REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
الجـمـهـوريـة الجـزائـريـة الديـمـقـراطيــة الشـعـبـيـة

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

HIGHER SCHOOL IN APPLIED SCIENCES
--T L E M C E N--

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية
-تلمسان-

Mémoire de fin d'étude

Pour l'obtention du diplôme de Master

Filière : Automatique
Spécialité : Automatique

Présenté par : ZIANI-KERARTI AbdelKarim

Thème

Simultaneous Localization And Mapping

Soutenu publiquement, le 12 / 07 /2021 , devant le jury composé de :

M. CHERKI Brahim	Professeur	ESSA. Tlemcen	Président
M. MOKHTARI Rida	MCA	ESSA. Tlemcen	Directeur de mémoire
M. ARICHI Fayssal	MCB	ESSA. Tlemcen	Co- Directeur de mémoire
Mme. CHOUKCHOU-BRAHAM Amal	Professeur	UNIV. Tlemcen	Examinateur 1
M. BENSALAH Choukri	MCB	UNIV. Tlemcen	Examinateur 2

Année universitaire : 2020 / 2021

ACKNOWLEDGMENTS

First and foremost, I would like to praise and thank God, the almighty, ho has granted countless blessing, and letting me live this amazing experience full of passion and enthusiasm, so that I have been finally able to accomplish the thesis. I would also express my deep and sincere gratitude to my family for their unconditional and unparalleled love, help and support. I am forever indebted to my parents for giving me thee opportunities and experiences that have made me who I am. They selflessly encouraged me to explore new directions in life and seek my own destiny. This journey would not have been possible without them.

Most profound thanks are addressed to Pr. CHERKI Brahim for accepting to be the president of the examiners, and also for his immense knowledge, plentiful experience in several different fields, his working methodology, his enthausiams and his teaching quality that have encouraged me all the time. Special thanks to Pr. CHOUKCHOU-BRAHAM Amal and BEN-SALAH Choukri for the particular interest they have taken in my work and the big pleasure they gave me by accepting to be part of the examiners. Their advices and remarks have been very precious to me.

My sincere thanks are extended to my advisors Dr. MOKHTARI Rida and ARICHI Fayssal for their supervision and technical support.

Apart from my effort, the success of this thesis depends on the encouragements, guidance, and contribution of Dr. RAHMOUN MRABET Rassia, from the School of Applied Sciences (ESSA) Tlemcen.

Finally, it is a pleasure to thank all my friends especially Mohammed, Omar, Fakhro, Reda and Ahmed for the wonderful times we shared and their infinite support.

Contents

List of Figures

INTRODUCTION

The development of SLAM-based robot control system as the integrated approach, which connects the localization, mapping and motion control fields, is presented in this thesis. A lot of robotic research goes into SLAM to develop robust systems for self-driving cars, last-mile delivery robots, security robots, warehouse management, disaster-relief robots and exploration robots. Mobile robots can be wheeled-based, biologically-inspired bipeds and quadrupeds, drones and hybrid soft robotics. These mobile robots however, need extensive hardware, on-board sensing, compute power, and accessories to support SLAM. All robot drive types including holonomic, Ackerman or different drive are suitable for SLAM, and ideally the powered wheel motors have encoder-based feedback. On-board sensing is used to measure the robot's pose as well as interpret the environment and objects within, to create a detailed map. Sensors are responsible to implement closed-loop feedback for control. Nevertheless, poor quality, noisy measurements and inaccurate robot models exacerbate the stochasticity in SLAM. Thus, the mobile robot sensor suite ideally has redundant sensors for each type of measurement also known as **Sensor Fusion**.

SLAM paradigms and Bayesian recursive state and map estimation techniques that include Kalman, and Particle filtering were studied in a detailed way in the SLAM chapter. The solutions considered in this thesis will include the Kalman Filter, which is the backbone of most SLAM research today, as well as the Extended Kalman Filter used for nonlinear systems and environments. Next, we will consider the Particle Filter, which utilizes a non-parametric approach to estimate the robot pose and landmarks location. And finally, we will consider FastSLAM which utilizes the Kalman Filter for estimating the landmarks location and incorporates particle for the pose estimation. These particular algorithms are chosen because they build off one another. The popular and powerful Kalman Filter will be used as a stepping stone on the way to the more state-of-the-art FastSLAM algorithm. This pattern of building up to more complicated algorithms is the main cause for the exponential development of SLAM-based control.

Therefore, the thesis research can be covered by the following chapters:

- **Chapter 1**: presents the research state-of-the-art and motivation, which contains a brief explanation of the state of modern mobile robotics along with taxonomy of the SLAM problem, and finally some of the recent applications of the SLAM problem in the field of robotics.

- **Chapter 2**: covers a brief mathematical description of the SLAM problem as a probabilistic approach with originates from the Bayes rule and Markov assumption. The Bayesian recursive estimation definition and the Bayes Filter implementations, like Kalman Filer, Extended Kalman Filter and a non-parametric approaches like Particle Filter and Fast-SLAM.
- **Chapter 3**: consists of a probabilistic formulation of the state transition probability density and the measurement density a mobile wheelded robot with non-holonomic constraints.
- **Chapter 4**: The simulation results of the described SLAM problem algorithms for the robot pose estimation and map featuring problem applied to the 2D wheelded robot.

CHAPTER

1 STATE OF THE ART

1.1 Introduction

The field of robotics has gone through a series of paradigms for software design. The first major paradigm emerged in the mid-1970s, and is known as the "model-based paradigm". The model-based paradigm began with a number of studies showing the hardness of controlling a high DoF robotics manipulator in continuous space. A first singly exponential general motion planning algorithm by Canny, [3] and Latombe's [1] seminal introductory text into the field of model-based motion planning. This early work largely ignored the problem of uncertainty even though it extensively began using randomization as a technique for solving hard motion planning problems [11]. Instead, the assumption was that a full and accurate model of the robot and the environment be given, and the robot be deterministic. The model had to be sufficiently accurate that the residual uncertainty was managed by a low-level motion controller. Most motion planning techniques simply produced a single reference trajectory for the control of the manipulator, although ideas such as Potential Fields [12] and Navigation Functions [18] provided mechanisms for reacting to the unforeseen as long as it could be sensed. Applications of these early techniques, if any, were confined to environments where every little bit of uncertainty could be engineered away, or sensed with sufficient accuracy. [22]

The historical roots of SLAM can be traced back to Gauss [6], who is largely credited for inventing the least mean squares method. for calculating planetary orbits. In the Twentieth Century, a number of fields outside robotics have studied the making of environment models from a moving sensor platform, most notably in Photogrammetry and Computer Vision. SLAM builds on this work, often extending the basic paradigms into more scalable algorithms. [24]

1.2 Motivation

Many things have changed since the time when, in the early 1920s, Karl Čapek first called a mechanical creature a **Robot**. The word Robot originates from a Czech word which means 'Work'. These machines, which are "mechanically perfect" and have "an enormously developed intelligence".In a very short time, they became an inspiration and one of the main research fields in mechanical and artificial intelligence sciences [20]. During many decades, they consequently went from science-fiction through the first laboratory prototype, to efficient "workers" in probably all engineering fields, especially industrial ones. Novel technologies and a rapid scientific progress resulted in the big robotics "revolution" at the turn of XX and XXI centuries, which introduced robots for social life of many countries. One of the most rapidly developing fields in robotics science in mobile robots, which are constructed to perform different tasks, such as inspection or servicing. Currently, modern mobile robots can perform the tasks of unknown space explorers, ordinary home cleaners, medical servants in hospitals or even pets for children. And they are no longer as exotic or expensive as they were before. [20]

The general SLAM problem has been the subject of substantial research since the inception of robotics research community and indeed before this in areas such as Manned Vehicle Navigation systems and geophysical surveying. A number of approaches have been proposed to address both the SLAM problem and also more simplified navigation problems where additional map or vehicle location information is made available. Broadly, these approaches adopt one of three main philosophies. The most popular of these is the estimation-theoretic or Kalman Filter based approach. The popularity of this approach is due to two main factors. Firstly, it directly provides both a recursive solution to the navigation problem and means of computing consistent estimates for the uncertainty in vehicle and map landmark locations on the basis of statistical models for vehicle motion and relative landmark observations. Secondly, a substantial corpus of method and experience has been developed in aerospace, maritime and other navigation applications, from which the autonomous vehicle community can draw. A second philosophy is to eschew the need for absolute position estimates and for precise measures of uncertainty and instead to employ more qualitative knowledge of the relative location of landmarks and vehicle to build maps and guide motion [5]. This general philosophy has been developed by a number of different groups in a number of different ways: The qualitative approach to navigation and the general SLAM problem [[2], [14], [15]] has many potential advantages over the estimation-theoretic methodology in terms of limiting the need for accurate models and the resulting computational requirements and in its significant "anthropomorphic appeal". The third very broad philosophy, does away with the rigorous Kalman Filter or statistical formalism while retaining an essentially numerical or computational approach to the navigation and SLAM problem. Such approaches include the use of iconic landmark matching [26] global map registration [4] bounded regions [9], and other measures to describe uncertainty. Notable are the work by Thrun and Yamauchi [22], [25] use a Bayesian approach to map building that does not assume Gaussian probability distributions as required by the Kalman Filter. This techniques while very effective for localization with respect to maps, does not lend itself to provide an incremental solution to SLAM where a map is gradually build as information is received from sensors. Yamauchi [25] use a evidence grid approach that requires that the environment is decomposed to a number of cells.

