ij}^{\top}(R_i^{\top}(t_j - t_i)) - t_{ij} \\ \theta_i - \theta_j - \theta_{ij} \end{bmatrix} \tag{4.5}$$

Pose-Landmark edge:

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \left[R_i^{\top}(t_j - t_i) - z_{ij}\right] \tag{4.6}$$

For the pose-landmark error function, the angle is excluded from observation ($z_{ij} \in R^{2\times 1}$) and error function, where in this equation $\mathbf{x}_i$ is the robot pose node and $\mathbf{x}_j$ is the landmark node.

Graph SLAM Optimization

The non-linear least-squares optimization is used to find the optimal node poses to minimize the measurement error according to Eq.(4.3). A good initial guess for the robot's nodes (position) $\breve{x}$ is computed from the front-end of the graph-based SLAM ($\mathbf{x}$ from Fig.4.2). Therefore, the error function (4.2) can be approximated by Gauss-Newton or Levenberg-Marquardt algorithms. Taking the first order of Taylor expansion from the error equation and approximate around the initial guess $\breve{x}$, the error function becomes:

$$e_{ij}(\breve{\mathbf{x}}_i + \Delta\mathbf{x}_i, \breve{\mathbf{x}}_j + \Delta\mathbf{x}_j) = e_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x}) \approx e_{ij} + J_{ij}\Delta\mathbf{x} \tag{4.7}$$

Then refer to Section 2.4.2, we are able to compute the optimized nodes poses iteratively. During the computation, the Jacobian and Hessian matrix of the error function is sparse since it only depends on the states of two nodes. For example, the Jacobian of the error function is derived as:

$$J_{ij} = \begin{pmatrix} 0 & \cdots & 0 & A_{ij} & 0 & \cdots & 0 & B_{ij} & 0 & \cdots & 0 \end{pmatrix} \tag{4.8}$$

Where A_{ij} and B_{ij} are the derivative with respect to x_i and x_j, respectively. Because of the sparsity, only the non-zero part is cooperated during the optimization computation.

4.1.1 Loop Closure

To further improve the graph-based SLAM, the loop closure technique is introduced to generate more constraints between the current node and its surrounding nodes, when the robot revisits a place, for example, the constraint between node $\mathbf{x}_i$ and $\mathbf{x}_j$ can be generated through loop closure algorithms. The exhausted method computes the likelihood between observation for a node with all other nodes, and find the best ones to construct loop-closure constraints. For example, while using LiDAR sensors, [45] proposed a multi-resolution scan matching algorithm for loop-closure detection. To reduce the computational cost, the work in [26] used a branch-and-bound approach to accelerate the loop closure search. The details of laser scan loop closure detection methods are addressed in section 5.2. When a valid loop-closure is detected, pose transformation is estimated as a loop-closure constraint.

For the visual based SLAM approaches, ORB-SLAM [42] uses the combination of ORB feature and bag of words method to accomplish real-time loop-closure and SLAM process. The details of visual based loop-closure are introduced in later sections.

4.2 Graph-based SLAM Simulation

This experiment is aiming to simulate the optimization (back-end) for the graph-based SLAM problem. Hereby we use pure odometry and EKF, UKF SLAM algorithms to pre-calculate the good guesses for robot pose. The EKF/UKF SLAM testing benchmark is successfully developed and presented in Section 3.4.

This simulation is the extension of the filter based SLAM simulation in Section 3.4. Before applying the optimization, the pose-graph is first constructed by adding edges between two adjacent poses when the robot moves. Then imitate loop closure by adding edges between random nodes and the first robot node, since the pure EKF/UKF algorithm cannot detect loop-closing and scan matching algorithms are not developed in the simulation.

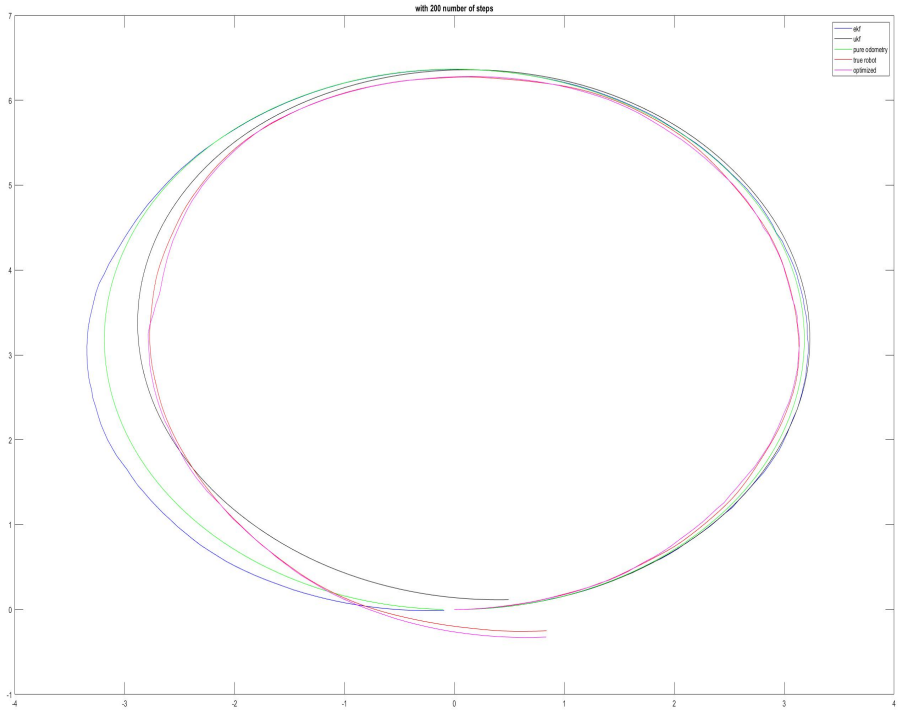

Figure 4.4: Estimated robot paths

Experiment 1

Similar to filter based SLAM experiments, the first experiment runs 200 iterations and the robot input speed is set to be $u\_v = 1 \ \ m/s$, angular velocity $u\_w = \pi/10 \ \ rad/s$. Also, there are 20 landmarks used and the noise standard deviation $q = [0.1; \pi/18]$, $v = [2, \pi/18]$. The result of the EKF-SLAM is used to build the pose-graph. In addition to the edges between adjacent nodes, ten loop closure edges are added between the first node and other ten random nodes. The loop-closure measurement is set to be the true distance between the random node and the first node. The estimated robot path is shown in Fig.4.4. The red line is the true robot path; the green line is the computed through motion model; blue is the estimated path from EKF-SLAM, and black is estimated path from UKF-SLAM. Finally, the pink is the optimized path, which is closest to the true path. For better clarity, Fig.4.5 shows the zoomed robot path and the global error during the pose-graph optimization process.

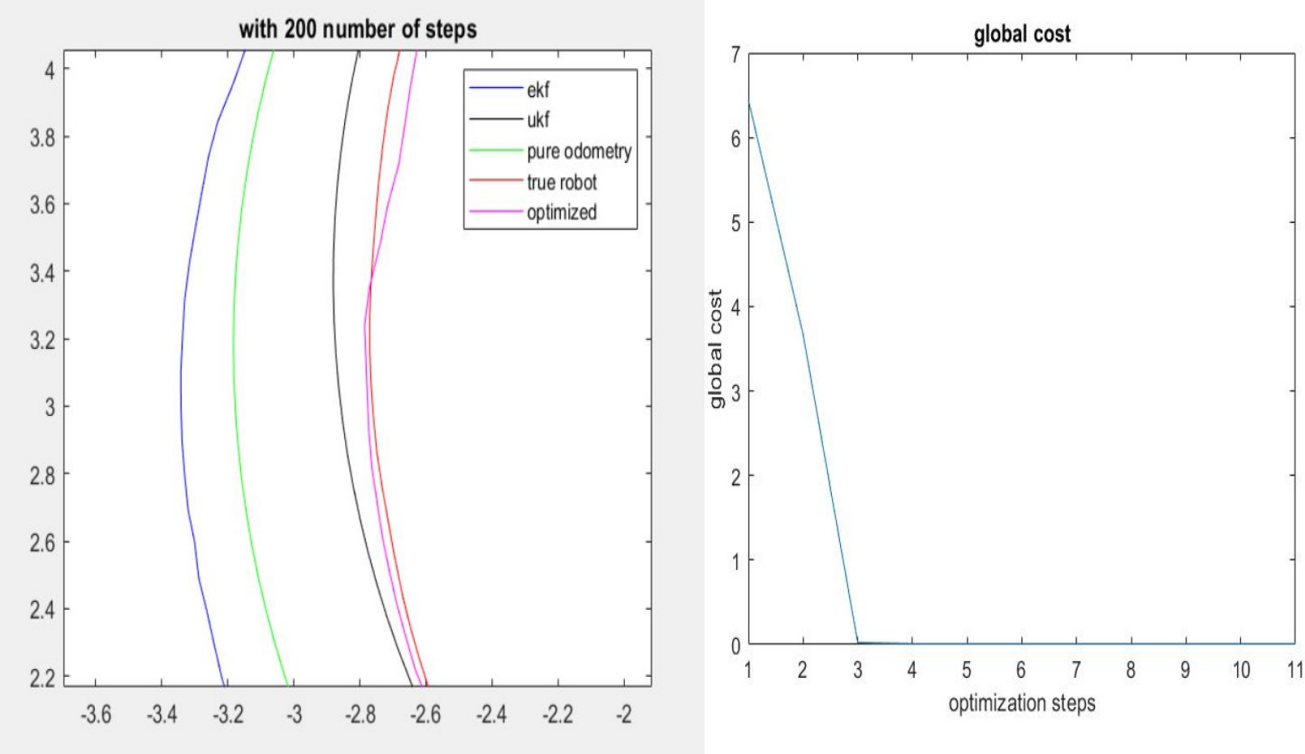

Figure 4.5: The figure on the left shows the zoom in robot path. The figure on the right shows the convergence of the error.

It is clear that the global cost converges with in 2 steps and the optimized robot trajectory is almost identical to the true path.

Experiment 2

To better visualize the effect of the optimization algorithm, larger noises are added to the system. The control input is kept the same, and the input and measurement noises are doubled. Similarly the EKF-SLAM result is used to construct the pose-graph, and the result for the estimated path shows in the Fig.4.6. The optimized path is forced closer to the true path, even when the EKF estimation is not relatively accurate.

Overall, due to the optimization process, the graph based SLAM has better performance than the filter based SLAM algorithm. In practice, the loop closure detection and optimization processes are executed for every number of SLAM steps to ensure the map and robot poses are optimized in real time.

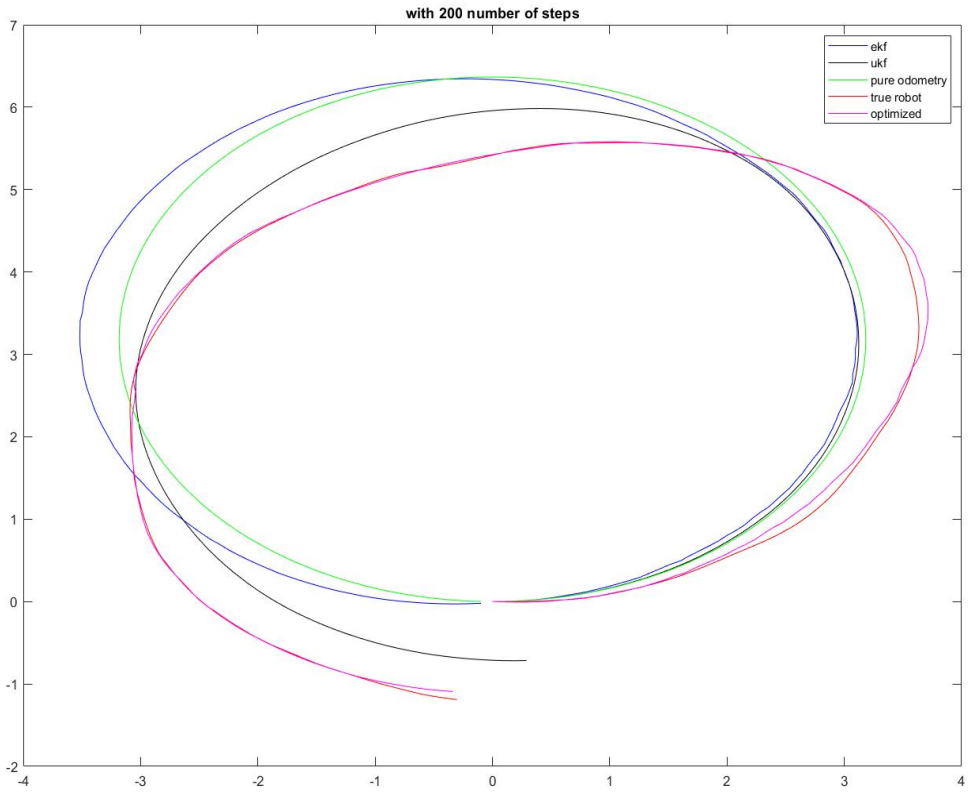

Figure 4.6: Estimated robot paths

4.3 Visual SLAM and Application

In this section, the visual based SLAM approach is introduced based on using monocular camera data. This approach is applied to a platform for agriculture equipment's docking task [1]. In the following, the visual odometry (VO), image feature extraction, motion estimation, and bag of visual word (bovw) algorithms are introduced at first.