An estimation-theoretic or Kalman Filter based approach to the SLAM problem is adopted later in the next chapter. A major advantage to this approach is that it is possible to develop a complete proof of the various proprieties of the SLAM problem and to study systematically the evolution of the map and the uncertainty in the map and the vehicle location [21]. A proof of existence and convergence for a solution of the SLAM problem within a formal estimation-theoretic framework work also encompasses the widest possible range of navigation problems and implies that solutions to the problem using other approaches are possible. [5]

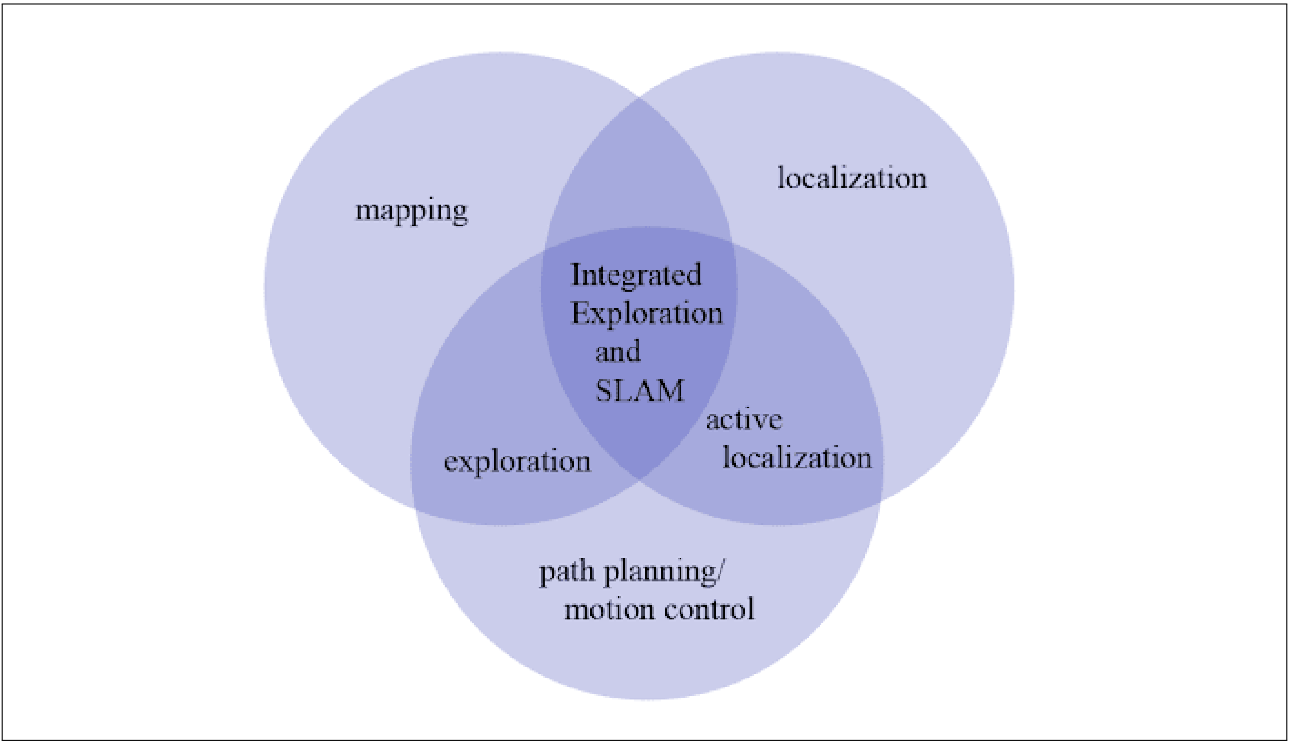

mapping
localization
Integrated Exploration and SLAM
active localization
exploration
path planning/ motion control

Figure 1.1: The fields of mobile robotics connected by the integrated approach

1.3 Taxonomy of the SLAM problem

SLAM problems are recognized along a number of different dimensions. Most important research papers identify the problem type by making the underlying assumptions explicit. The most common distinctions encountered in the literature are as follows: [20]

- **Volumetric versus Feature-based**. In volumetric SLAM, the map is sampled at a resolution high enough to allow for photo-realistic reconstruction of the environment. The map m in volumetric SLAM is usually quite high-dimensional, with the result that the computation can be quite involved. Feature-based SLAM extracts sparse features from the sensor stream. The map is then only comprised of features. Feature-based SLAM techniques tend to be more efficient, but their results may be inferior to volumetric SLAM due to the fact that the extraction of features discards information in the sensor measurements.

- **Topological versus Metric**. Some mapping techniques recover only a qualitative description of the environment, which characterizes the relation of basic locations. Such methods are known as topological. A topological map might be defined over a set of distinct places and a set of qualitative relations between theses places (e.g place A is adjacent to place B). Metric SLAM methods provide metric information between the relation of such places. In recent years, topological methods have fallen out of fashion, despite ample evidence that humans often use topological information for navigation.

- **Known versus Unknown correspondence**. The correspondence problem is the problem of relating the identity of sensed things to other sensed things. The problem if estimating the correspondence is known as data association problem and it is one of the most difficult problems in SLAM.

- **Static versus Dynamic**. Static SLAM algorithms assume that the environment does not change over time. Dynamic methods allow for changes in the environment. The vast literature on SLAM assumes static environments. Dynamic effects are often treated just as measurements outliers. Methods that reason about motion in the environment are more involved, but they tend to be more robust in most applications.

- **Small versus Large uncertainty**. SLAM problems are distinguished by the degree of location uncertainty that they can handle. The most simple SLAM algorithms allow only for small errors in the location estimate. They are good for situations in which a robot goes down a path that does not intersect itself, and then returns along the same path. In many environments it is possible to reach the same location from multiple directions. Here the robot may accrue a large amount of uncertainty. This problem is known as the **loop closing problem**. When closing a loop, the uncertainty may be large. The ability to close loops is a key characteristic of modern-day SLAM algorithms. The uncertainty can be reduced if the robot can sens information about its position in some absolute coordinate frame, e.g through the use of satellite -based global positioning receiver (GPS).

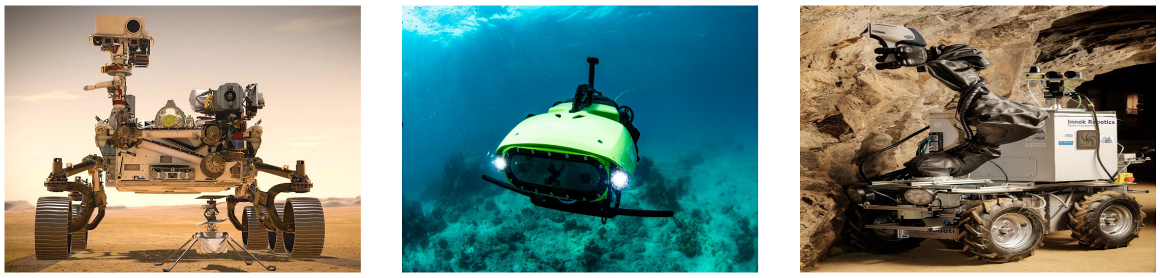
- **Active versus Passive**. In passive SLAM algorithms, some other entity controls the robot, and the SLAM algorithm is purely observing. The vast majority of algorithms are of this type, they give the robot designer the freedom to implement arbitrary motion controllers, and pursue arbitrary motion objectives. In active SLAM, the robot actively explores its environment in the pursuit of an accurate map. Active SLAM methods tend to yield more accurate maps in less time, but they constrain the robot motion. There exist hybrid techniques in which the SLAM algorithm controls only the pointing direction of the robot's sensors, but not the motion direction.

- **Single-robot versus Multi-robot**. Most SLAM problems are defined for a single-robot platform, although recently the problem of multi-robot exploration has gained in popularity. Multi-robot SLAM problems are also distinguished by the type communication allowed between the different robots. In some, the robots can communicate with no latency and infinite bandwidth. More realistic are setups in which only nearby robots can communicate, and the communication is subject to latency and bandwidth limitations.

As this taxonomy suggests, there exists a flurry of SLAM algorithms. Most modern-day conferences dedicate multiple session to SLAM. We will manly focus on most simple and basic SLAM setups. In particular we assume a static environment with a single robot.

1.4 SLAM Application

SLAM is an essential capability for mobile robots travelling in unknown environments where globally accurate position data is not available. In particular, mobile robots have shown significant promise for remote exploration, going places that are too distant [7], too dangerous [23], or simply too costly to allow human access. If robots are to operate autonomously in



(a) Other planets (Perseverance) (b) Undersea LarvalBot (c) Underground (ARIDuA)

Figure 1.2: Different type of robots using SLAM used for different environments

extreme environments undersea, underground, and on the surfaces of other planets, they must be capable of building maps and navigating reliably according to these maps. Even in benign environments, such as the interiors of buildings, accurate, prior maps are often difficult to acquire. The capability to map an unknown environment allow a robot to be deployed with minimal infrastructure. This is especially important if the environment changes over time. [20]

The maps produced by SLAM algorithms typically serve as the basis for motion planning and exploration. However, the maps often have value in their own right. In July of 2002, nine miners in the Quecreek Mine in Sommerset Pennsylvania were trapped underground for three and a half days after accidentally drilling into a nearby abandoned mine. A subsequent investigation attributed the cause of the accident to inaccurate maps [10]. Since the accident, mobile robots and SLAM have been investigated as possible technologies for acquiring accurate maps of abandoned mines. One such robot, show in the figure 1.2c below, is capable of building 3D reconstructions of the interior of abandoned mines using SLAM technology [23]

CHAPTER

2

SIMULTANEOUS LOCALIZATION AND MAPPING

2.1 Introduction

Simultaneous Localization and Mapping (SLAM) has been an immensely well know, and expensively investigated topic among the mobile robotics community for more than two decades. The accomplishments of this field is tightly bound to the fact that solving the SLAM problem, that is locating a moving robot in an unknown environment and incrementally build a consistent map of this environment while simultaneously determining its location within this generated map at anytime. SLAM has increasingly various applications ranging from spacial exploration to autonomous driving.

SLAM problem has been one of the eminent achievement of the robotics community over the past years. It has been formulated and solved as a theoretical problem in various forms, and it has also been carried out in different areas from raging from indoor robots, to outdoor, underwater and airborne systems. From a conceptual and theoretical point of view, SLAM can now be considered a solved problem. However, scalability and flexibility in SLAM should be asserted more often, they are difficult to perform in large-scale experiments even with inexpensive robots. Flexibility should be achieved within reasonable constraints, for example, a method that works indoor but not outdoor is not flexible, but a method that requires a chains to be added to the robot wheels to allow them to operate in the snow could still be considered flexible. In a general manner, localizing a vehicle whether in a global or a local frame, is a fundamental usefulness to perform some other insight or planification tasks. Anticipating the evolution of other obstacles on the map, and choosing what maneuver is the most suitable, requires knowing precisely where the vehicle is located and how it will evolve in the coming seconds. The SLAM Framework offers a response to this problematic while as yet being sufficiently general to permit the utilization of any sensor or estimation technique that suits the prerequisite of estimating both the localization of the vehicle and the map at the same time.

The map is of prime interest when autonomous driving is considered as a whole as it offers the first level of perception needed in order to make appropriate decisions.

2.2 SLAM: Problem Definition

2.2.1 Mathematical Basis

The SLAM problem is typically formalized in a probabilistic terminology. The goal is to be able to estimate at the same time the state of the moving robot and the map being built. The robot state can be characterized depending on the application: 2D position and orientation, 6D position, speed, acceleration... etc. We denote x_k the vehicle position estimate at the time k, and m the map of the environment. To estimate these variables, it is possible to take advantage of what we call control inputs u_k and which represent an estimation of the motion between $k-1$ and k. They usually come from wheel encoders or any sensor able to give a first idea of the displacement. For noise-free motion, u_k would be sufficient to recover the past x_k from the initial location x_0. However odometry measurements are noisy and path integration techniques inevitably diverge from reality, hence error in the robot's path correlates errors in the map. The particularity of SLAM approaches is to take into consideration measurements coming from sensor readings denoted by z_k to help build and improve the map and indirectly, estimate the vehicle position. The map m is typically assumed to be time-invariant, and the environment may comprised of landmarks, objects, and surfaces and their locations is described by the estimated map m.

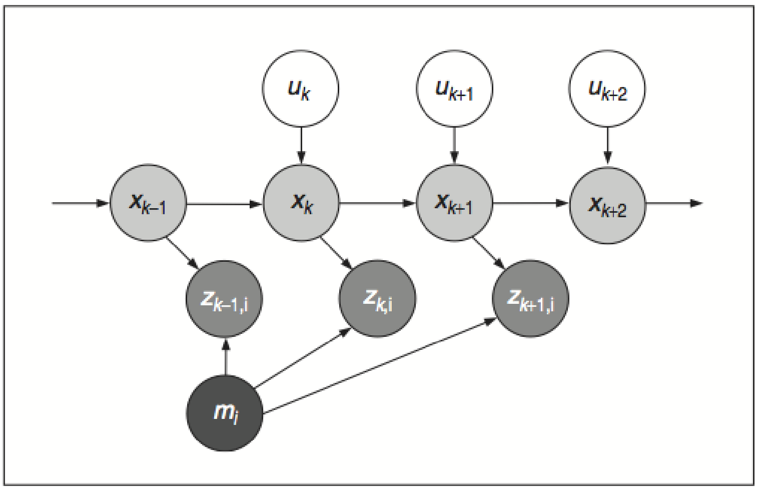


Figure 2.1: Graphical model of the SLAM problem

Figure 2.1 illustrates the various variables involved in the SLAM problem. It shows the sequence of locations and sensor measurements, and the causal relationship between them.

2.2.2 SLAM Posterior

As discussed earlier, the robot's position will be denoted as x_k. For a robot operating in a planar environment, this position consists of the robot's x-y position in the plane and its heading direction. The sequence of locations, or path in given as:

$$x = \{x_0, x_1, x_2, ..., x_k\} \tag{2.1}$$

Where k is either finite (terminal time) or infinite, and x_0 the initial location is know as opposed to other locations that cannot be sensed. We shall further assume that the robot's environment can be modeled as a set of N immobile, point landmarks, which represent the position of features extracted from sensor data, such as laser scanners and cameras. While moving through the environment, the robot extract relative information about its motion using sensors such as encoders attached to the robot's wheels, inertial measurement units , or simply observing the control commands executed by the robot. The set of all controls executed is written as:

$$u = \{u_0, u_1, u_2, ..., u_k\} \tag{2.2}$$

The robot measurements highlight the nearby observed landmarks and obstacles, then establishes information between features in the map m and its locations x_k. We will assume that the robot takes exactly one measurement as each point in time, the set of all measurements is:

$$z = \{z_0, z_1, z_2, ..., z_k\} \tag{2.3}$$

The primary SLAM goal now is to recover the best estimate of the world m and the robot pose x_k, given the set of noisy odometry data (observation) z_k and controls u_k. We distinguish two main forms of the SLAM problem which are both of equal practical importance. One is know as the **full-SLAM** problem: it involves estimating the posterior over the entire robot path together with the world's map:

$$p(x_{1:k}, m | z_{1:k}, u_{1:k}) \tag{2.4}$$

Written this way, the full SLAM problem is the problem of calculating the joint posterior over x_k and m from the available data. Notice that the variables right of the conditioning bar are all directly observable to the robot, whereas those on the left are the ones that we wanted. As we shall see, algorithms for the offline SLAM problem are often batch, that is, they process all data at the same time. [20] The second, equally important SLAM problem is the online SLAM problem, it is defined via:

$$p(x_k, m | z_{1:k}, u_{1:k}) \tag{2.5}$$

Online SLAM seeks to recover the present robot location, instead of the entire path. Algorithms that address the line problem are usually incremental and can process one date item at a time. In the engineering literature, such algorithms are called filters. [20]

The essential SLAM problem is illustrated in the figure 2.2 below. A simultaneous estimate of both the robot and landmark locations. The true locations are never known or measured directly, observations are made between true robot and landmark locations.

To solve the SLAM problem, the robot should be enriched with two more models: a mathematical model that relates the odometry measurements u to the robot's locations at any two successive time steps x_{k-1} and x_k, and a model that relates the observations captured by sensors and the robot location x_k. It is common to think of those mathematical models as probability distributions: $p(x_k | x_{k-1}, u_k)$ characterizes the probability distribution of the robot's location x_k assuming that he started at a known location x_{k-1} and measured the odometry data

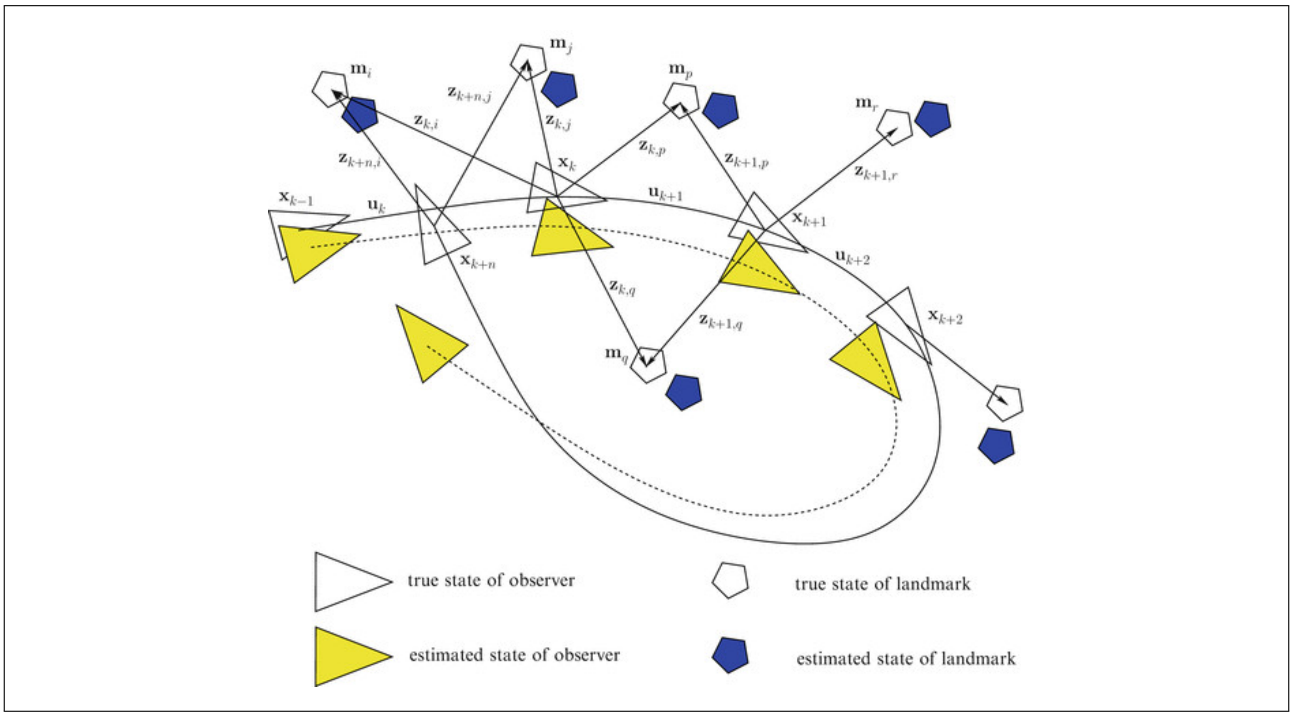


Figure 2.2: SLAM problem

u_k. Likewise, $p(z_k|x_k, m)$ is the probability for measuring z_k if the measurement is taking at a known location x_k in a known environment m. Of course, in the SLAM problem, either the robot location and the environment are unknown, so we need a way to recover probability distributions over latent variables from the measured data.