4.3.1 Visual Odometry (VO)

Contrary to wheel odometry, VO is not affected by wheel slip or other adverse conditions. Also, it provides a relatively accurate trajectory estimation

[1]Special thanks to Prof. Hong Zhang (Computing Science, UofA) for sharing the image datasets for this case study. Permission has been granted to include images and results in this thesis

and more dimensional information compared with wheel odometry. The VO process can be divided into five ordered steps. **Image sequence → Feature detection → Feature matching → Motion estimation → Local optimization**. If additional loop-closure detection step and global optimization step are included in VO, it performs visual SLAM. Here we mainly study the monocular VO, in which both relative motion and 3-D structure must be computed from 2-D bearing data.

The VO algorithm uses a sequence of images set as input. Image features are extracted by some visual feature extraction algorithms like ORB [51], SURF [2], etc. Then extracted features are matched between images, and good matches are applied to the motion estimation algorithm. Finally, for the Visual SLAM the loop closure and optimization step added to reduce the global error accumulated overtime through local frame-to-frame motion uncertainty.

To perform the feature matching step, hereby the Brute-Force Matcher is used to match features between images, which takes the descriptor (vector of numbers) of a feature in the first image and calculates the Euclidean distance to all other features vectors in the second image. Then the closest one is chosen to be the matched feature. After computing all the matches, the RANdom Sample Consensus(RANSAC) algorithm is used to remove outliers and the motion between two images is calculated by the five-point or eight-point method, [24].

The relative scale [53] is unknown for a monocular camera. In this section, scales are extracted from the given wheel speed data. The relative scale can also be computed by the scale of the distance between two keypoints in different frames. The distance between the first two camera poses is usually set to one. When a new image is received, the relative scale and camera pose with respect to the first two frames are determined using the trifocal tensor or external sensor's knowledge, etc. This method usually has to consider the scale drifting problem, especially when more images are added to the dataset. The loop-closure optimization is used to handle the scale drifting problem.

Image Features

.5

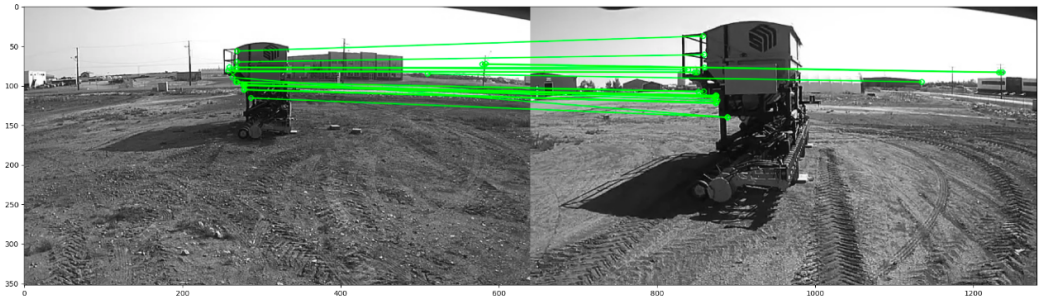

Figure 4.7: ORB feature matching

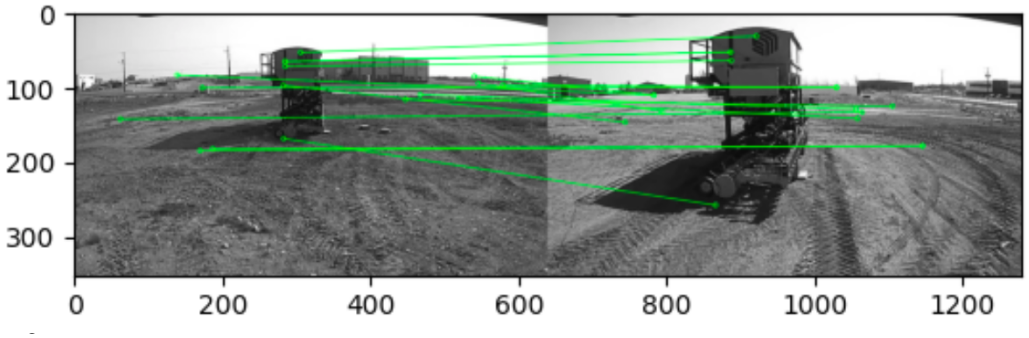

Figure 4.8: SIFT feature matching

.5

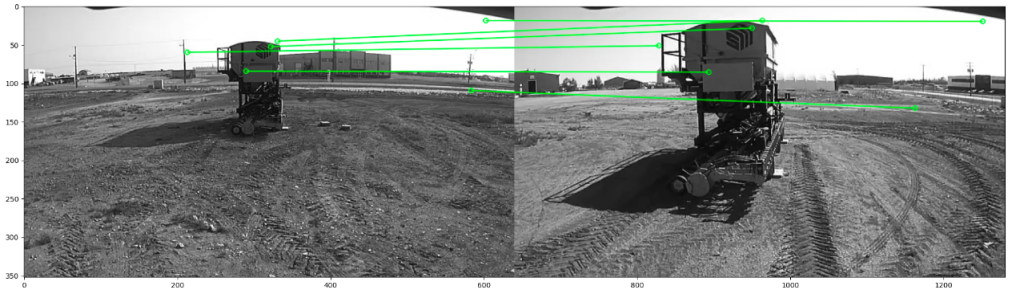

Figure 4.9: SURF feature matching

Image sequences are recorded from a docking process for agriculture equipment. Unlike the 2D/3D LiDAR scans, it is challenging to incorporate full image information (every pixel) to the SLAM algorithm. Using features is much faster than using all pixels and also more robust to noise and light.

First of all, we need to find the best feature extraction algorithm for the dataset. There are three feature extraction algorithms tested, including ORB,

SURF, and SIFT feature algorithms. The Fig.4.7-4.9 show comparison of the feature extraction performance on a same pair of images. The green lines are image feature matches. It is clear that the ORB technique demonstrates a better performance in capturing features on the equipment. Also, by the characteristics of ORBs [51], it is more robust to image rotation noise and faster than SURF and SIFT. Therefore the ORB features are used for later developments.

Relative Transformation Estimation

Epipolar geometry is used to describe the relations between 3D points and their 2D projections at two camera poses. These relations can be characterized by epipolar geometry if the camera can be approximated to a pinhole camera model. For example, [58] provides an example of two cameras looking at the same point **X**, see Fig.4.10.

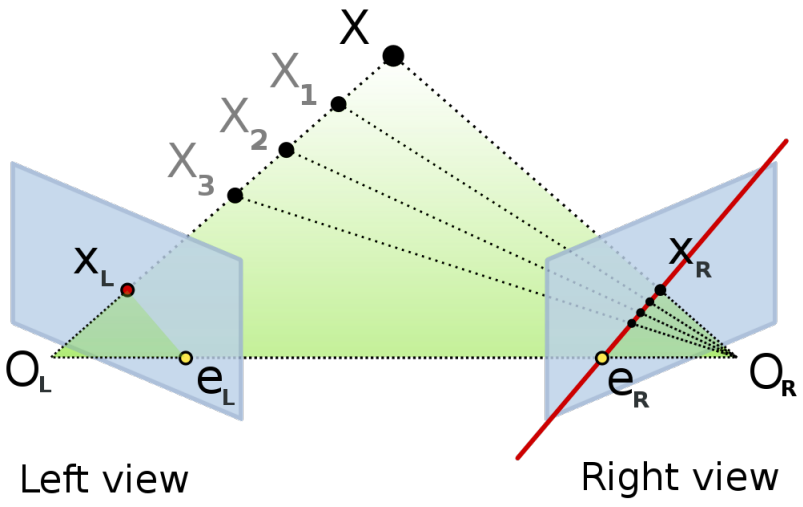

Figure 4.10: A 3D point X can be seen from two cameras at O_L and O_R in the world coordinate. The X_L and X_R are the projections of the **X** on two cameras image planes. Let e_L and e_R be epipoles, and the line pass through $\{X_L, e_L\}$ and $\{X_R, e_R\}$ are epipolar lines l_L and l_R, the points $X_1, X_2, X_L, X_R, \cdots$ and camera lenses positions O_L, O_R lies on a plane called the epipolar plane, [58].

Assume that the world reference system is associated with the left camera origin O_L, and the right camera offsets the left by a rotation R and a translation

T. In this case, projection points on the left camera plane remain $M_L = K_L \begin{bmatrix} I & 0 \end{bmatrix}$ and points on the right plane need to use the transformation to change coordination $M_R = K_R \begin{bmatrix} R & t \end{bmatrix}$, where K_L and K_R are camera intrinsic matrices. M_R and M_L represent the projection matrix with dimension (3×4) and map 3D points into the camera frame.

Consider the simplest case, let the camera parameters be 1 and the camera intrinsic matrix

$$K_L = K_R = \begin{bmatrix} I & 0 \end{bmatrix}$$

The rotation matrix R is orthogonal and the coordination transformation is

$$\begin{aligned} M_L &= \begin{bmatrix} I & 0 \end{bmatrix} \\ M_R &= \begin{bmatrix} R & t \end{bmatrix} \end{aligned} \tag{4.9}$$

Then the location of X_R on the left camera coordination system is $X_R^{'}$, and

$$\begin{aligned} X_R &= RX_R^{'} + t \\ X_R^{'} &= R^{-1}(X_R - t) \\ &= R^{\top} X_R - R^{\top} t \end{aligned}$$

Furthermore, the left camera coordinate of X_R at $R^{\top} X_R - R^{\top} t$ and O_R at $R^{\top} t$ lies in the epipolar plane, and apply the cross production to compute the plane normal vector.

$$R^{\top} t \times (R^{\top} X_R - R^{\top} t) = R^{\top} t \times R^{\top} X_R = R^{\top}(t \times X_R) \tag{4.10}$$

Since the point X_L is also on the epipolar plane, which is perpendicular to the $R^{\top}(t \times X_R)$, then their dot product is equal to zero,

$$\begin{aligned} (R^{\top}(t \times X_R))^{\top} X_L &= 0 \\ (t \times X_R)^{\top} R X_L &= 0 \end{aligned} \tag{4.11}$$

Recall that the cross product term can be rewritten into a dot production term by the following format, let a and b represent any two vectors:

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = [\mathbf{a}_x]\mathbf{b}$$

Apply this expression to Eq.(4.11) to replace the cross product into matrix multiplication:

$$\begin{aligned} (t \times X_R)^\top R X_L &= 0 \\ ([t]_x X_R)^\top R X_L &= 0 \\ X_R^\top [t]_x^\top R X_L &= 0 \\ X_R^\top [t]_x R X_L &= 0 \end{aligned} \tag{4.12}$$

Let the matrix

$$E = [t]_x R \tag{4.13}$$

where it is called the essential matrix [37], and the Eq.(4.14) is called coplanarity constraint.

$$X_R^\top E X_L = 0 \tag{4.14}$$

The essential matrix is a 3×3 matrix and has five or six degrees of freedom, depending on whether or not it is seen as a projective element (with well determined scaling). The rotation matrix R and the translation vector T each has three degrees of freedom, in total six. However, if the essential matrix is considered as a projective element, one degree of freedom related to scalar multiplication must be subtracted, leaving five degrees of freedom in total. The essential matrix has rank 2.

$$\det(E) = 0 \tag{4.15}$$

A real non-zero 3×3 matrix is an essential matrix if and only if satisfies the following important cubic constraints [25]:

$$E E^\top E - \frac{1}{2} trace(E E^\top) E = 0 \tag{4.16}$$

When the intrinsic matrix K_L and K_R are not identity in the Eq.(4.9), for all pairs of corresponding points they have following constraints

$$\begin{aligned} X_R^\top F X_L &= 0 \\ X_R^\top K_R^{-\top} [t]_x R K_L^{-1} X_L &= 0 \end{aligned} \tag{4.17}$$

Where the matrix $F = K_R^{-\top} [t]_x R K_L^{-1}$ is known as the fundamental matrix. The essential matrix and fundamental matrix have similar properties. For

example, they can compute the epipolar lines by using X_L and X_R. The epipolar line on one image frame can be calculated by the product of essential/fundamental matrix with the projected point on another image frame, for instance, $l_R = EX_L$ or $l_R = FX_L$. The matrix production of essential/fundamental matrix with the epipoles are equal to zero, $Ee_L = Ee_R = 0$ and $Fe_L = Fe_R = 0$. These properties are beneficial when the essential/fundamental matrix is computed and used to estimate the corresponding point's location from another image.