Bayes Filter Derivation

The Bayes Filter derivation from the SLAM posterior is straightforward, using the Bayes theorem into equation 2.4, the recursive posterior estimation looks like follows:

$$p(x_{1:k}, m|z_{1:k}, u_{1:k}) = \eta\ p(z_k|x_k, m, z_{1:k-1}, u_{1:k})\ p(x_k, m|z_{1:k-1}, u_{1:k}) \tag{2.6}$$

The denominator from Bayes theorem is a normalizing constant η. Note that z_k is a solely a function of the robot's pose x_k and the map m. Applying Markov assumption to 2.6 that states that x_k is a result of all history information from 0 to k that we can encode. Simplifying the probability of observation at time k gives us:

$$p(x_{1:k}, m|z_{1:k}, u_{1:k}) = \eta\ p(z_k|x_k, m)\ p(x_k, m|z_{1:k-1}, u_{1:k}) \tag{2.7}$$

The rightmost term of the above equation can further be simplified using the same Markov assumption as before and the Theorem of Total Probability at time $k-1$:

$$= \eta\ p(z_k|x_k, m) \sum_{x_{k-1}\in\mathbb{X}} p(x_k, m|x_{k-1}, z_{1:k-1}, u_{1:k})p(x_{k-1}, m|z_{1:k-1}, u_{1:k}) \tag{2.8}$$

Note that the robot's pose x_k is a function of x_{k-1} and u_k, we can think of it as the control command needed to get the robot from x_{k-1} to x_k or the robot motion model, this simplifies further the first term inside the previous summation:

$$= \eta\ p(z_k|x_k, m) \sum_{x_{k-1}\in\mathbb{X}} p(x_k|x_{k-1}, u_k)p(x_{k-1}, m|z_{1:k-1}, u_{1:k}) \tag{2.9}$$

We can drop u_k from the rightmost term in summation because it provides no information about the robot's latest pose x_k or the map m without the latest observation z_k. The result is a recursive formula 2.10 for computing the SLAM posterior at time k given the SLAM posterior at time $k-1$:

$$= \eta\, p(z_k|x_k, m) \sum_{x_{k-1}\in\mathbb{X}} p(x_k|x_{k-1}, u_k) p(x_{k-1}, m|z_{1:k-1}, u_{1:k-1}) \tag{2.10}$$

The definition of recursive state and map estimation by the previous equation (2.10) is called Full-SLAM or a smoothing problem. The estimation here is going through a full way with recuperating every conceivable poses, observations and controls for every time step. The recursive formula also depend explicitly on two probability functions, the motion model $p(x_k|x_{k-1}, u_k)$, and the observation model $p(z_k|x_k, m)$.

This approach is much more efficient when very accurate full path is searched and task has enough computing resources and time for calculations. During previous years many researchers is SLAM community were focusing on robust optimization of smoothing, and its variant **incremental smoothing** $(p(x_k, m|z_{1:k}, u_{1:k}))$, optimization mostly based on the least mean square problem solutions. [13]

The figure [2.3] below illustrates the graphical Bayesian Network of the full-SLAM problem:

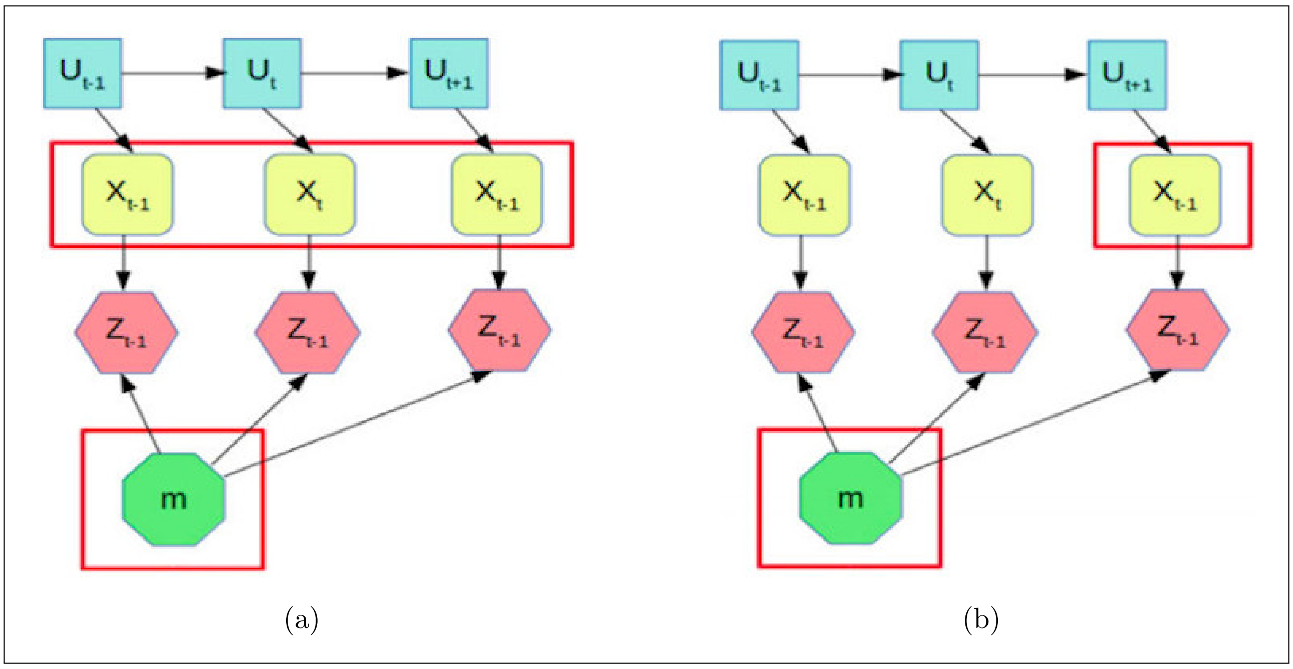


Figure 2.3: Graphical representation by Bayesian network of online-SLAM

Filtering is a most utilized SLAM approach, especially since it permits to have the recent information in a quick time without the need for heavy computing resources. This estimation is beneficial for short-term usage and sufficient for path planning or mapping systems that is being applied in autonomous vehicles application usage. [16] In SLAM researcher community, the three main paradigms are existing independently. There are particle filters, Kalman filters and graph-based or topological SLAM. We will look at the essential and regular executions of each SLAM paradigms in the next part of this chapter. [16]

2.3 Gaussian Filters

Historically, Gaussian filters are the earliest implementations of the recursive Bayes filter for continuous spaces. They are wildly used in various domains including navigation and guidance for spacecrafts to signal processing and economics despite a number if shortcomings.

Gaussian filters are based on the basic idea that beliefs are represented by multivariate normal distributions.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{2.11}$$

The probability distribution over the variable x is characterized by two parameters: the mean μ and the covariance Σ. Such parameterization is called the moments parameterization because the mean and covariance are the first and second moments of probability distribution. The mean μ is a column vector with the same dimension as x, and the covariance is a symmetric positive-semidefinite matrix with the same dimensionality as the state vector x but squared. Thus, the number of elements in the covariance matrix depends quadratically on the number of elements in the state vector.

The commitment to represent the posterior by a Gaussian has important ramifications. Most importantly, Gaussian are unimodal, they posses a single maximum. Such a posterior is characteristic of many tracking problems in robotics, in which the posterior is focused around the true state with a small margin of uncertainty. Gaussian posteriors are a poor match for many global estimation problems in which many distinct hypotheses exist, each of which forms its own mode in the posterior. [22]

2.3.1 Linear Kalman Filter

One of the most studied techniques for implementing recursive Bayes filters is the linear Kalman filer or KF. The Kalman filter is an algorithm that uses a series of measurements observed over time, containing statistical noise for filtering and prediction in Linear Systems defined by the recurrence relation with Gaussian noises added, where A_k is the state matrix, B_k, the input matrix, C_k the output matrix, ϵ_k and δ_k are random Gaussian vectors that model the uncertainty introduced by the state transition and the measurement noise respectively, with zero mean and covariance R_k and Q_k respectively:

$$\begin{cases} x_{k+1} = A_k x_k + B_k u_k + \epsilon_k \\ z_k = C_k x_k + \delta_k \end{cases} \tag{2.12}$$

The Kalman filter uses the moment parameterization to represent beliefs. At time k, the belief is represented by the mean μ_k and the covariance Σ_k. In addition to the Markov assumption of the Bayes filer, posteriors are Gaussian if the system in question has a linear state transition probability $p(x_{k+1}|u_k, x_k)$, the measurement probability $p(z_k|x_k)$ is also linear and finally, the initial belief $p(x_0)$ must be normally distributed with a mean μ_0 and covariance Σ_0

The state transition probability, the measurement probability and the initial belief are obtained by plugging the equation [2.12] into the definition of the multivariate normal distribution

[2.11]:

$$bel(x_{k+1}) = \det(2\pi R_k)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_{k+1} - A_k x_k - B_k u_k)^T R_k^{-1}(x_{k+1} - A_k x_k - B_k u_k)\right\} \tag{2.13}$$

$$bel(z_k) = \det(2\pi Q_k)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_k - C_k x_k)^T Q_k^{-1}(z_k - C_k x_k)\right\} \tag{2.14}$$

$$bel(x_0) = \det(2\pi \Sigma_0)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1}(x_0 - \mu_0)\right\} \tag{2.15}$$

These assumptions are sufficient to ensure that the posterior $bel(x_{k+1})$ is always Gaussian at anytime.

The Kalman Filter Algorithm

As described earlier, the Kalman filter represent the belief $bel(x_k)$ at time k. The input of the filter is the belief at time $k-1$, represented by the μ_{k-1} and Σ_{k-1}. To update these parameters, The Kalman filter requires the control u_k and the measurement z_k. The output is the belief at time k represented by the mean μ_k and the covariance Σ_k. The algorithm is presented below:

Algorithm 1 Kalman Filter

```
Inputs: $\mu_{k-1}$, $\Sigma_{k-1}$, $u_k$, $z_k$
Prediction
    $\overline{\mu}_k = A_k \mu_{k-1} + B_k u_k$
    $\overline{\Sigma}_k = A_k \Sigma_{k-1} A_k^T + R_k$
Kalman Gain
    $K_k = \overline{\Sigma}_k C_k^T (C_k \overline{\Sigma}_k C_k^T + Q_k)^{-1}$
Correction
    $\mu_k = \overline{\mu}_k + K_k(z_k - C_t \overline{\mu}_k)$
    $\Sigma_k = (I - K_k C_k)\overline{\Sigma}_k$
return $\mu_k$, $\Sigma_k$
```

The predicted $\overline{\mu}_k$ and $\overline{\Sigma}_k$ are calculated representing the belief $\overline{bel}(x_k)$ one time step later, but before incorporating the measurement z_k. This belief is obtained by incorporating the control u_k The mean is updated using the deterministic version of the state function described by the equation [2.12], with the state x_{k-1} substituted by the mean μ_{k-1}. The update of the covariance considers the fact that states depend on previous states through the linear state matrix A_k. This matrix is multiplied twice into the covariance, since the covariance is a quadratic matrix. [22]

The predicted belief $\overline{bel}(x_k)$ is subsequently transformed into the desired belief $bel(x_k)$ by incorporating the measurement z_k. The variable K_k is called the Kalman gain used to specify the degree to which the measurements is incorporated into the new state estimate. The correction is then made to the mean by adjusting it in proportion to the Kalman gain and the deviation of the actual measurement z_k and the measurement predicted according to the probability from the second equation [2.12]. Finally the new covariance of the posterior belief is calculated, adjusting for the information gain resulting from the measurement. [22]

The figure 2.4 illustrates the Kalman filter algorithm for a basic localization scenario. Suppose that the robot moves along the horizontal axis in each diagram. Let the prior over the robot location be given by the normal distribution shown in figure 2.4a The robot queries its sensors on its location (e.g., a GPS system), and those return a measurement that is centered at the peak of the bold Gaussian in Figure 2.4b. This bold Gaussian illustrates this measurement: Its peak is the value predicted by the sensors, and its width (variance) corresponds to the uncertainty in the measurement. Combining the prior with the measurement, via lines 4 through 6 of the Kalman filter algorithm presented above, yields the bold Gaussian in Figure 2.4c. This belief's mean lies between the two original means, and its uncertainty radius is smaller than both contributing Gaussians. The fact that the residual uncertainty is smaller than the contributing Gaussians may appear counter-intuitive, but it is a general characteristic of information integration in Kalman filters.

Next, assume the robot moves towards the right. Its uncertainty grows due to the fact that the state transition is stochastic. Lines 2 and 3 of the Kalman filter provide us with the Gaussian shown in bold in Figure 2.4d. This Gaussian is shifted by the amount the robot moved, and it is also wider for the reasons just explained. The robot receives a second measurement illustrated by the bold Gaussian in Figure 2.4e which leads to the posterior shown in bold in Figure 2.4f.

As this example illustrates, the Kalman filter alternates a measurement update step (lines 5-7), in which sensor data is integrated into the present belief, with a prediction step (or control update step), which modifies the belief in accordance to an action. The update step decreases and the prediction step increases uncertainty in the robot's belief. [22]

2.3.2 Extended Kalman Filter

The suppositions that observations are linear functions of the state and that the state equation is a linear recurrent formula with respect to the previous state are pivotal for the correctness of the Kalman Filter. The observation that any linear transformation of a Gaussian random variable results in another Gaussian random variable played an important role in the derivation of the Kalman Filter algorithm. The Kalman filter's efficiency is then because the parameters of the resulting Gaussian can be computed in closed form.

In many robotic problems, the state transitions and measurements are nonlinear, which makes plain Kalman Filters inapplicable. For this, the Extended Kalman Filter (EKF) relaxes on the assumption that the state transition probability and the measurements probabilities are governed by nonlinear functions g and h respectively:

$$\begin{cases} x_k = g(x_{k-1}, u_k) + \epsilon_k \\ z_k = h(x_k) + \delta_k \end{cases} \tag{2.16}$$

The function g replaces the state matrix A_k and the input matrix B_k for the linear state transition, and the function h replaces the output matrix C_k of the measurement as described by the equation [2.12]. This nonlinear model generalizes the Linear Gaussian model underlying Kalman filters. Note that, with arbitrary function g and h, the beliefs $bel(x_k) = p(x_k|u_k, x_{k-1})$ and $bel(z_k) = p(z_k|x_k)$ are no longer Gaussian. Therefore, performing belief update exactly is impossible and the Bayes filter does not possess a closed-form solution.

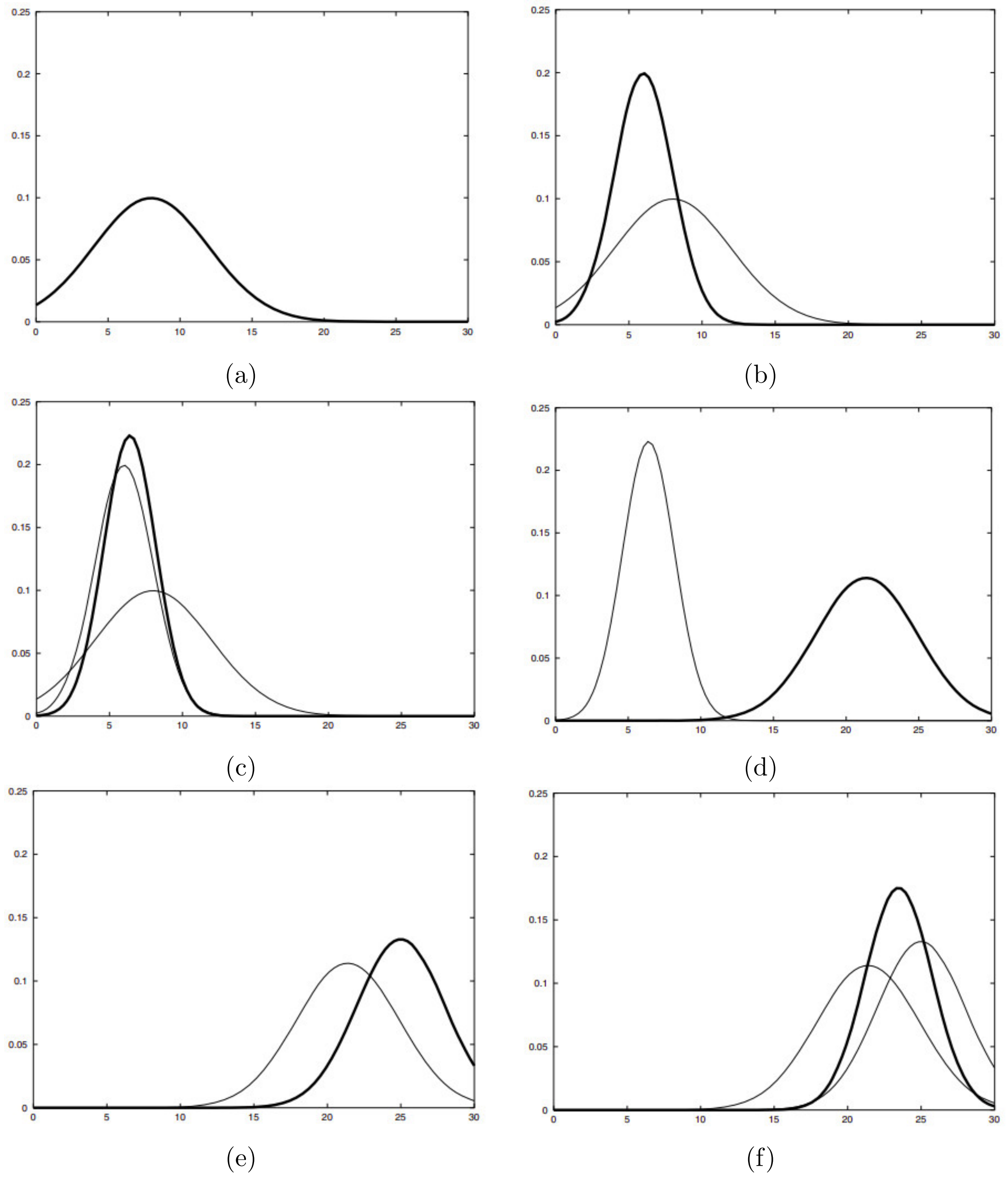


Figure 2.4: Illustration of Kalman filters

The Extended Kalman Filter calculates a Gaussian approximation to the true belief, it represents the true belief $bel(x_k)$ at time k by a mean μ_k and covariance Σ_k. Thus, the Extended Kalman filter inherits from the Kalman filter the basic belief representation, but it differs in that this belief is only approximate, not exact as was the case in Kalman filters. The goal is thus shifted from computing the exact posterior to efficiently estimating its mean and covariance. However, since these statistics cannot be computed in closed form, the Extended Kalman filer has to resort to an additional approximation.

The key idea underlying the Extended Kalman filter approximation is called linearization. linearization approximates the nonlinear function g by a linear function that is tangent to g at the mean of the Gaussian, and projecting the Gaussian through this linear approximation results in a Gaussian density. The major advantage of the linearization, however lies in its efficiency, approximating the nonlinear state transition and measurement probabilities by linear approximations help retain the Gaussian nature of the posterior belief, Thus, the mechanics of the Extended Kalman filter belief propagation are equivalent to those of the linear Kalman filer.

Extended Kalman filter utilizes a linearization method called first order **Taylor Expansion**. Taylor expansion constructs a linear approximation to the function g by its value at the mean of the posterior μ_{k-1}, then, a linear extrapolation is achieved by adding an additional term proportional to the gradient of g and μ_{k-1} and u_k:

$$\begin{aligned} g(x_{k-1}, u_k) &\approx g(\mu_{k-1}, u_k) + \frac{\partial g(x_{k-1}, u_k)}{\partial x_{k-1}}(x_{k-1} - \mu_{k-1}) && (2.17)\\ &\approx g(\mu_{k-1}, u_k) + G_k(x_{k-1} - \mu_{k-1}) && (2.18)\end{aligned}$$

The matrix G_k is called the Jacobian matrix and its value depends on u_k and μ_{k-1}, hence it differs for different points in time.