Eight-point Method

The knowledge of the essential matrix and fundamental matrix allows us to estimate the relative transformation between two camera poses. To find these matrices by using a few corresponding points is challenging. The fundamental matrix has eight degrees of freedom. In [24] an improvement is proposed to solve the fundamental matrix by using eight corresponding points. Hereby we provide a brief introduction to the eight-point method. For given corresponding points X_L and X_R and its fundamental matrix F:

$$X_L = \begin{bmatrix} x_l \\ y_l \\ 1 \end{bmatrix}, X_R = \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix}, F = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \tag{4.18}$$

Inserting these equations to the constraint Eq.(4.17), and rewritten as:

$$x_r x_l F_{11} + x_r y_l F_{12} + x_r F_{13} + y_r x_l F_{21} + y_r y_l F_{22} + y_r F_{23} + x_l F_{31} + y_l F_{32} + F_{33} = 0 \tag{4.19}$$

this can be simplified to a matrix multiplication $af = 0$, where:

$$\begin{aligned} a &= \begin{bmatrix} x_r x_l & x_r y_l & x_r & y_r x_l & y_r y_l & y_r & x_l & y_l & 1 \end{bmatrix} \\ f &= \begin{bmatrix} F_{11} & F_{12} & F_{13} & F_{21} & F_{22} & F_{23} & F_{31} & F_{32} & F_{33} \end{bmatrix}^\top \end{aligned} \tag{4.20}$$

Given a set of N matched point, a set of linear equation can be obtained as the form:

$$\begin{aligned} Af &= 0 \\ A &= \begin{bmatrix} a^1 \\ \vdots \\ a^N \end{bmatrix} \end{aligned} \tag{4.21}$$

The fundamental matrix is determined only up to scale, to give a constraint $||f|| = f^\top f = 1$. To avoid the solution of $f = 0$, rank$\{A\} \leq 8$, since F has 8 degrees of freedom. Because of noises in the measurement, the matrix A can be of full column rank (i.e. 9). To solve this, we can apply a least-squares optimization to the system to minimize $||Af||$. The Singular Value Decomposition (SVD) can be used to solve the least-squares problem and find the fundamental matrix F.

Five-point Method

With the knowledge of the camera intrinsic matrix, it is sufficient to use five feature point pairs to compute the essential matrix. Although the implementation is not straightforward since it involves various non-linear equations, the five-point algorithms are more commonly used. Hereby we briefly introduce how this method works. Similar to the eight-point algorithm, a linear system for essential can be written as:

$$Qe = 0 \tag{4.22}$$

where

$$\begin{aligned} q &= \begin{bmatrix} x_r x_l & x_r y_l & x_r & y_r x_l & y_r y_l & y_r & x_l & y_l & 1 \end{bmatrix} \\ e &= \begin{bmatrix} E_{11} & E_{12} & E_{13} & E_{21} & E_{22} & E_{23} & E_{31} & E_{32} & E_{33} \end{bmatrix}^\top \end{aligned} \tag{4.23}$$

Similar to the matrix A from Eq.(4.21), the matrix Q is built from five matched points and it is a 5×9 matrix. For using a given Q, the solution to the essential matrix E can be written as a combination of its null-space:

$$E = xE_0 + yE_1 + zE_2 + wE_3 \tag{4.24}$$

where $E_i, i = 0, 1, 2, 3$ are null-space bases calculated from Q by either SVD or QR factorization. For scalars x, y, z, w are defined only up to a common scale factor, so let $w = 1$. Given the null-space bases, the essential matrix E is hence determined by some $(x,\ y,\ z)$. To compute these parameters, we substitutes the Eq.(4.24) into Eq.(4.16) and performing Gaussi-Jordan elimination with partial pivoting to obtain a coefficient matrix corresponding to the 20-dimensional vector:

$$\begin{bmatrix} x^3 & x^2y & x^2z & xy^2 & xyz & xz^2 & y^3 & y^2z & yz^2 & z^3 & x^2 & xy \\ & & & & & & xz & y^2 & yz & z^2 & x & y & z & 1 \end{bmatrix} \tag{4.25}$$

To compute the essential matrix, in [48] it is proposed to use the 13^{th} order polynomial, but in [44] an improved solution is proposed, in which a 10^{th} order polynomial is used to compute essential matrix.

The RANSAC algorithm is usually incorporated with the motion estimation process, which makes the estimation more efficient and robust to outliers. For example, for a five-point algorithm, the RANSAC searches for a good five feature point pairs, in which the estimated transformation/rotation is valid to most of the other feature pairs.

Rotation and Transformation Recovering

The transformation T and rotation R between two camera poses can be calculated [25] by using SVD,

$$E = \mathbf{U}\Sigma\mathbf{V}^\top \tag{4.26}$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal 3×3 matrices and ,the Σ contains singular values of the essential matrix E, which has the following form because the essential/fundamental matrix has a rank 2.

$$\Sigma = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Then we define an orthogonal matrix $\mathbf{D}$ as:

$$\mathbf{D} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.27}$$

and based on the Eq.(4.13), the essential matrix can be reformulated as:

$$E = [T]_x R = \mathbf{U}\Sigma\mathbf{V}^\top = \mathbf{U}\mathbf{D}\Sigma\mathbf{U}^\top\mathbf{U}\mathbf{D}^{-1}\mathbf{V}^\top \tag{4.28}$$

The the transformation and rotation can be computed by $T = \mathbf{U}\mathbf{D}\Sigma\mathbf{U}^\top$ and $R = \mathbf{U}\mathbf{D}^{-1}\mathbf{V}^\top$.

Random Sample Consensus (RANSAC)

The RANSAC algorithm [15] is proposed to manage a dataset with a large proportion of outliers. The traditional sampling techniques use as much data as possible to get rid of outliers and obtain an initial solution. The RANSAC uses the smallest set possible then proceeds to enlarge this set with consistent data points, and it has the following steps:

1. Randomly select a minimum number of required points to estimate the parameters of the model
2. Determine the number of points from the dataset fit with the computed model with a preset tolerance value
3. If the fraction of the number of the inliner points over the dataset exceeds the predefined threshold, re-estimate the parameters using all the identified inliers and then terminate.
4. Otherwise, repeat the step 1 to 3.

For camera motion estimation one can use the RANSAC algorithm to iteratively search for the best group of five/eight feature points to calculate the most robust transformation between camera poses. For example, the five-point algorithm is iteratively applied to five randomly selected samples to generate its hypotheses. The first group of feature points satisfies the RANSAC requirement is selected as the solution.

Visual Bag of Words (vBow)

The visual bag of words [56] [18] is a simplified representation of images, and it converts image features to a dictionary of words. The vBow is commonly used for image classification and matching by comparing vBow between different images. We use the vBow during the loop closure process, to quickly search for well matched images.

To train the vBow vocabularies, image features from all images in a dataset are extracted using feature extractor algorithms, for example, ORB, SIFT, etc.

The feature descriptors are multi-dimensional vectors, and we make clusters from the descriptors using clustering algorithms like K-Means, Mean-shift, etc. The center of each cluster will be used as the visual dictionary's vocabularies, and up to this point, the vBow training is completed.

Next, to use the trained vocabularies, descriptors of the image can be represented by vocabularies and then construct a histogram of the number of vocabularies' in the image. To find the best matched image, the similarity cost of vocabulary histogram can be computed through Euclidean distance, and cosine similarity, etc. Finally, matches with costs smaller than a threshold value are considered as good loop-closures and can be added to the pose-graph as loop closure constraints.

4.3.2 Visual SLAM Experiments

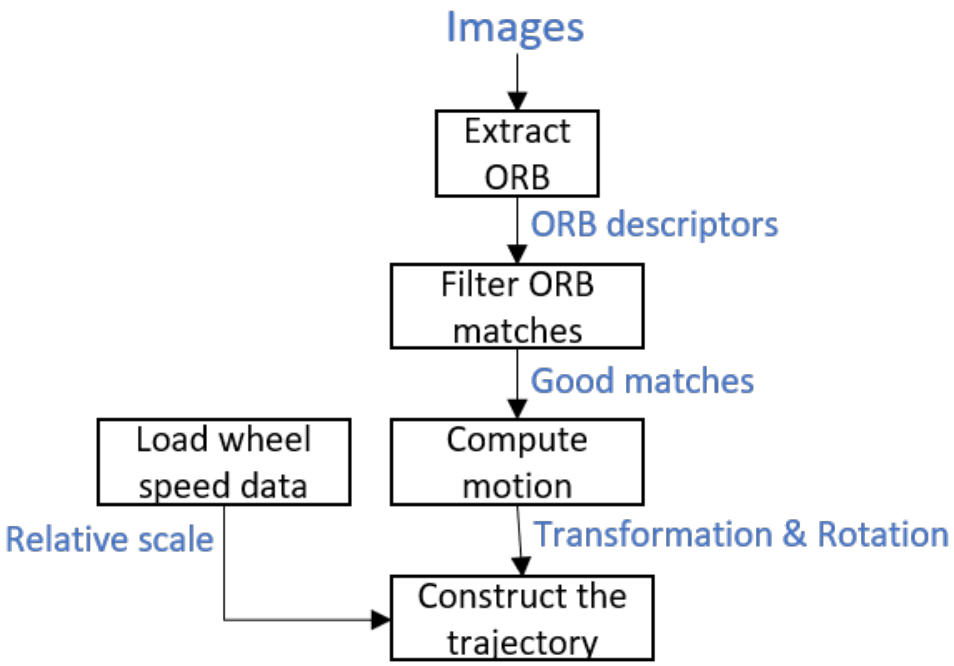

Figure 4.11: Flowchart of the ORB trajectory builder (VO)

The visual SLAM algorithm is tested on a self-implemented program, which can be found at `https://github.com/linjianxiang/mono_camera_docking`. Several algorithm libraries are developed, including ORB feature

extraction and matching, visual odometry estimation, vBow, loop-closure detection, pose graph and optimization. Because of the time limitation, the program only generates the trajectory of the camera pose, without the estimated map (for example a feature map). The map point matching and bundle adjustment are missing, therefore the relative scale is not implemented in this experiment. In fact it needs to compute the distance between map points in different image frames to determine the scales, hence a map is required for image frames to find the same map points. In addition to that, the loop-closure can be detected but it also needs scales to build edges. The flowchart of the current program process is shown in Fig.4.11

Figure 4.12: Example of the equipment docking process. For this dataset the equipment is driven from position where image 1 is taken to where the image 4 is taken.

During the experiment, our program takes given image series, and the equipment wheel speeds, where images are taken by a monocular camera installed on the agriculture equipment. The camera collects images while the equipment is manually moved to the docking station and then slowly docked at a specific position without any collision, see Fig.4.12.

The equipment's trajectory is generated from docking image sets, and the relative scale is extracted from the equipment speed data. It should be noted that there is not a ground truth to evaluate the performance of the program. But the agriculture equipment motion can be observed from image sets, show-

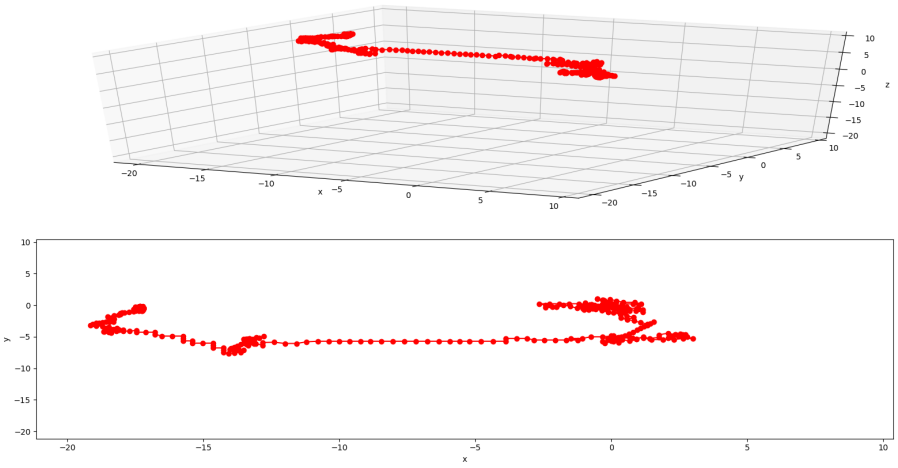

Figure 4.13: 3D and 2D trajectory generated from the docking image series

Figure 4.14: The image shows good ORB matches (filtered by RANSAC). ORBs on the equipment are filtered out for both top and bottom matches, and only ORBs on a white house from the background are remaining to compute transformations.

ing that the equipment moves from initial location to the docking area without much direction adjustment, so we intent to see a similar motion from the constructed trajectory. For example, one of the built 3D/2D trajectory is shown in Fig.4.13, where the red dots from -20 to 0 on 2D x-axis indicate that the

equipment is moving towards the docking station. Moreover, for the last part of dots, the equipment is closer to the docking station so it slows down with more motion adjustments. The motion estimation seems difficult during this part, mainly because the matching algorithm captures most background ORB features, see Fig.4.14. Since good matched ORBs are sometimes from the background of the image, which is far away from the camera, and their size in images varies only slightly while equipment is moving. Other than that, there are situations when the equipment is stationary, and an object moves into the scene. This also generates the wrong motion for the camera poses.