Written as Gaussian, the state transition probability is approximated as follows:

$$\begin{aligned} &p(x_k|x_{k-1},u_k)\\ &\quad\approx \det(2\pi R_k)^{-\frac{1}{2}} \exp\{-\frac{1}{2}[x_k - g(\mu_{k-1}, u_k) - G_k(x_{k-1} - \mu_{k-1})]^T \\ &\quad R_k^{-1}[x_k - g(\mu_{k-1}, u_k) - G_k(x_{k-1} - \mu_{k-1})]\} \end{aligned} \tag{2.19}$$

For the measurement fucntion h, Extended Kalman filter implent the exact same linearization. However, the Taylor expansion is developed around $\overline{\mu}_k$. The linear measurement approximate is obtained as follows:

$$\begin{aligned} h(x_k) &\approx h(\overline{\mu}_k) + \frac{\partial h(x_k)}{\partial x_k}(x_k - \overline{\mu}_k) && (2.20)\\ &\approx h(\overline{\mu}_k) + H_k(x_k - \overline{\mu}_k) && (2.21)\end{aligned}$$

Written as Gaussian, we have:

$$\begin{aligned} p(z_k|x_k) \approx &\det(2\pi Q_k)^{-\frac{1}{2}} \exp\{-\frac{1}{2}[z_k - h(\overline{\mu}_k) - H_k(x_k - \overline{\mu}_k)]^T \\ &Q_k^{-1}[z_k - h(\overline{\mu}_k) - H_k(x_k - \overline{\mu}_k)]\} \end{aligned} \tag{2.22}$$

Extended Kalman Filter Algorithm

The Extended Kalman Filter algorithm is similar to the linear Kalman Filter except using the nonlinear state transition and measurement for the prediction. Moreover, Extended Kalman Filter uses Jacobians G_k and H_k instead of the corresponding linear system matrices A_k, B_k and C_k in Kalman filter. The Extended Kalman filter algorithm is presented below:

Algorithm 2 Extended Kalman Filter

```
Inputs: $\mu_{k-1}$, $\Sigma_{k-1}$, $u_k$, $z_k$
Prediction
    $\overline{\mu}_k = g(\mu_{k-1}, u_k)$
    $\overline{\Sigma}_k = G_k \Sigma_{k-1} G_k^T + R_k$
Kalman Gain
    $K_k = \overline{\Sigma}_k H_k^T (H_k \overline{\Sigma}_k H_k^T + Q_k)^{-1}$
Correction
    $\mu_k = \overline{\mu}_k + K_k(z_k - h(\overline{\mu}_k))$
    $\Sigma_k = (I - K_k H_k)\overline{\Sigma}_k$
return $\mu_k$, $\Sigma_k$
```

The Extended Kalman filter has gotten just about the most popular tool for state estimation in robotics. Its strength lies in its simplicity and its computational efficiency that comes mainly from representing beliefs by a multivariate Gaussian distributions. A Gaussian is a unimodal distribution which can be though of as a single guess annotated with an uncertainty ellipse. Extended Kalman filters have been applied with great success to many state estimation problems. However, an important limitation of the Extended Kalman filter arises from the fact that it approximates state transitions and measurements using linear Taylor Expansions. The goodness of the linear approximation applied to Extended Kalman filters depends on the degree of uncertainty and the degree of the local non-linearity of the functions that are being approximated.

2.3.3 Extended Kalman Filter SLAM

The SLAM filtering solution, which is based on the Extended Kalman filter application is the first successfully implemented and most often used Online SLAM algorithm. As described earlier, the Extended Kalman filter is similar to the linear Kalman filter with the exception that the Extended Kalman filter uses a Gaussian approximate to the nonlinear state transition and measurement probabilities characterized by the mean μ_k and the covariance Σ_k. The amount if uncertainty in the posterior estimation must be relatively small, since otherwise the linearization in Extended Kalman filter tens to introduce intolerable errors.

In the Extended Kalman filter algorithm, the prediction consist of calculating the Jacobian matrices G_k and H_k of the state transition and measurement g and h respectively. Therefore the prediction step will have the following form:

$$\overline{bel}(x_k) = \begin{cases} \overline{\mu}_k = g(\mu_{k-1}, u_k) \\ \overline{\Sigma}_k = G_k \Sigma_{k-1} G_k^T + R_k \end{cases} \tag{2.23}$$

The update step uses the nonlinear observation function h alongside its Jacobian H_k. The Kalman gain is then calculated as:

$$K_k = \overline{\Sigma}_k H_k^T (H_k \overline{\Sigma}_k H_k^T + Q_k)^{-1} \tag{2.24}$$

Then, the posterior might be written in the following way:

$$bel(x_k) = \begin{cases} \mu_k = \overline{\mu}_k + K_k(z_k - h(\overline{\mu}_k)) \\ \Sigma_k = (I - K_k H_k)\overline{\Sigma}_k \end{cases} \tag{2.25}$$

Maps in EKF SLAM are feature-based, they consist of point landmarks. For computational reasons, the number of point landmarks is usually smaller than 1000. Further the Extended Kalman filter approach tends to work well in less ambiguous the landmarks are. For this reason, EKF SLAM requires significant engineering of feature detectors, sometimes using artificial beacons as features. [22]

Since SLAM process estimates the map alongside the pose, in EKF SLAM as the estimated value the state vector is used. Its two dimensional structure consisting of the pose x_k, the landmarks m_{N_x} and m_{N_y} and signature s_N. For robots that move in a plane, the mean vector μ_k is of dimension $3N + 3$, where N is the number of landmarks. Three dimensions are require to represent the pose of the robot, and two dimensions are required to specify the position of each landmark. Likewise, the covariance matrix is of size $3N + 3$ by $3N + 3$. Thus, the number of parameters needed to describe the EKF posterior is quadratic in the number of landmarks in the map:

$$Y_k = \begin{bmatrix} x_k & m \end{bmatrix}^T = \begin{bmatrix} x & y & \theta & m_{1,x} & m_{1,y} & s_1 & ... & m_{N,x} & m_{N,y} & s_N \end{bmatrix}^T \tag{2.26}$$

Thus the SLAM posterior has to be described as:

$$p(Y_k | z_{1:k}, u_{1:k}) \tag{2.27}$$

As stated in the previous paragraph, the mean and the covariance of the state vector will take the forms:

$$\mu_k = \begin{bmatrix} x_k \\ m \end{bmatrix} \tag{2.28}$$

and

$$\Sigma_k = \begin{bmatrix} \Sigma_{X_k X_k} & \Sigma_{X_k m} \\ \Sigma_{m X_k} & \Sigma_{mm} \end{bmatrix} = \begin{bmatrix} \Sigma_{X_k X_k} & \Sigma_{X_k m_1} & \dots & \Sigma_{X_k m_N} \\ \Sigma_{m_1 X_k} & \Sigma_{m_1 m_1} & \dots & \Sigma_{m_1 m_N} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{m_N X_k} & \Sigma_{m_N m_1} & \dots & \Sigma_{m_N m_N} \end{bmatrix} \tag{2.29}$$

At the initial state in the EKF-SLAM for the case with know-correspondence is taken to be the origin of the coordinate system. This definition is somewhat arbitrary, in that it can be replaced by any coordinate. None of the landmark locations are known initially. The mean and the covariance matrices will be filled by zero. Then, the computing is going through three steps similar to regular Kalman Filter: The current state estimate by control information, the estimated state update from observed landmarks, the new landmarks, add to the current state. It is repeating in a loop with returning μ_k and Σ_k for each time step calculation.

The figure [2.5] illustrates the EKF SLAM algorithm for an artificial example. The robot navigates from a start pose that serves as the origin of its coordinate system. As it moves, its own pose uncertainty increases, as indicated by uncertainty ellipses of growing diameter.

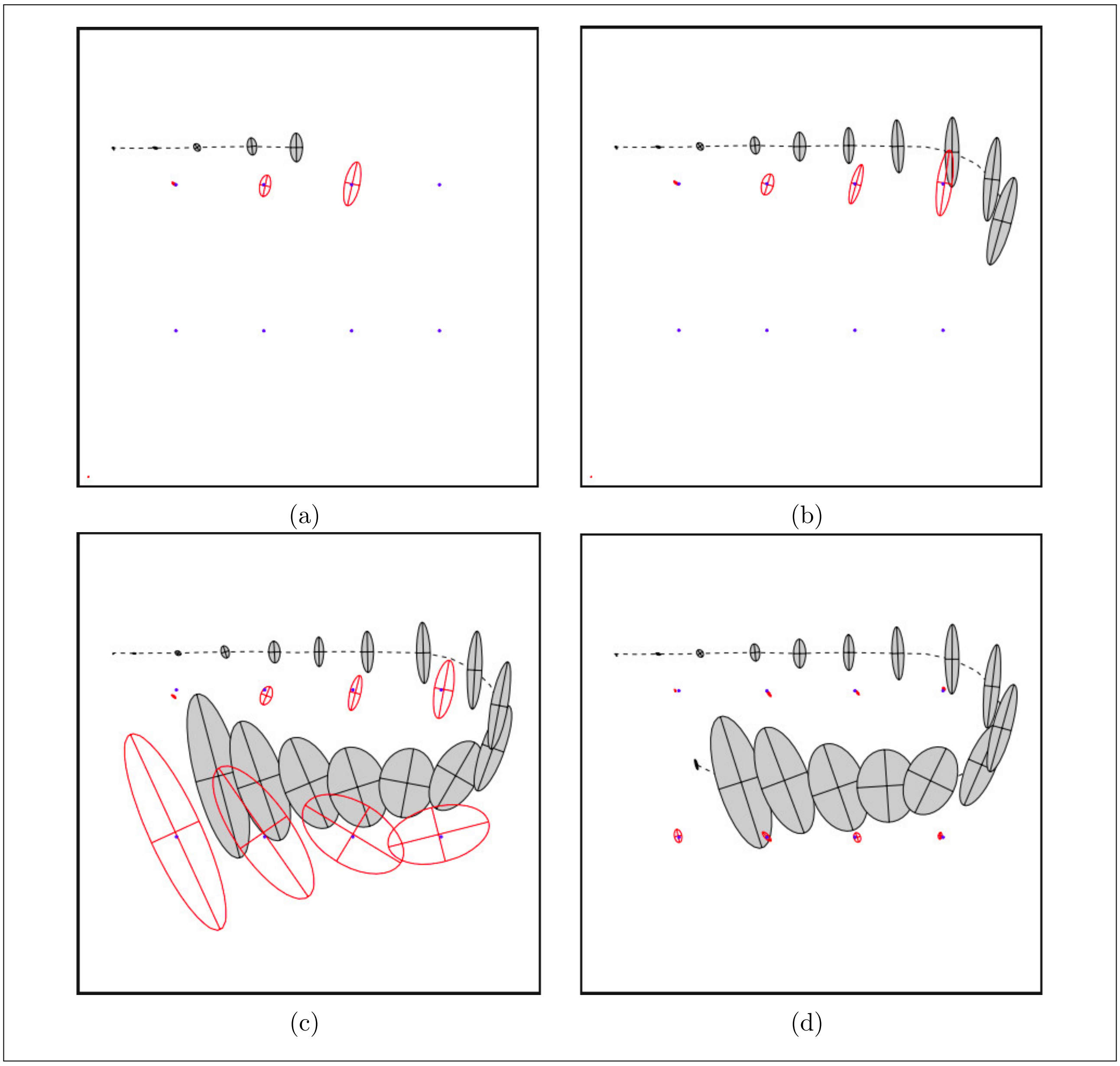


Figure 2.5: EKF applied to the online SLAM problem

It also senses nearby landmarks and maps them with an uncertainty that combines the fixed measurement uncertainty with the increasing pose uncertainty. As a result, the uncertainty in the landmark locations grows overtime. In fact, it parallels that of the pose uncertainty at the time a landmark is observed. The interesting transition happens in figure 2.5d: here the robot observes the landmark it saw in the very beginning of mapping, and whose location is relatively well known. Through this observation, the robot's pose error is reduced, as indicated by the very small error ellipse for the final robot pose. This observation also reduces the uncertainty for other landmarks in the map. [22]

2.4 Particle Filter

The Particle Filter is another alternative nonparametric implementation of the Bayes Filter [2.10]. It is mainly used for Online-SLAM problem approach. The key idea with Particle Filter is, instead of representing the posterior belief $bel(x_k)$ by a parametric form which would have been the Gaussian density of normal distribution, Particle Filter however approximate the same distribution by a set of random state samples called Particles hence the name Particle Filter (see figure [2.6]). This nonparametric approximation can represent a much broader space of the distributions than Gaussian normal density distribution. Another advantage of the sample based representation is the ability to model nonlinear transformations of random variables.