4.4 Discussion

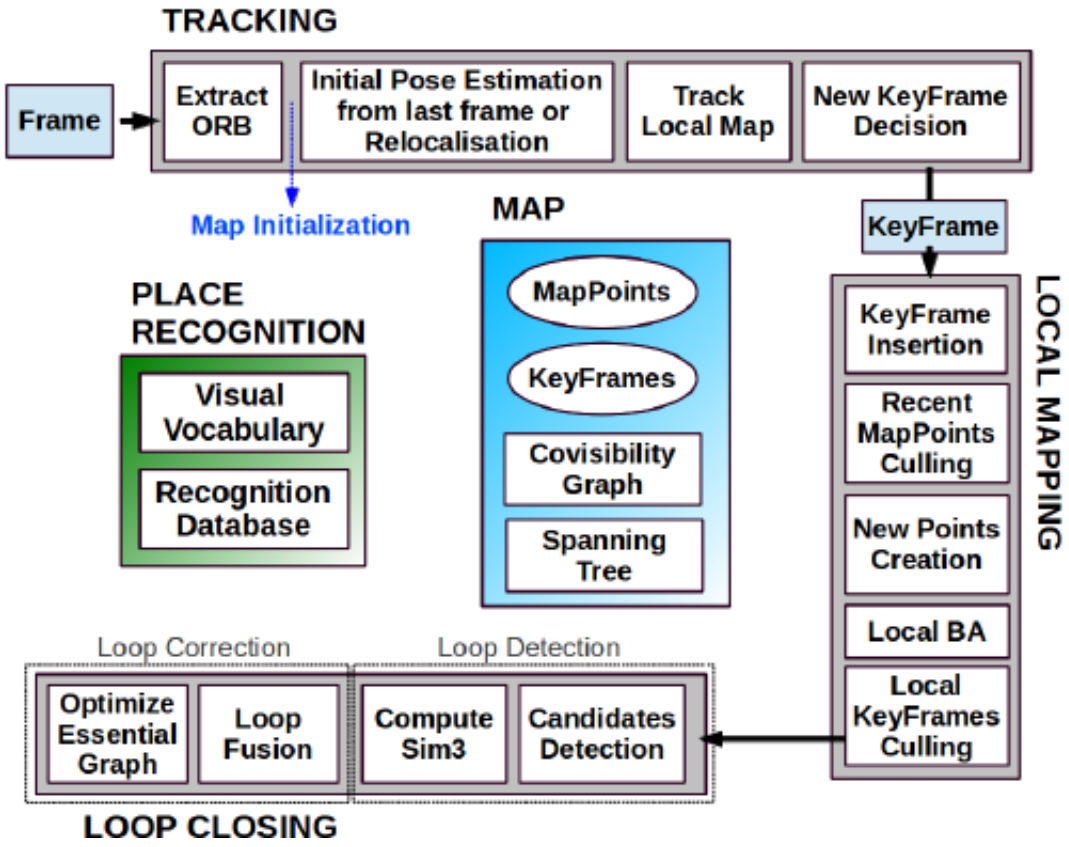

Figure 4.15: The ORB-SLAM system flowchart,[42]

From the above experiment, it is seen that wrong trajectory estimation can occur because of the background and unexpected moving object. Increasing the number of ORBs, and RANSAC's tolerance threshold could help reduce

the effect of background and moving objects. Also, constructing a feature map and applying bundle adjustment to it can also help to mitigate the problem. Another solution tested is to apply an object detection algorithm to only capture the docking station part in the image[2]. The result of detection is shown in `https://www.youtube.com/watch?v=k-ZmleC9v4o`. Then ORBs located within the bounding boxes are applied for matching. But it is possible that only a small amount of ORBs are located in the bounding box, especially when the camera is far away. This can be solved by applying pre-docking, a different method used when the machine is far away from the docking station. Overall, building an optimized map is a game changer and should be consider in the future work of this research.

The visual SLAM processes' fundamental libraries and their interfaces are developed and tested, including ORB feature extraction, ORB feature matching, keyframe detection, Visual odometry estimation, vBow, vBow loop-closure detection, and pose graph optimization. For future development, image points can be constructed as a map and use the map to compute the relative scale and loop-closure constraints. Depth estimation algorithms for monocular images could also be helpful, for example, [21] uses unsupervised learning to estimate depth. In addition, point culling algorithms are needed to remove redundant points and bundle adjustment (BD) algorithm to adjust the camera poses.

The flowchart originally given in ORB-SLAM [42] Fig.4.15 provides a systematic overview of image feature based SLAM algorithm. In our development, we closely follow this flowchart, and most of the algorithms from the Tracking and Loop-closing process are implemented in this project, but the Local Mapping module will need to be developed to complete a full SLAM problem. The keyframe detection library can be applied to reconstruct image frames, ensuring scene content changes between each keyframe. Furthermore, this process enables running the SLAM process for a lifelong time with excellent robustness and generates a compact and trackable map. In addition to that, the loop-closure detection is achieved by matching the current image vBow with the

[2]special thanks to Siqi Yan for helping with the implementation of yolo-v3 [50] for object detection

vBow database, and then compute constraints between these frames. Finally, the optimization algorithm is applied to reconstruct the optimized pose-graph.

Chapter 5

Kidnapped Robotic Problem

The Kidnapped Robot Problem (KRP) refers to the case when the system loses knowledge of the true position of a robot, which is teleported to an arbitrary location, [60] [64]. To recover from KPR, a proper approach should first detect the kidnapping event and then apply a global localization algorithm to retrieve the robot's true pose.

Normally there exist two kinds of KPR, real kidnapping and localization failures. The real kidnapping means that the robot is taken to another position due to a significant drifting from the original path; the localization failure occurs when the system has the wrong belief of a robot pose. For example, unmodelled objects can cause a localization algorithm's failure.

The global localization is useful when a robot does not know its pose, for which the Monte-Carlo method is proposed, namely the Monte-Carlo Localization (MCL) method, [10] [60] [59]. In such an approach, particles are uniformly distributed on the map. The best-matched particle is determined as the robot's hypothetical pose based on control data and sensor data. In addition, vision based global localization algorithms have been developed. In [54], a vision sensor is used and the particle filter is applied to match specific image features. In [52], clusters of particles are used to localize the robot globally. Furthermore, several algorithms are proposed to detect the kidnapping event. In [7], the detection of kidnapping events is achieved by comparing the maximum current weight and weight changes of particles against certain threshold values. In [9], increment of entropy is used to detect kidnapping

event.

One advantage of the Monte-Carlo Localization (MCL) is due to that it can handle non-Gaussian distributions. Nevertheless, the MCL sometimes cannot effectively recover the KRP, because of characteristics of the particle filter used in MCL. In many cases, KRP may not be recovered if the map is complex and when there exist many areas of the map that are not covered by particles. One useful MCL approach is the so-called Augmented Monte-Carlo Localization (AMCL) [60], in which extra particles are randomly generated and added to the map when the average weight drops drastically so that the true pose region is likely to be discovered by new particles.

In Section 5.1 a new improvement is proposed to assist in solving the kidnapped robot problem (KRP) in the Monte Carlo localization approach. It is known that the Augmented Monte Carlo Localization (AMCL) method can detect and solve kidnapping problems by tracking the sudden drop of particles' average weight and then performing a global localization to estimate the robot's new pose. The proposed work improves the AMCL by adopting the idea of the global/static costmap. The costmap aided AMCL algorithm is able to recognize those absolutely wrong particles and then randomize them to enhance the speed of recovery from the localization failure. Simulations are carried out on different maps to validate and demonstrate the performance improvement using the proposed method.

In Section 5.2, a graph-based method is given to detect the KRP event using an average filter on scan matching. Then, in order to recover from KRP, the proposed method generates a new map which is merged with the existing one upon the loop-closure detection.

5.1 Application of Costmap to Monte Carlo Localization

To improve the KRP recovery performance further from original AMCL, an algorithm is given based on the costmap [39], and it is applicable to all MCL based localization approaches. The proposed approach can recycle invalid

particles before calculating weights. Specifically, the global costmap recognizes invalid regions on the map, and particles in those regions are regenerated. When a complex map is used and/or not enough particles are generated in the localization, the costmap based method can achieve considerable performance improvement. Another advantage of the costmap is that it can be readily incorporated into any particle filter based algorithms. For example, integration of the costmap in Augmented MCL is presented, and the result is shown in the simulation section.

5.1.1 Partical Filter & Monte Carlo Location

The Bayes filter (Section 2.1.3) is the foundation of filter based SLAM algorithms. Particularly, the MCL algorithm adopted in this research is built upon the particle filter introduced in Section 2.3. Denote $\overline{bel}(x_t)$ as the prior probability distribution before incorporating observation, and $bel(x_t)$ as the posterior probability distribution over robot states conditioned on the available data. In MCL, the posterior $bel(x_t)$ and prior $\overline{bel}(x_t)$ are represented by the set $\mathcal{X}_t$ and $\overline{\mathcal{X}_t}$ with M number of weighted particles at time t.

$$bel(x_t) = \{x_t^{[i]}, w_t^{[i]}\}_{i=1,\dots,M} \tag{5.1}$$

where $x_t^{[i]}$ represents the state of the i^{th} particle at time t, and $w_t^{[i]}$ is the importance factor or weight of the i_{th} particle at time t.

The standard MCL algorithm follows exactly the same structure as the particle filter in Algorithm 5. Each of the particle contains information of its current pose and a 2D-scan. This is different from particle filter mapping application in which each particle has information of the whole map. From the algorithm, the first **for** loop generates the weighted prior distributed particles. Based on the motion model, each particle utilizes control inputs and previous particle states (line 4) to compute new states at time t; then, the sensor mode is used to calculate the weight w_t of each particle (line 5). The sensor model uses measurements and each particle's environment to calculate the probability of a particle located at the right place. The second **for** loop is called particle resampling. Base on the importance weights, the resampling acts similarly

as the updating process, which renders a particle distribution close to the posterior $bel(x_t)$.

5.1.2 The KRP and the Augmented Monte-Carlo Localization

The kidnapped robot problem (KRP)

Due to computational complexity, usually, the number of particles used in the MCL algorithm cannot be chosen as arbitrarily high. Certain areas of the map may not have enough particles covered. When the KRP happens, the true robot pose may be the absence of particle coverage. A global localization method must respond faster and more frequently when a true pose is out of particle distribution. For demonstration, a simple map called Maze is adopted to simulate robotic kidnapping, shown in Fig.5.1. The black line indicates the true obstacles, while the pink blocks are the generated costmap (by considering the robot's physical size). The idea of costmap will be explained in the subsequent section. Fig.5.1 clearly shows the kidnapping event when the robot's true location is out of the posterior distribution of all particles (marked by red arrows). As another example, a complex map called Willow is used in the simulation, for which the part of the map is shown in the Fig.5.2.

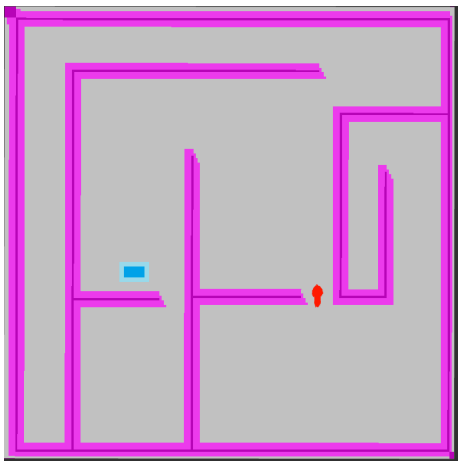

Figure 5.1: The blue block shows where the robot is located on the map; red arrows are particles. Because the KRP, the true location is out of posterior distribution. The figure also shows the generated costmap (pink block).

Figure 5.2: Part of the willow map

The AMCL algorithm

In the following, the Augmented MCL method and how it solves robotic kidnapping or localization failure problem is described. The Augmented MCL suggests to use exponential smoothing with weights w_{fast} and w_{slow} to detect drastic decay of the averaged weights w_{avg}. The more the average weight decreases, the more likely the robot kidnapping event has occurred. When the sensor measurement no longer matches the environment data, a much smaller importance weight is calculated by the sensor model for each of the particle. Often particles' weights drop because the robot drives itself into a different (new) environment. For example, when a robot is driving through a door, particles behind the true pose can have a large weight drop. For the exponential smoother, larger weight w_{fast} responds faster to change of w_{ave} than the smaller weight w_{slow}.

```
1  Algorithm Augmented MCL(X_{t-1}, u_t, z_t)
2  static w_slow, w_fast ;
3  X̄_t = X_t = ∅ ;
4  for i = 1 to M do
5  |  sample x_t^[i] = motion_model(u_t, x_{t-1}^[i]) ;
6  |  w_t^[i] = measurement_model(z_t, x_t^[i], map) ;
7  |  X̄_t = X̄_t + < x_t^[i], w_t^[i] > ;
8  |  w_avg = w_avg + (1/M) w_t^[i]
9  end
10 w_slow = w_slow + α_slow(w_avg − w_slow) ;
11 w_fast = w_fast + α_fast(w_avg − w_fast) ;
12 for i =1 to M do
13 |  if with probability max{1 − w_fast/w_slow, 0} then
14 |  |  add random pose to X_t
15 |  else
16 |  |  draw i with probability ∝ w_t^[i] ;
17 |  |  add x_t^[i] to X_t
18 |  end
19 end
20 return X_t
```

Algorithm 9: Augmented Monte Carlo localization pseudo algorithm

The Augmented MCL is given in Algorithm 9 [60]: it shows that when a significant drop of w_{ave} is detected, the AMCL algorithm generates new particles randomly to globally search for the robot location. The number of randomized particles is proportional to the weight drop (line 10, line 11), and the ratio $1 - w_{fast}/w_{slow}$ represents the percentage of randomized particles. If it is required to have a fixed size of the particle set, the Augmented MCL will randomize part of particles arbitrarily when the average weight drops.