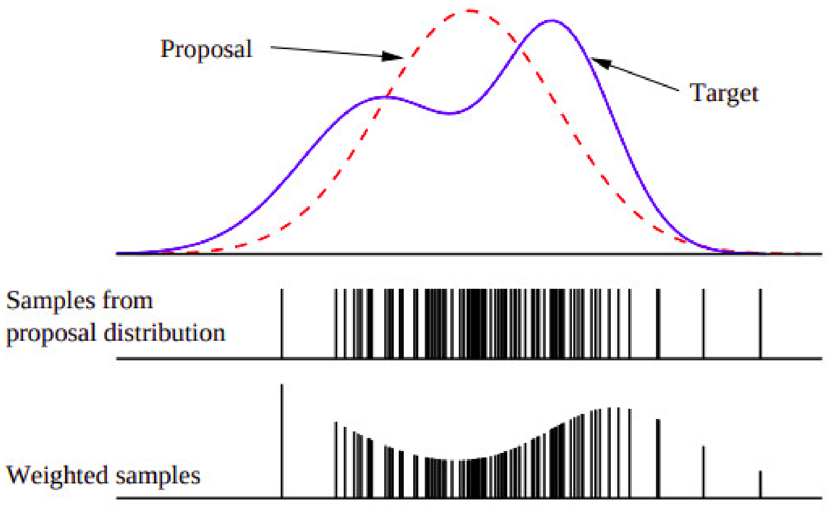


Figure 2.6: Particle samples for estimating multi-modal target distribution

The samples of a posterior distribution are denoted $\chi_k = \{x_k^{[1]}, x_k^{[2]}, ..., x_k^{[M]}\}$. Each particle $x_k^{[m]}$ with $1 \leq m \leq M$ is an instantiation of the state at time k. In another words, a particle is a hypothesis as to what the true world state may be at time k. Here M denotes the number of particles in the particle set χ_k. In practice, the number of particles M is big (e.g, $M \geq 1000$). Ideally, the likelihood for a state hypothesis x_k to be included in the particle set χ_k shall be proportional to its Bayes Filter posterior belief $bel(x_k)$. Therefore, the denser the subregion of the state space is populated by samples, the more likely it is that the true state falls into this region:

$$x_k^{[M]} \sim p(x_k|z_{1:k}, u_{1:k}) \tag{2.30}$$

Just like all other Bayes Filters discussed earlier in this chapter, the Particle Filter algorithm approximate the belief $bel(x_k)$ recursively from the belief $bel(x_{k-1})$ one time step earlier. Since beliefs are represented by sets of particles, this means that the Particle Filter construct the particle set χ_k recursively from the set χ_{k-1} [22]. The most basic variant of the Particle Filter algorithm is presented below:

Algorithm 3 Particle Filter Algorithm

```
particle set initialization
Inputs: $\chi_{k-1}$, $u_k$, $z_k$
Instantiation
for $m = 1$ to $M$ do
    sample $x_k^{[M]} \sim p(x_k|x_{k-1}^{[M]}, u_k)$
    $w_k^{[M]} = p(z_k|x_k^{[M]})$
    $\overline{\chi}_k = \overline{\chi}_k + \langle x_k^{[M]}, w_k^{[M]} \rangle$
end for
Importance Sampling
for $m = 1$ to $M$ do
    draw $i$ with probability $\propto w_k^{[i]}$
    add $x_k^{[i]}$ to $\chi_k$
end for
return $\chi_k$
```

The algorithm first construct the set of particles which likelihood is proportional to its Bayes Filter posterior belief $bel(x_k)$. Each particle or hypothetical state $x_k^{[m]}$ for time step k based the previous state $x_{k-1}^{[m]}$ and the control u_k is indexed by m indicating that it is generated from the m-th particle in χ_{k-1}. Moreover, for every particle $x_k^{[m]}$, the algorithm calculates the so-called the importance factor $w_k^{[m]}$. Importance factor or weight is used to incorporate the measurement z_k into the particle set. The importance, thus is the probability of the measurement z_k under the particle $x_k^{[m]}$ given by $w_k^{[m]} = p(z_k|x_k^{[m]})$. The set of weighted particles represents in approximation the Bayes Filter posterior $bel(x_k)$. Then the algorithm constructs a temporary particle set $\overline{\chi}$ that represented the belief $\overline{bel}(x_k)$. It does this by systematically processing each particle $x_{k-1}^{[m]}$ in the input particle set $\overline{\chi}_{k-1}$.

In the importance sampling step, the algorithm draws replacement M particles from the temporary set $\overline{bel}(x_k)$. The probability of drawing each particle is given by its importance weight. Resampling transforms a particle set of M particles into another particle set of the same size. By incorporating the importance wright into the resampling process, the distribution of the particles change. Whereas before the resampling step, they were distributed according to the $\overline{bel}(x_k)$, after the resampling they are distributed approximately according to the posterior $bel(x_k) = \eta p(z_k|x_k^{[m]})\overline{bel}(x_k)$. In fact, the resulting sample set usually possesses many duplicates, since particles are drawn with replacement. More important are the particles not contained in χ_k: Those tend to be the particles with lower importance weights. [22].

Such a Particle Filter algorithm would still approximate the posterior, but many of its particles would end up in regions of low posterior probability. As a result, it would require many more particles: how many depends on the shape of the posterior. The importance resampling step is a probabilistic implementation of the Darwinian idea. It refocuses the particle set to regions in state space with high posterior probability. By doing so, it focuses the computational resources of the filter algorithm to regions in the state space where they matter the most.

2.5 FastSLAM

As we have saw in the previous section, Particle Filter uses a set of particles also known as hypothesis to estimate the posterior belief $bel(x_k)$ rather than using a parametric Gaussian normal density distribution. Such representation has the advantage to describe a much broader space of the distribution than Gaussian density distribution, and it can also model nonlinear transformations of random variables. In research community, many attempts were made to test whether Particle Filters are applicable to the SLAM problem. Unfortunately, they are the subject to the curse of dimensionality and any implementation of the Particle Filters to the SLAM problem is doomed to fail, due to the large number of variables involved in describing the map.

An important characteristic of the full-SLAM problem with known correspondences is that any two disjoint set of features in the map are conditionally independence, given the robot pose. This structural characteristic will make it possible to apply a version of Particle Filters to represent the posterior over some variables, along with Gaussians or some other parametric probability distribution functions to represent all other variables.

FastSLAM takes advantage of this fundamental characteristic and uses Particle Filter for estimating the robot path. For each of these particles, the individual map errors are conditionally independent. Hence the mapping problem can be factored into many separate problems, one of each feature in the map. FastSLAM estimates these map feature locations by Extended Kalman Filter. This is fundamentally different from SLAM algorithms discussed earlier, which all use a single Gaussian to estimate the location of all features jointly.

$$\begin{aligned} p(Y_k|z_{1:k}, u_{1:k}) &= p(x_{1:k}, m_{1:M}|z_{1:k}, u_{1:k}) && (2.31)\\ &= p(x_{1:k}|z_{1:k}, u_{1:k})p(m_{1:M}|z_{1:k}, u_{1:k}) && (2.32)\\ &= \underbrace{p(x_{1:k}|z_{1:k}, u_{1:k})}_{\text{Particle Filter}} \prod_{i=1}^{M} \underbrace{p(m_i|z_{1:k}, u_{1:k})}_{\text{EKF}} && (2.33) \end{aligned}$$

with:

$$Y_k = \begin{bmatrix} x_k & m \end{bmatrix}^T \tag{2.34}$$

The key advantage of FastSLAM over other SLAM algorithms arises from the fact that Particle Filter can cope with nonlinear robot motion models, whereas previous techniques approximate such models via linear functions using the Taylor Expansion. Moreover, FastSLAM solves both the full-SLAM and the Online SLAM problem. It was conceived to calculate the full path posterior: only the full path renders feature locations conditionally independent. However, because Particle Filter estimates one pose at a time, FastSLAM is indeed an online SLAM. Hence it solves the online SLAM problem as well and it is the only algorithm that fits both categories.

The Basic Algorithm

Particles in the basic FastSLAM algorithm are of the form shown in the Table 2.1 below. Each particle contains an estimated robot pose, denoted $x_k^{[i]}$ and a set of Kalman filters with mean $\mu_{j,k}^{[i]}$ and covariance $\Sigma j, k^{[i]}$, one for each feature m_j in the map. Here i is the index of the particle $(1 \leq i \leq M)$.