Challenges of the AMCL algorithm

In the AMCL algorithm, there exist performance trade-offs dependent on the choice of α_{fast} and α_{slow}. The algorithm can perform randomization more frequently if α_{fast} is much larger than α_{slow} (according to the exponential smother). However, at the same time, particles around the true pose may be chosen to be randomized and redistributed to another location. On the other hand, if the α_{fast} is chosen to be close to α_{slow}, then it requires a much larger weight drop to trigger the randomization. This is not likely to be effective

in solving KRP because there are not sufficient particles being randomized. Hence proper parameter tuning is required in AMCL for acceptable performance.

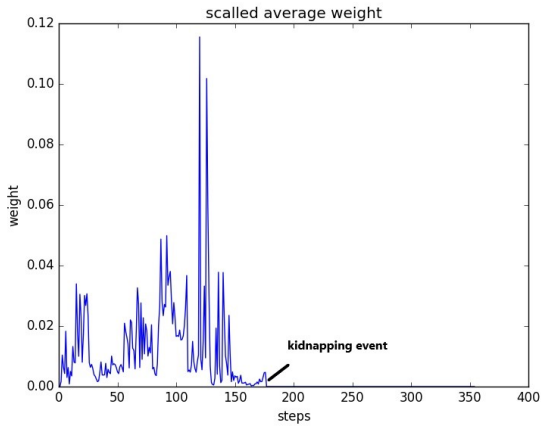

Figure 5.3: The plot shows the weight change before and after a kidnapping event. The weight is always low after the kidnapping event. The weight change before the robot is kidnapped is due to the change of environment.

Furthermore, based on the simulation result of the AMCL, one can observe that in the step right after the robot is kidnapped, a sufficient amount of particles are regenerated. This is because the average weight decreases extremely fast at this step, meaning that the ratio $1 - w_{fast}/w_{slow}$ is large and the probability to randomize particles is high. However, if the true pose is not found at this step (i.e. particles are not near the true pose), in this case both of w_{fast} and w_{slow} are low, which possibly result in a low ratio $1 - w_{fast}/w_{slow}$ or equivalently a low probability of randomization as shown in Fig.5.3, then in the subsequent steps there will be insufficient amount of particles generated, possibly leading to the localization failure.

5.1.3 The Proposed Monte-Carlo Localization based on Costmap

The Costmap Monte-Carlo Localization (CMCL)

Inspired by the idea of static costmap, this section proposes two improved algorithms to solve the robotic kidnapping problem. Originally, the costmap is

used in robotic path planning. Based on the world map and sensor data, different cost values are assigned to different map grids, hence the name costmap. Usually, the inflated costs are calculated based on the specified robot radius, and the inflation propagates the cost values from obstacles/occupied cells. Furthermore, the cost decreases with distance. For example, within the distance of the robot radius, if cells in the grid map are closer to the occupied cells (e.g. obstacles), they are assigned larger cost values. During the path planning, the algorithm can choose the path with the lowest cost to determine the robot's optimized motion.

In this section, a binary costmap is adopted for simplicity (but without loss of generality), in which the inflated cells have the same value (e.g. 1). In the proposed algorithm, to implement the static binary costmap, at first the given map is convert to a binary map (with 1's or 0's). This map serves as a mask to identify the occupied and unoccupied grids (regions). Then the algorithm proceeds to calculate the averaged values for cells around unoccupied grids (those with value 0). If the value is larger than a given threshold, then the algorithm changes the grid value to 1. The number of layers of cells to be inflated is smaller than or equal to the robot's radius divided by the grid length.

The costmap algorithm utilizes the map and the knowledge on the robot size to generate blocks around obstacles as inflated-cost region. Specifically those blocks represent the region that the robot is likely to collide with the obstacle, which is referred to as invalid zones. For example, in Fig.5.2 and 5.1, the black lines represent the original map, and the pink region represents the map inflated by the given robot radius, and the blue block represents the footprint of the robot. To avoid collision, the footprint of the robot should never intersect with the black lines and the center point of the robot should never cross the pink blocks. In the localization step, when the robot's location is lost, the proposed Costmap-MCL randomizes the absolutely wrong particles (e.g. those in and crossing the invalid zones based on costmap), instead of regenerating random particles solely from the prior distribution. When a randomized particle is generated around the true robot pose, it is likely for

the SLAM algorithm to re-identify the robot position.

Particles' poses before and after using the motion model (for updating) can be calculated. However, whether a particle has crossed the invalid zone cannot be observed directly during this updating process. To solve this problem, in the algorithm, the path of a particle is divided into a number of short intervals. For convenience, the particle's short path interval is assumed as a straight line between the poses before and after applying the motion model. The end poses of these shorter paths are consecutively calculated and checked to determine whether they are located in the invalid zone (i.e., collision). The higher the updating frequency of the particles, the fewer paths are needed.

```
1  Algorithm Costmap-MCL ( X_{t-1}, u_t, z_t)
2  X̄_t = X_t = ∅ ;
3  for i = 1 to M do
4  |   sample x_t^[i] = motion_model(u_t, x_{t-1}^[i]) ;
5  |   if Costmap^[i] is invalid then
6  |   |   randomize pose for x_t^[i]
7  |   end
8  |   w_t^[i] = measurement_model(z_t, x_t^[i], map) ;
9  |   X̄_t = X̄_t + < x_t^[i], w_t^[i] > ;
10 end
11 for i =1 to M do
12 |   draw i with probability ∝ w_t^[i] ;
13 |   add x_t^[i] to X_t
14 end
15 return X_t
```

Algorithm 10: Proposed Costmap-MCL pseudo algorithm

In Algorithm 10, the pose of a particle represents its own belief of the true location of the robot. When there is localization failure and the robot keeps moving, wrong particles will eventually across the invalid region or out of the map. Using the costmap, invalid particles can be firstly identified. The proposed algorithm can then regenerate these particles before the resampling step in any of the Monte-Carlo localization approaches.

Augmented Costmap Monte-Carlo localization (ACMCL)

In this section, the costmap is incorporated in the Augmented MCL for localization failure recovery, which is called the Augmented Costmap MCL (ACMCL) and shown in Algorithm 11. One key difference of this algorithm from the existing AMCL is that the costmap condition is checked before updating particles' importance weights. If a particle is deemed invalid, it will be randomized (line 8), and the invalid particle set is represented by $\mathcal{X}_c$. A invalid particle is recognized if it locates or has crossed invalid costmap region. For the rest of the valid particles, the same procedure/treatment as in AMCL is applied. In addition, the randomized particles are not included in the calculation of the average weights, in order to prevent them from varying too much after randomization, which may cause failure of the exponential smoothing filter.

Algorithm ACMCL $(\mathcal{X}_{t-1}, u_t, z_t)$
static w_{slow}, w_{fast} ;
$\mathcal{X}_c = \emptyset$;
$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$;
for *i = 1 to M* **do**
  sample $x_t^{[i]} = \text{motion\_model}(u_t, x_{t-1}^{[i]})$;
  if $Costmap^{[i]}$ *is invalid* **then**
    randomized pose for $x_t^{[i]}$;
    $\mathcal{X}_c = \mathcal{X}_c + < x_t^{[i]} >$;
  else
    $w_t^{[i]} = \text{measurement\_model}(z_t, x_t^{[i]}, map)$;
    $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + < x_t^{[i]}, w_t^{[i]} >$;
    $w_{avg} = w_{avg} + \frac{1}{M} w_t^{[i]}$;
    $w_{slow} = w_{slow} + \alpha_{slow}(w_{avg} - w_{slow})$;
    $w_{fast} = w_{fast} + \alpha_{fast}(w_{avg} - w_{fast})$;
  end
end
for *i =1 to M* **do**
  if *with probability* $max\{1 - w_{fast}/w_{slow}, 0\}$ **then**
    add random pose to $\mathcal{X}_t$
  else
    $\bar{\mathcal{X}}_t = \text{resample\_model}(\ \mathcal{X}_t, w_t)$;
  end
end
return $< \mathcal{X}_t, \mathcal{X}_c >$

Algorithm 11: Augmented Costmap Monte Carlo localization pseudo algorithm

5.1.4 Simulation Result

In this section, simulations are performed by using the two maps shown in Fig. 5.1 & 5.2, and comparison results are presented. Usually, localization algorithms are evaluated by two different performances, namely the position tracking and localization failure recovery. For evaluation purposes, the distance between the hypothetical position and the true robot position is used to measure the tracking and recovery performance. Simulations are run on the Robotic Operating System (ROS), where the algorithm modification is implemented and applied to the existing AMCL library. The two maps (one simple and small, and one complex and large) are used. In addition, in the simu-

lation settings, the noisy odometry model is used for the robot motion, and the maximum likelihood laser beam sensor model is used as the measurement model.

Fig.5.4 depicts the convergence of the AMCL and the proposed ACMCL on recovery from KRP, using the maze map. One thousand particles are used in both algorithms. It can be seen that the error between the detected pose and the true pose converges to zero for both algorithms, but it is clear that the proposed ACMCL takes much fewer steps to converge hence the improved recovery performance. Furthermore, for clarity, the comparison of

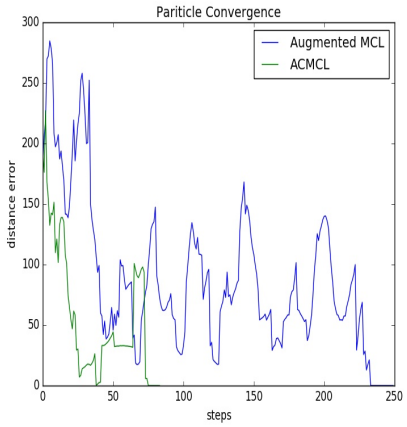

Figure 5.4: The plot shows the convergence of Augmented MCL and proposed ACMCL method for KRP recovery

average convergence steps to recover from KRP using the maze map is also shown in Table 5.1. In this case, five different numbers of particles are used in five simulation cases. It can be seen that when there are fewer particles provided, the Augmented MCL performs poorly, while as particle number increases, its convergence steps decreases. However, in all simulation cases, the proposed ACMCL demonstrates a superior convergence performance as it converges much faster than the Augmented MCL.

To further validate the performance, the more complex willow map is also used. Usually for bigger maps, a sufficiently large number of particles are needed to guarantee global localization. In this case, the particle recovery ratio instead of recovery (or convergence) steps is used to assess the performance.

Converge steps after KRP					
Particle Number	500	1000	1500	2000	2500
Augmented MC	379	108	104	72	67
ACMCL	50	50	24	28	18

Table 5.1: The table lists the steps needed for converge in the event of robotic kidnapping using the maze map

It is known that the localization recovery of MCL-type algorithms depends on how frequent the particles get randomized. The best case scenario is to randomize as many particles as possible when localization fails. Therefore, Table 5.2 shows the recovery ratio of Augmented MCL and ACMCL in two different maps. For both maps, the recovery ratio for AMCL is only 0.8%, meaning that in every step approximately 0.8% of particles are randomized in order to relocate the true pose. On the other hand, the propose ACMCL algorithm performs 5 times better than AMCL in the simple maze map, and more than 10 times better in the more complex willow map.

Particle recovery ratio		
Map	Maze	Willow
Augmented MCL ratio	0.8%	0.81%
ACMCL ratio	4.7%	8.5%

Table 5.2: The table shows the particle recover ratio for the Maze and the Willow map

Since both algorithms adopt the same particle filter settings as well as the same measurement and motion models, they should have similar robotic position tracking performance. The proposed method modification is useful to particle filter based localization algorithms for the improved recovery performance in the kidnapped robot problem. After incorporating the static Costmap to identify and regenerate invalid particles, the proposed algorithms gain much faster convergence speed and better recovery ratio compared to the existing AMCL algorithm, especially when the map is complex and/or fewer particles are used in the particle filter. However, the proposed method still

takes many steps to detect and recover KRP and it cannot perform mapping and solving KRP at the time.

In the next section, a more efficient graph based approach is introduced to solve both KRP and mapping, which is faster but computationally more expensive with more computer memory storage required.

5.2 Loop Closure Approach

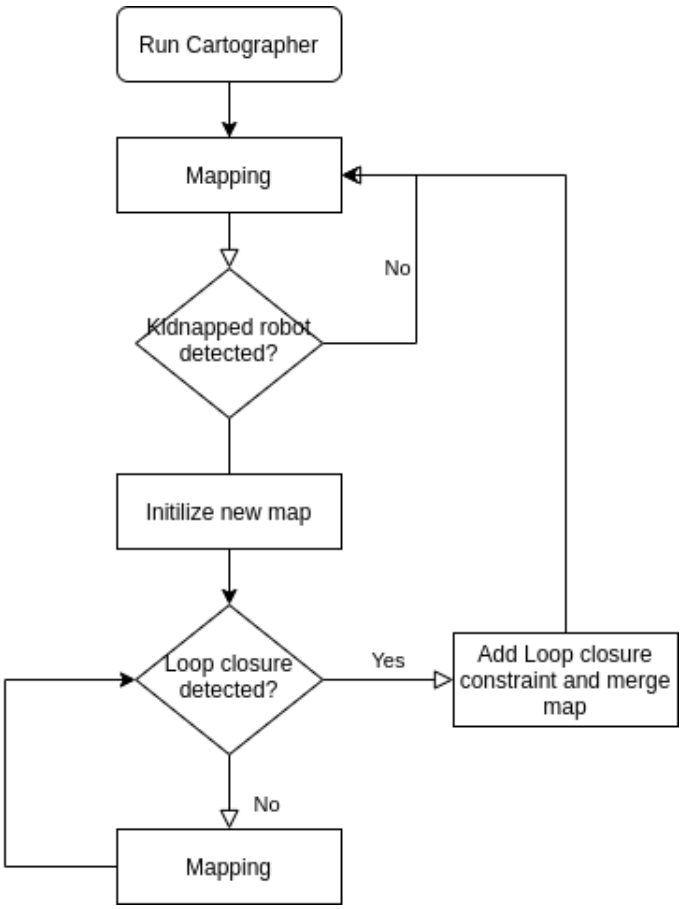

Figure 5.5: Flow chart of the KRP recovery based on loop-closure

The kidnapped robot problem (KRP) can also be thought of as a multi-session SLAM problem [35], which deals with the robot being moved to another location without knowing it. The multi-session SLAM also includes the situation that the robot is moved to a location where it cannot get the relative position to the given or pre-built map. This is called the initial state problem

(ISP), and can happen during SLAM mapping process. To solve the ISP, a new map is initialized with default reference. When the robot runs into a previously visited location, the transformation between two maps needs to be calculated and then merged into one map.

The previous section introduced the particle filter based approach to handle the KRP. It requires a map to re-localize the robot when the KRP happens and takes many iterations to converge if the map is large. In this section, our interest is to use the knowledge of loop-closure to handle the multi-session SLAM and KRP.

The global loop closure algorithms are used to detect the visited locations, where the detection algorithm uses scan matching approaches to achieve. For example, Besl and Mac Kay [4] proposed iterative closest point (ICP) for point-to-point matching. The major problem of the global loop closure lies in the fact that when the map gets bigger, the matching becomes more computationally expensive. There are many research works focused on handling the computational problem to speed up the matching process. For visual based SLAM, image feature extraction approaches are proposed, for example, SURF [2], SIFT [38], and OBR [51] are popularly used to extract image features, which are introduced in the previous section. Furthermore,in [41] features from the laser scans are extracted for matching. To further reduce the time cost, the fast Digital Bag of Binary Word (DBoW2) [18] is used to obtain descriptors along with ORB feature detectors [42]. Moreover, [27] developed histogram based matching to loop closure detection and [26] used branch-and-bound approach to accelerate the loop closure search.

To find the transformation between two maps, [5] calculates the maximum overlap between occupancy grid maps, but it is computationally exhausted. [6] proposed to use a new set of constraints to merge pose-graph between two maps. The global loop closure algorithm also calculates the transformation from the current location to a map. This transformation can be used to merge maps when the robot encounters a previously-visited location. Integrating these solutions into a graph-based SLAM [47] to minimize locally accumulated errors will further increase the accuracy of the merged map.

Google's Cartographer provides a viable implementation for robotic applications, [33] [26]. It is reviewed in details in the following. Then how to use it for a multi-session robot problem is discussed. Fig.5.5 shows the flow chart of the proposed method to handle the KRP as the multi-session SLAM.

5.2.1 Google Cartographer Overview

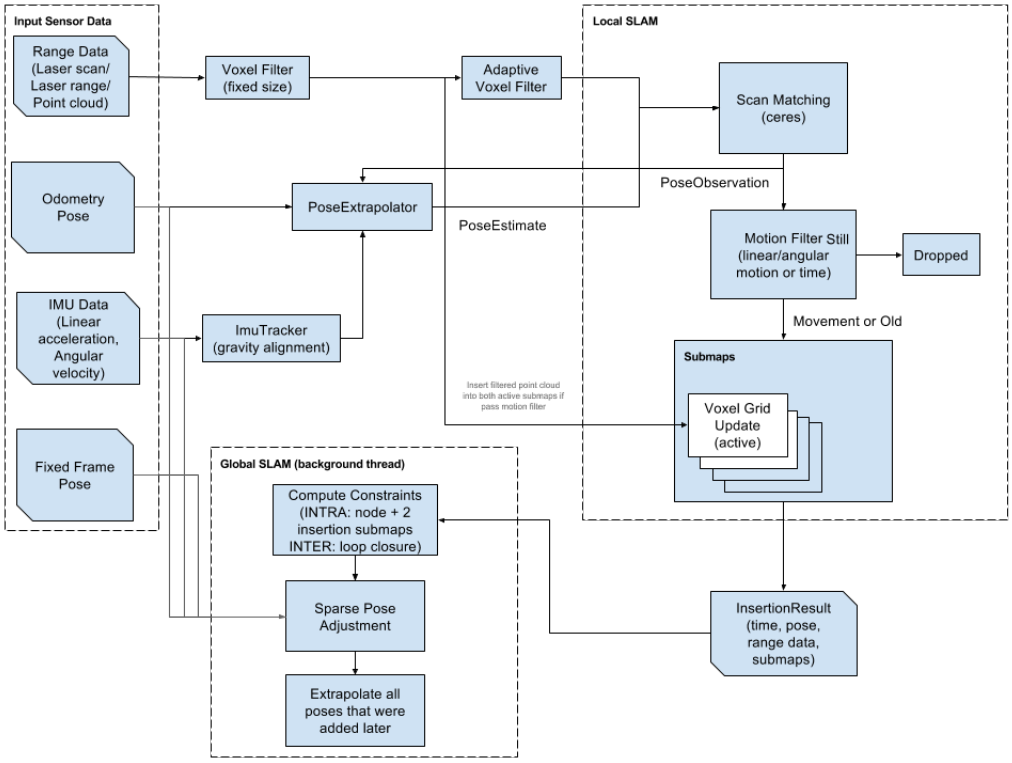

Figure 5.6: Schematics of Cartographer, [33]

The Cartographer is a SLAM package which can apply to 2D and 3D laser scans across multiple platforms and sensor configurations. The program and its insides can be found on `https://google-cartographer.readthedocs.io/en/latest/evaluation.html`. It has the overall SLAM structure shown in Fig.5.6. The Cartographer uses the idea from [26] for 2D-LIDAR data, which uses the Branch and Bound scan-matcher (BBS) to achieve the real-time loop closure.

The SLAM process firstly inserts laser scans into a *submap* at the best-estimated position, where the estimation is calculated by *local scan matching*

algorithm. Because the local scan matching applies only on recent scans, the error of pose estimation accumulates in the world frame. To handle the local errors, pose graph optimization is regularly running. Also, all completed submaps and scans are used for loop closure detection, and for this the BBS is applied to accelerate the searching. If a sufficiently good match is found, a loop closing constraint is added to the optimization problem. The optimization runs every few seconds to ensure loops are immediately closed when revisiting a location. By using the branch-and-bound approach and pre-computed grids per completed submap enables the real-time computation of loop closure detection, which is faster than using added new scans.

Local scan matching

The local scan matching calculates the optimal pose and transformation between the current LiDAR scan and the submap. The submap M is a small chunk of the world, and it is constructed by continuously align new LiDAR scans and submap coordinate frames. Let the origin of a scan at $0 \in \mathbf{R}^2$ and scan points be $H = h_{k=1,\dots,K}, h_k \in \mathbf{R}^2$. Also let the submap pose transformation $\xi = (\xi_x, \xi_y, \xi_\theta)$ and T_ξ represent the transformation between scan frame to submap frame. To transform a scan point pose p from the scan frame into the submap frame, the following is used,

$$T_\xi p = \underbrace{\begin{pmatrix} \cos\xi_\theta & -\sin\xi_\theta \\ \sin\xi_\theta & \cos\xi_\theta \end{pmatrix}}_{R_\xi} + \underbrace{\begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}}_{t_\xi} \tag{5.2}$$

The submaps take the form of probability grids and the grid value is set between $[p_{min}, p_{max}]$, and each grid has the resolution of r. Whenever a scan is inserted to the probability grid map, a set of *hit* and a disjoint set *missed* are computed. For every hit, the grids located closest to the hit point is added to the hit set. For every miss, the grids between the hit point and original point are inserted into the missed set. Each grid in either hit set or missed set is assigned with probability p_{hit} and p_{miss}. When a new laser beam is observed, the probability value updates by the following,

$$odds(p) = \frac{p}{1-p} \tag{5.3}$$

$$M_{new}(x) = \text{clamp}(\text{odds}^{-1}(\text{odds}(M_{old}(x)) \cdot \text{odds}(p_{hit}))) \tag{5.4}$$

where the function clamp(x) does:

$$if\ (x > p_{max}), \quad x = p_{max}$$
$$if(x < p_{min}), \quad x = p_{min}$$

To minimize the local error, the algorithm needs to find the optimized relative pose from the scan pose to the current local submap, which is equivalent to finding a scan pose that maximizes the probabilities of hits in the submap. The maximum probability can be represented by a nonlinear least-squares problem

$$\arg\min_{\xi} \sum_{k=1}^{K} (1 - M_{smooth}(T_{\xi} h_k))^2 \tag{5.5}$$

The function M_{smooth} applies bicubic interpolation to get a smooth version of the scan point probability values in the local submap.

Global loop closure

As scans are inserted into submaps, local error slowly accumulates. To minimize the accumulated error, [26] proposed to use the relative poses between the scans and submaps to construct a pose graph and apply the Sparse Pose Adjustment [32]. When a good match is found from local scan matching, the global scan matching runs to find the loop closing constraints for the loop-closure optimization.

The loop closure optimization can also be expressed as a nonlinear least-squares problem, which can extend easily to take into additional constraints. Let the submap poses be $\Xi^m = \{\xi_i^m\}_{i=1,\ldots,m}$ and scan poses $\Xi^s = \{\xi_j^s\}_{j=1,\ldots,n}$. The pose graph constraints take the form of relative poses ξ_{ij} calculated through global scan matching, and the associated covariance matrix Σ_{ij} can be evaluated, [46] or directly obtained through the covariance estimation of an optimization packages, e.g. the Ceres [1] optimization library in Cartographer. The residual E for a constraint can be computed in the following equations:

$$E^2(\xi_i^m, \xi_j^s; \Sigma ij, \xi_{ij}) = e(\xi_i^m, \xi_j^s; \xi_{ij})^T \Sigma ij^{-1} e(\xi_i^m, \xi_j^s; \xi_{ij}) \tag{5.6}$$

[1] http://ceres-solver.org/

$$e(\xi_i^m, \xi_j^s; \xi_{ij}) = \xi_{ij} - \begin{pmatrix} R_{\xi_i^m}^{-1}(t_{\xi_i^m} - t_{\xi_j^s}) \\ \xi_{i;\theta}^m - \xi_{j;\theta}^s \end{pmatrix} \tag{5.7}$$

The nonlinear least-squares optimization is applied to minimize the error:

$$\arg\min_{\Xi^m, \Xi^s} \frac{1}{2} \sum_{ij} \rho\left(E^2(\xi_i^m, \xi_j^s; \Sigma ij, \xi_{ij})\right) \tag{5.8}$$

This paper [26] uses Huber loss function ρ to reduce the influence of outliers, which normally appears when an incorrect constraint is added to the optimization problem. Generally outliers come from wrong loop closure constraints, bad sensor measurements, or sometimes unexpected moving objects, etc. The Huber loss function is given as:

$$\rho_\delta(a^2) = \begin{cases} \frac{1}{2}a^2, & \text{for } |a| \leqslant \delta \\ \delta|a| - \frac{1}{2}\delta^2, & \text{otherwise} \end{cases} \tag{5.9}$$

Where a^2 denotes the squared error E^2.

Branch-and-bound scan matching

The Branch-and-bound scan matching is applied to quickly find the best global loop closure match. To find the best performance between the scan to the map, the equation below is applied to compute an optimized pose ξ^*, with a searching window $\mathbf{W}$. Where the optimized pose can have the most matched scan points h_k with map grids. The $M_{nearest}$ function maps the scan points to the nearest grid.

$$\xi^* = \arg\max_{\xi \in \mathbf{W}} \sum_{k=1}^{K} M_{nearest}(T_\xi h_k) \tag{5.10}$$

To ensure the searching performance, the search step size should not be greater than pixel width r. Applying the law of cosines to the maximum range d_{max} scan point,

$$d_{max} = \max_{k=1,\ldots,K} ||h_k|| \tag{5.11}$$

The angular should not be larger than δ_θ,

$$\delta_\theta = \arccos(1 - \frac{r^2}{2d_{max}^2}) \tag{5.12}$$

The pixel width r is the horizontal and vertical step size and δ_θ as the angular search step. After choosing a step sizes, a naive algorithm will search for the

best matching score and pose ξ^* by exhaustively computing the score for every possible point in a searching window.

DFS Branch-and-bound scan matcher

```
1  Algorithm DFS branch and bound scan matcher (X_{t-1}, u_t, z_t) ;
2  best_score = score_threshold ;
3  Initialize the stack C = C_0 and sort by score of each element in C_0,
     the maximum score at the top
4  for C is not empty do
5  |  Take out c from bottom of C ;
6  |  if score(c) > best_score then
7  |  |  if c is a leaf node then
8  |  |  |  match = ξ_c ;
9  |  |  |  best_score = score(c) ;
10 |  |  else
11 |  |  |  Branch out c into higher leaf nodes C_c;
12 |  |  |  Push C_c onto the stack C and sort by score of each element
   |  |  |    in C_0, the maximum score at the top ;
13 |  |  end
14 |  end
15 end
16 return best_score and match ;
```

Algorithm 12: Pseudo algorithm for the loop closure searching for the best matched scan

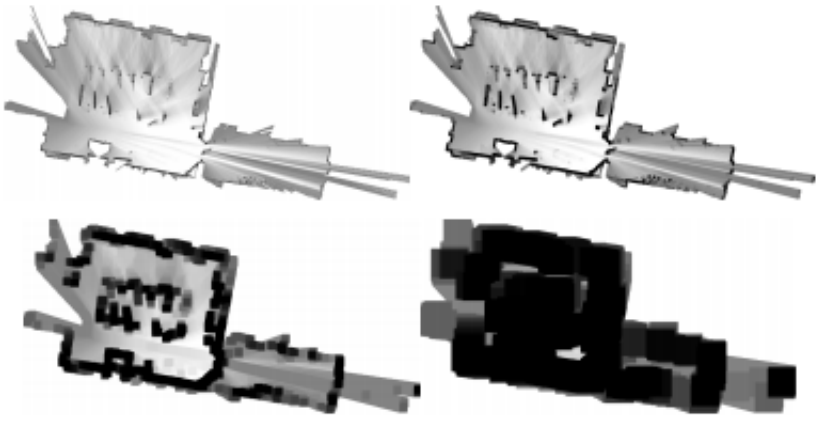

Figure 5.7: Pre-computed grids of size 1,4,16,64, [26]

To reduce the time of searching, the paper [26] proposed to use branch and bound structure for searching. The main idea is to partition the total set of feasible solutions into smaller subsets of solutions called nodes. These smaller

sets of solutions can also be branched out to inner nodes until the best solution is found. The depth-first search (DFS) algorithm is used for searching through branches to quickly evaluate the matching performance of leaf nodes. Each node in the searching tree is described by its position and the node at heigh h has combined up to $2^h \times 2^h$ number of possible solutions (translations and specific rotation).

To increase the searching speed, grids are pre-computed for every height h. Also, the pre-computed grids $M^h_{precomp}$ only stores the maximum value of the $2^h \times 2^h$ box of pixels. Let pixels have length r and the equation for pre-computed grid and node score is following:

$$M^h_{precomp}(x, y) = \max_{\substack{x' \in [x, x+r(2^h-1)] \\ y' \in [y, y+r(2^h-1)]}} M_{nearest}(x', y') \tag{5.13}$$

$$score(c) = \sum_{k=1}^{K} M^h_{precomp}(T_{\xi_c} h_k) \tag{5.14}$$

The BBS algorithm is shown in algorithm 12 based on [26] and the Fig.5.7 shows an example of pre-computed grids for different nodes height.

Kidnapped robot event detection and recovery

The KRP detection and recovery program is developed based on the Google Cartographer package. At first, the Cere optimization package provides the cost of local scan matching, then an average filter is applied to smoother the local cost for every scan. After that, the kidnapped robot event is readily detected as the dramatic cost increase. The pseudo code for the kidnapped robot event detection is the following:

1 Algorithm KRP_detection $(\mathcal{C}, c_{new}, y_{last})$
2 $Trigger = False$;
3 Initialize T; f_{size};
4 $\mathcal{C} = \mathcal{C} + < (c_{new}) >$;
5 **if** *size of* $\mathcal{C} \geqslant f_{size}$ **then**
6 | $filtered\_value =$ average_filter($\mathcal{C}$) ;
7 **end**
8 **if** $filtered\_value > T \times y_{last}$ **then**
9 | $Trigger = True$;
10 **end**
11 $\mathcal{C}$.pop(first item) ;
12 return $Trigger$

Algorithm 13: Average filter for Kidnapped robot event detection

$\mathcal{C}$ is the list of local scan matching cost, c_{new} is the local scan cost for the current scan, and y_{last} is the last average filtered value. Moreover, two parameters are available for tuning, which are the kidnap trigger threshold T and the average filter size f_{size}.

The recovery process is also built upon the existing Cartographer libraries. When the KRP happens during the SLAM mapping process, the recovery process is activated by following the flow chart in Fig.5.5. When the program detects the kidnapped event, the original mapping process stops, and then a new map is initialized and constructed by sensor inputs. After that, the method applies the BBS loop-closure algorithm [26] until the loop closure is detected. Then apply the scan matching algorithm to merge the current and the original map to complete the recovery step. Specifically, the modified program generates a new map and a new pose graph (the reference is not set). When the loop-closure is detected between a node from the new map and another node from the original map, the constraints are created between them. Then, two pose graphs are connected and the optimization is applied to relocate the second map and its submap positions.

5.2.2 Experiment

The KRP is tested on ROS integrated Cartographer library. The process of KRP detection and recovery are developed upon the existing Cartographer module [33], and the proposed program is made available to download, see link

https://github.com/linjianxiang/cartographer-mapping-kidnap. A ROS bag is used for testing, it contains range data collected from a low-cost Revo Laser Distance Sensor on Neato robotics vacuum cleaners. For KRP recovery testing, the bag was cropped to simulate KRP, for which the middle part of continuous sensor data is removed.

KRP detection experiment

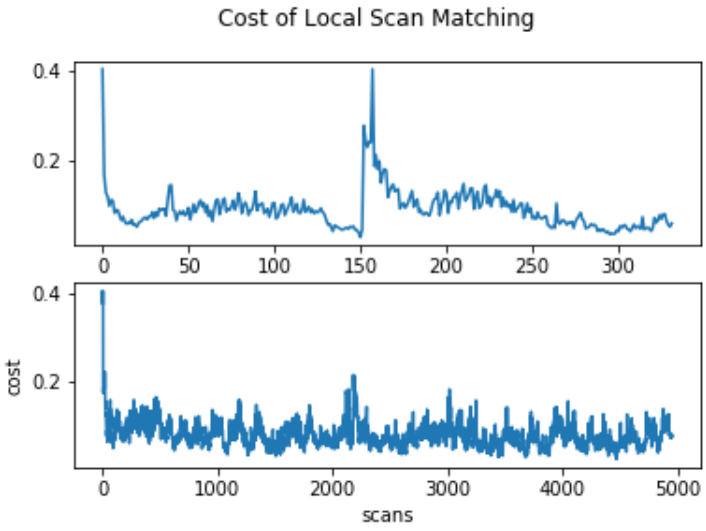

Figure 5.8: Local scan cost of two tests. The top figure shows the results from the first test with kidnapped robot event occurring at 152^{th} scan and the bottom figure shows the results from the second test which does not have kidnapped robot event

The modified ROS bags are firstly used to test KRP detection. The local optimization costs for the KRP and KRP-free cases are shown in Fig.5.8. The cost drastically increases around the 150^{th} step, in which the kidnap event has happened. Fig.5.9 shows the result of applying the average filter and KRP detection algorithm (Algorithm13) to detect the kidnapped event and estimate the time of kidnap. It is clear that the event is readily detected, and from the estimation the event occurs at the 152^{th} scan. After proper tuning, the average filter is able to effectively and quickly detect the KRP event with low very computational cost.

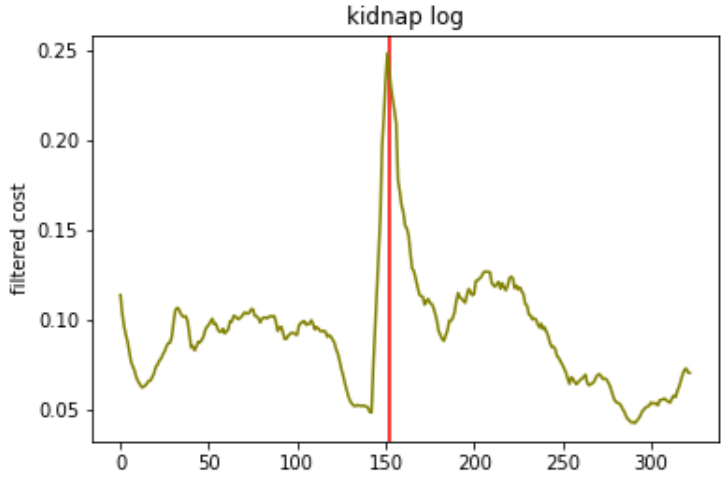

Figure 5.9: This figure shows kidnapped event detection using the proposed average filter, where the KRP happens at $152^{th} scan$

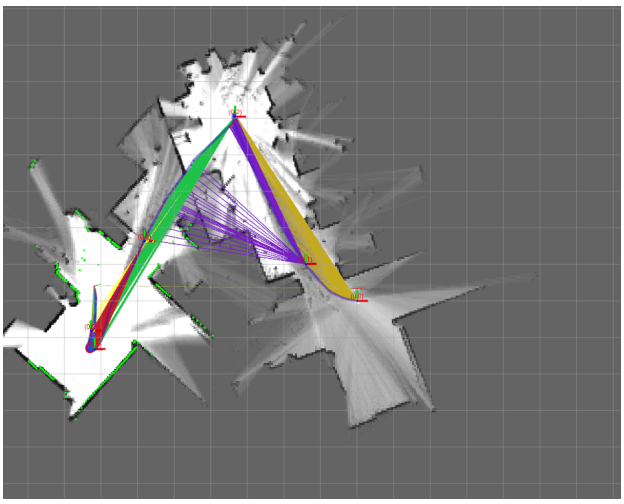

Figure 5.10: The original Cartographer: when KRP occurs, an unreadable map is generated.

KRP recovery experiment

In this experiment, the capability of the KRP recovery is tested. In this case, a new map is generated and merged with the original map when loop closure is detected. The same cropped ROS bag is used for this purpose. Fig.5.10 shows the result of using the original Cartographer algorithm, and the mapping fails when a KRP happens. Because of the KRP, the robot does not know where it is located but still keeps generating new grids on the original map, resulting

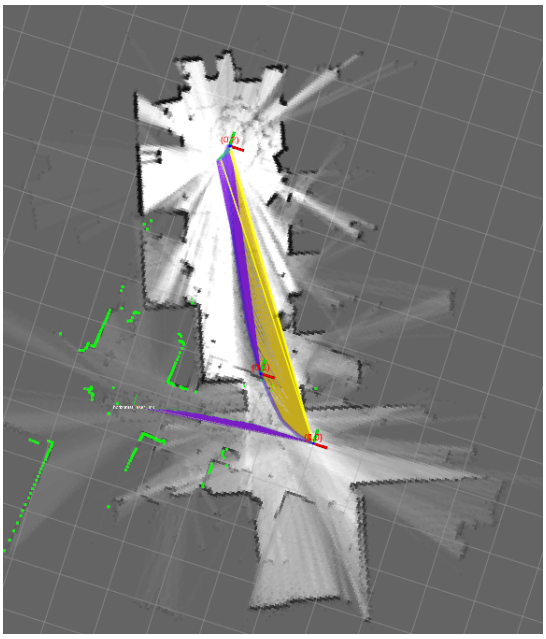

Figure 5.11: The modified program: when KRP occurs, a new map starting from the origin is generated.

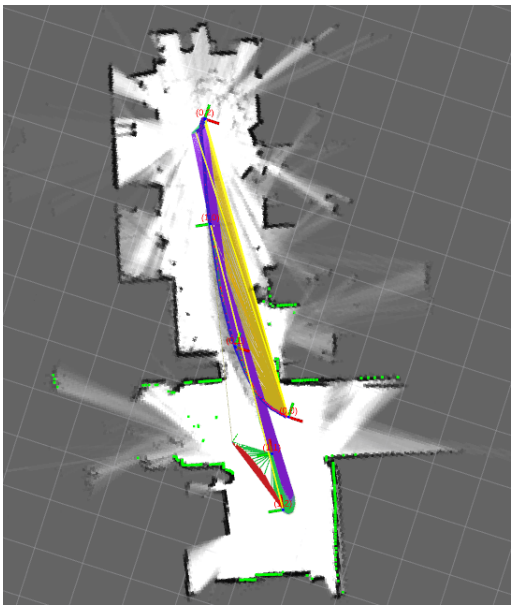

Figure 5.12: The modified program: map merged when loop-closure is detected. The green lines are loop closure constraints, and other colors are for local scan matching constraints.

in an incorrect map.

Fig.5.11 shows the mapping process after KRP, when the modified Cartographer is used. When the KRP is detected, a new map begins to be generated

using the origin as the reference. At this moment, the map is still incorrect since the relationship between the new generated map and the original is not found yet. Then after a loop closure is detected between the current map and a pre-generated map, a loop closure constraint is generated between two maps and the new map is correctly merged to the pre-generated map, see Fig.5.12 as the result. Furthermore, the robot position is successfully identified from the KRP, while both pose graph and map are well merged together. This completes the recovery from KRP.

5.2.3 Discussion

In this chapter, detection of kidnapped robot problem and its recovery is considered. AMCL based and the loop-closure based kidnapping recovery algorithms are developed. The idea behind the AMCL based algorithms is simply to remove bad particles and when KRP occurs, re-localize all particles to the map and find the best one. The implementation is straightforward for particle filter based KRP recovery because some of the MCL libraries can be reused for the KRP recovery step (re-localization). In addition, only a pre-generated map is needed for particle filter based localization and KRP recovery. However, when the map is large or if the algorithm is applied to a 3D case, the number of particles needed will increase for a satisfactory KRP recovery process.

For loop-closure based KRP detection and recovery, we use the Cartographer as a platform. It has better SLAM performance on either mapping or localization, since the global optimization is performed on the map. Also, the loop-closure based algorithm has a better KRP recovery solution based on speed and consistency. As long as the map is built, the recovery can be completed immediately after the KRP event is detected, despite the map's size. On the other hand, in the particle filter based algorithm, it requires more steps for the particle distribution to converge. More importantly, the Cartographer can resolve KRP while working on the mapping process. As for disadvantages of the loop-closure based algorithms, it is relatively more difficult to implement since it involves many conditioning processes to accelerate the SLAM.

Furthermore, the Cartographer has to use the pre-built map, which contains nodes, submaps and other information, for localization process.

Chapter 6

Conclusion

One of the contributions for this thesis is to provide a detailed reference for learning the SLAM. This thesis has presented and experimented with a number of popular SLAM algorithms, which provides the reader with sufficient details and insight to understand and re-implement explained SLAM algorithms. Another contribution is that the thesis proposed a costmap augmented particle filter and a loop-closure based SLAM algorithm for solving the robotic kidnapped problem.

In Chapter 3, we explain and test the filter based SLAM solutions includeing the EKF-SLAM and UKF-SLAM. They use the measurement of features to update current states. The experiment result shows the convergence of both algorithms on robot and landmark position estimation. Also, the UKF-SLAM is more robust to the noise because of the better error covariance estimation. Different from the feature map and KF based methods, applying the pose-graph and loop-closure method further improves the mapping and robot localization performance, which is introduced in chapter 4. Moreover, in Chapter 4 the techniques for visual SLAM are introduced, including processes for visual odometry and matching. Experiments of visual odometry are tested on a series of docking images, in which the ORB method is used for image feature extraction and camera pose trajectory is generated to visualize algorithm performance. After that, in Chapter 5, we propose a costmap based particle filter and a loop-closure based KRP recovery solution. The ACMCL algorithm incorporates the costmap with AMCL method to add the capability

to resample wrong particles detected by the costmap. At last the loop-closure KRP detection and recovery method is developed on the Cartographer package, which generates a new map when the KRP is detected and merges the map when loop-closure is detected.

6.1 Discussion and Future Work

For the KF based SLAM algorithms, the correspondence is one of the major problems. As more feature detection and matching algorithms are invented nowdays (LiDAR and camera based), we can include the feature detection module to filter the data prior to KF-SLAM processes. Another disadvantage for KF based SLAM is that the robot has to have an initial position to have valid SLAM estimation, which means it cannot handle the global localization problem. We can incorporate an external global localization model ahead of the KF-SLAM solution.

The particle filter based SLAM is now among the most commonly used method as it is easy to implement and at the same time can achieve excellent performance when the map is small. Both filter based SLAM algorithms do not handle the accumulated noise well, so in the future, we could implement a global optimization module after the filter based SLAM process.

The visual SLAM implementation is not complete due to the time limitation, where only trajectory estimation, visual bag of word, and loop-closure matching are performed. The next step is to map the extracted feature points and perform BBS as well as pose-graph optimization in the end. Furthermore, the algorithms can be extended and use to other sensors like, depth camera and 3D LiDAR sensor. Also, sensor fusing and application of ML on SLAM can potentially improve the SLAM performance.

Finally, for KRP recovery solutions, we proposed the ACMCL, which augmented the knowledge of the costmap to the AMCL to improve the recovery performance. We can apply other methods to perform the global localization process to accelerate the KRP recovery time. Also, adding additional sensor information may help estimate the robot's location, which would help with

recovery speed. Next, the loop-closure based KRP recovery solution is introduced, and it uses the scan matching cost to detect the KRP event and recover the KRP when loop-closure is detected. The same method can be applied to all graph based SLAM algorithms and modified for a multi-robot problem to merge their maps and estimate the relative pose between robots.

References

[1] N. Akai, T. Hirayama, and H. Murase, "Hybrid localization using model- and learning-based methods: Fusion of monte carlo and e2e localizations via importance sampling," May 2020. 3

[2] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust feature," in *2006 European Conference on Computer Vision (ECCV*, 2006, pp. 404–417. 60, 89

[3] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992. 3

[4] ——, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992. 89

[5] A. Birk and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384–1397, 2006. 89

[6] T. M. Bonanni, B. Della Corte, and G. Grisetti, "3-d map merging on pose graphs," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1031–1038, 2017. 89

[7] I. Bukhori, Z. H. Ismail, and T. Namerikawa, "Detection strategy for kidnapped robot problem in landmark-based map monte carlo localization," in *2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS)*, Langkawi, Oct. 2015, pp. 75–80. 75

[8] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek, "The trimmed iterative closest point algorithm," vol. 16, Feb. 2002, 545–548 vol.3. 3

[9] J. Choi, M. Choi, and W. Chung, "Topological localization with kidnap recovery using sonar grid map matching in a home environment," *Robotics and Computer-integrated Manufacturing - ROBOT COMPUT-INTEGR MANUF*, vol. 28, Jun. 2012. 75

[10] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA99)*, Detroit, MI, USA, May 1999. 75

[11] ——, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation*, vol. 2, 1999, 1322–1328 vol.2. 2, 31

[12] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *I. J. Robotic Res.*, vol. 25, pp. 1181–1203, Dec. 2006. 3

[13] A. Doucet, N. de Freitas, K. Murphy, and S. Russell, *Rao-blackwellised particle filtering for dynamic bayesian networks*, 2013. 2

[14] Feng Lu and Milios, "Robot pose estimation in unknown environments by matching 2d range scans," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 935–938. 3

[15] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. 3, 68

[16] D. Fox, "Kld-sampling: Adaptive particle filters and mobile robot localization," Oct. 2001. 2

[17] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," Jan. 1999, pp. 343–349. 2

[18] D. Galvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012. 3, 68, 89

[19] B. P. Gerkey. (). "Amcl," [Online]. Available: `http://wiki.ros.org/amcl`. 2, 4

[20] ——, (). "Gmapping," [Online]. Available: `http://wiki.ros.org/gmapping`. 4

[21] A. Gordon, H. Li, R. Jonschkowski, and A. Angelova, *Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras*, 2019. 73

[22] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Transactions on Intelligent Transportation Systems Magazine*, vol. 2, pp. 31–43, Dec. 2010. 52–54

[23] S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," Nov. 2003. 3

[24] R. I. Hartley, "In defense of the eight-point algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 6, pp. 580–593, 1997. 60, 65

[25] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. USA: Cambridge University Press, 2003. 64, 67

[26] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 1271–1278. 4, 56, 89, 90, 92–96

[27] M. Himstedt, J. Frost, S. Hellbach, H. Böhme, and E. Maehle, "Large scale place recognition in 2d lidar scans using geometrical landmark relations," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 5030–5035. 4, 89

[28] S. Julier, J. Uhlmann, and H. F. Durrant-Whyte, "A new method for the nonlinear transformation of means and covariances in filters and estimators," *IEEE Transactions on Automatic Control*, vol. 45, no. 3, pp. 477–482, 2000. 15

[29] S. J. Julier and J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004. 15

[30] M. Kaess, A. Ranganathan, and F. Dellaert, "Isam: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008. 3

[31] K. Kanazawa, D. Koller, and S. Russell, *Stochastic simulation algorithms for dynamic probabilistic networks*. 2

[32] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2d mapping," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 22–29. 92

[33] S. A. Kumar. (). "Cartographer," [Online]. Available: `https://github.com/cartographer-project`. 4, 90, 96

[34] M. Labbe and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation: Labbé and michaud," *Journal of Field Robotics*, vol. 36, Oct. 2018. 4

[35] M. Labbé and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based slam," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2661–2666. 88

[36] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," in *Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, 1991, 1442–1447 vol.3. 2

[37] H. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," in *Readings in Computer Vision*, M. A. Fischler and O. Firschein, Eds., San Francisco (CA): Morgan Kaufmann, 1987, pp. 61–62. 64

[38] D. G. Lowe, "Distinctive image features from scale-invariant keypoint," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. 89

[39] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL, Sep. 2014, pp. 709–715. 76

[40] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, pp. 333–349, 1997. 3

[41] F. Martín, R. Triebel, L. Moreno, and R. Siegwart, "Two different tools for three-dimensional mapping: De-based scan matching and feature-based loop detection," *Robotica*, vol. 32, no. 1, pp. 19–41, 2014. 89

[42] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *CoRR*, vol. abs/1502.00956, 2015. 56, 72, 73, 89

[43] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017. 4

[44] D. Nister, "An efficient solution to the five-point relative pose problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 6, pp. 756–770, 2004. 67

[45] E. Olson, "M3rsm: Many-to-many multi-resolution scan matching," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5815–5821. 56

[46] E. B. Olson, "Real-time correlative scan matching," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 4387–4393. 92

[47] E. Olson, J. Leonard, and S. Teller, "Fast iterative optimization of pose graphs with poor initial estimates," 2006, pp. 2262–2269. 89

[48] J. Philip, "A non-iterative algorithm for determining all essential matrices corresponding to five point pairs," *The Photogrammetric Record*, vol. 15, pp. 589–599, Feb. 2003. 67

[49] M. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *Journal of the American Statistical Association*, vol. 94, Nov. 1997. 2

[50] J. Redmon and A. Farhadi, *Yolov3: An incremental improvement*, 2018. 73

[51] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," Nov. 2011, pp. 2564–2571. 60, 62, 89

[52] J. Sanchez, A. Milstein, and E. Williamson, "Robust global localization using clustered particle filtering," *CoRR*, vol. cs.RO/0204044, Jan. 2002. 75

[53] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011. 60

[54] S. Se, D. G. Lowe, and J. J. Little, "Vision-based global localization and mapping for mobile robots," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 364–375, Jun. 2005. 75

[55] R. Sim, P. Elinas, and J. Little, "A study of the rao-blackwellised particle filter for efficient and accurate vision-based slam," *International Journal of Computer Vision*, vol. 74, pp. 303–318, Jul. 2007. 2

[56] J. Sivic and A. Zisserman, "Efficient visual search of videos cast as text retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 591–606, 2009. 3, 68

[57] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986. eprint: `https://doi.org/10.1177/027836498600500404`. [Online]. Available: `https://doi.org/10.1177/027836498600500404`. 2

[58] N. Tatar and H. Arefi, "Stereo rectification of pushbroom satellite images by robustly estimating the fundamental matrix," *International Journal of Remote Sensing*, vol. 40, no. 23, pp. 8879–8898, 2019. 62

[59] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, "Probabilistic algorithms and the interactive museum tour-guide robot minerva," *The International Journal of Robotics Research*, vol. 19, no. 11, pp. 972–999, 2000. 75

[60] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. 7, 8, 13, 16, 19, 21, 22, 3

[61] S. Thrun, D. Fox, and W. Burgard, "Monte carlo localization with mixture proposal distribution," Nov. 2002. 2

[62] E. A. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, 2000, pp. 153–158. 15

[63] J. Welle, D. Schulz, T. Bachran, and A. B. Cremers, "Optimization techniques for laser-based 3d particle filter slam," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 3525–3530. 2

[64] L. Zhang, R. Zapata, and P. Lepinay, "Self-adaptive monte carlo localization for mobile robots using range finders," *Robotica*, vol. 30, no. 2, pp. 229–244, 2012. 75