	Robot path	feature 1	feature 2	...	feature N
Particle $i=1$	$x_{1:k}^{[1]} = \{(x\ y\ \theta)^T\}_{1:k}^{[1]}$	$\mu_1^{[1]},\Sigma_1^{[1]}$	$\mu_2^{[1]},\Sigma_2^{[1]}$	...	$\mu_N^{[1]},\Sigma_N^{[1]}$
Particle $i=2$	$x_{1:k}^{[2]} = \{(x\ y\ \theta)^T\}_{1:k}^{[2]}$	$\mu_1^{[2]},\Sigma_1^{[2]}$	$\mu_2^{[2]},\Sigma_2^{[2]}$	...	$\mu_N^{[2]},\Sigma_N^{[2]}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Particle $i=M$	$x_{1:k}^{[M]} = \{(x\ y\ \theta)^T\}_{1:k}^{[M]}$	$\mu_1^{[M]},\Sigma_1^{[M]}$	$\mu_2^{[M]},\Sigma_2^{[M]}$	...	$\mu_N^{[M]},\Sigma_N^{[M]}$

Table 2.1: Particles in FastSLAM are composed of a path estimate and a set of estimators of individual feature locations with associated covariances.

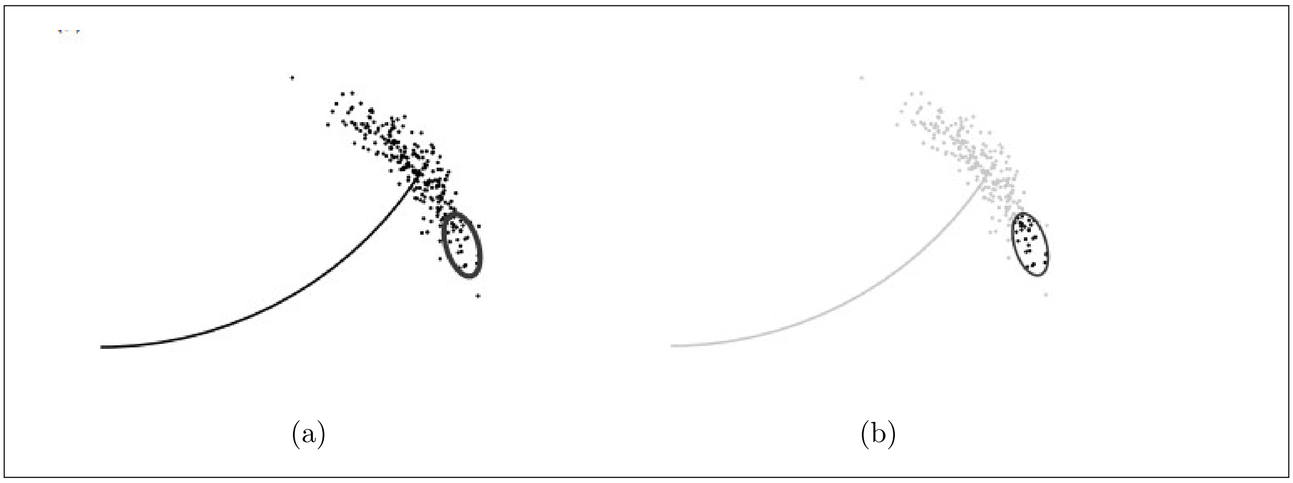


Figure 2.7: Mismatch between proposal and posterior distributions

The Particles in FastSLAM are used to compute the posterior over robot paths denoted by $p(x_{1:k}|z_{1:k}, u_{1:k})$. For each feature in the map, FastSLAM uses a separate estimator over its location $p(m_i|x_{1:k}, z_{1:k})$ one for each feature $i = 1, ..., N$. Thus, in total there are $N + 1$ posteriors in FastSLAM. The feature estimators are conditioned on the robot path, which means we will have a separate copy of each feature estimator, one for each particle. With M particles, the number of filters will actually be $1 + MN$. The product of these probabilities represents the desired posterior in a factored way. This factored representation is exact, not just an approximation. It is a generic characteristic of the SLAM problem. [22]

As noted, FastSLAM estimates the path posterior using a Particle Filter and the map feature locations are estimated using Extended Kalman filters. Because of our factorization, FastSLAM can maintain a separate Extended Kalman Filter for each feature which makes the update more efficient than in Extended Kalman Filter SLAM (EKF SLAM). Each individual Extended Kalman Filter is conditioned on a robot path. Hence, each particle possesses its own set of EKFs. In total there are NM EKFs, one for each feature in the map and one for each particle in the Particle Filter.

In the case of known data association, particles in FastSLAM are denoted:

$$Y_k^{[i]} = \langle x_k^{[i]}, \mu_{1,k}^{[i]}, \Sigma_{1,k}^{[i]}, ..., \mu_{N,k}^{[i]}, \Sigma_{N,k}^{[i]} \rangle \tag{2.35}$$

All these quantities are from the i-th particle $Y_k^{[i]}$, of which there are a total of M in the FastSLAM posterior.

Calculating the posterior at time step k from the one at time step $k-1$ involves generating a new particle set Y_k from Y_{k-1}. This new particle set incorporates a new control u_k and a measurement z_k. The update is performed in the following steps:

- **Extending the path posterior by sampling new poses**. FastSLAM uses the control u_k to sample new robot pose x_k for each particle in Y_{k-1}. The pose x_k is sampled in accordance with the i-particle according to the motion posterior:

$$x_k^{[i]} \sim p(x_k | x_{k-1}^{[i]}, u_k) \tag{2.36}$$

  The resulting sample is then added to a temporary set of particles, along with the path of previous poses $x_{1:k-1}^{[i]}$.

- **Updating the observed feature estimate** Next the FastSLAM updates the posterior over the feature estimates, represened by the mean $\mu_{j,k-1}^{[i]}$ and the covariance $\Sigma_{j,k-1}^{[i]}$. The updated values are then added to the temporary particle set along with the new pose. For the observed feature the update is done via the equation $p(m_j|x_{1:k}, z_{1:k}) = \eta p(z_k|x_k, m_j)p(m_j|x_{1:k-1}, z_{1:k-1})$. The new estimate at time step k is updated using the linearized measurement model $p(z_k|x_k, m_j)$ in the same way as EKF SLAM:

$$h(m_j, x_k^{[i]}) \approx h(\mu_{j,k-1}^{[i]}, x_k^{[i]}) + H_k^{[j]}(m_j - \mu_{j,k-1}^{[i]}) \tag{2.37}$$

$$= \hat{z}_k^{[i]} + H_k^{[j]}(m_j - \mu_{j,k-1}^{[i]}) \tag{2.38}$$

  The mean and the covariance of the posterior are obtained using the standard Extended Kalman Filter measurement update:

$$K_k^{[i]} = \Sigma_{j,k-1}^{[i]} H_k^{[j]T} (H_k^{[j]} \Sigma_{j,k-1}^{[i]} H_k^{[j]T} + Q_k)^{-1} \tag{2.39}$$

$$\mu_{j,k}^{[i]} = \mu_{j,k-1}^{[i]} + K_k^{[i]}(z_k - \hat{z}_k^{[i]}) \tag{2.40}$$

$$\Sigma_{j,k}^{[i]} = (I - K_k^{[i]} H_k^{[j]}) \Sigma_{j,k-1}^{[i]} \tag{2.41}$$

  The two previous steps are repeated M times, resulting in a temporary set of M particles.

- **Importance Resampling** FastSLAM resamples this set of particles. it draws from its temporary set M particles (with replacement) according to a the importance weight. The resulting set then forms the new final set Y_k. The necessity to resample arises from the fact that the particles in the temporary set are not distributed according to the desired posterior. The importance factor is given by:

$$w_k^{[k]} \approx \eta |2\pi Q_k^{[i]}|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2}(z_k - \hat{z}_k^{[i]})^T Q_k^{[i]-1} (z_k - \hat{z}_k^{[i]}) \right\} \tag{2.42}$$

  with the covariance: $Q_k^{[i]} = H_k^{[j]T} \Sigma_{j,k-1}^{[i]} H_k^{[j]} + Q_k$ Through the importance resampling process, particles survive in proportion of their measurement probability.

A summary of the FastSLAM algorithm with known data association is provided below:

2.6 Conclusion

The three paradigm just discussed cover the vast majority of work in the field of Simultaneous Localization and Mapping. Extended Kalman Filter based SLAM (EKF-SLAM) comes

Algorithm 4 FastSLAM Algorithm

```
Do the following M times:
    **Retrieval**: Retrieve a pose $x_{k-1}^{[i]}$ from the particle set $Y_{k-1}$
    **Prediction**: Sample a new pose $x_k^{[i]} \sim p(x_k|x_{k-1}^{[i]}, u_k)$
    **Measurement update**: For each observed feature $z_k^{[i]}$:
    Identify the correspondence $j$ for the measurement $z_k^{[i]}$.
    Incorporate the measurement $z_k^{[i]}$ into the corresponding EKF by updating the mean $\mu_{j,k-1}^{[i]}$ and covariance $\Sigma_{j,k-1}^{[i]}$.
    **Importance weight**: Calculate the importance weight $w^{[j]}$ for the new particle.
**Resampling**: Sample with replacement $M$ particles where each particle is sampled with probability proportional to $w^{[j]}$.
```

with a computational hurdle that poses serious scaling limitations. The most promising extension of EKF-SLAM are based on building local submaps. However, in many the resulting algorithms resemble the graph-based approach. [20]

In practice, EKF-SLAM has been applied with some success. When landmarks are sufficiently distinct, the approach approximates the posterior well. However, EKF-SLAM suffers from its enormous update complexity, and the limitation to sparse maps.

Particle Filter methods sidestep some of the issues arising from the natural inter-feature correlations in the map which plagued the EKF-SLAM. By sampling from robot poses, the individual landmarks in the map become independent, and hence decorrelated. As a result FastSLAM can represent the posterior by a sampled robot pose, and many local independent Gaussian for its landmarks. The particle representation of FastSLAM has a number of advantages. Computationally, FastSLAM can be used as a filter, and its update requires linear-logarithmic time where EKF needed quadratic time. Further, FastSLAM can sample over data association, which makes it a prime method for SLAM with unknow data association. On the negative side, the number of necessary particles can grow very large, especially for robots seeking to map multiple nested loops. [20]

CHAPTER

3 ROBOT MOTION

3.1 Introduction

In this chapter, we will discribe the two remaining components for implementing the filter algorithms described in the previous chapter: the motion and the measurement models. We will initially focus on the motion model that comprises the state transition probability $p(x_k|u_k, x_{k-1})$, which plays an essential role in the prediction step of the Bayes Filter. Later on, we will then describe the probabilistic model of sensor measurements $p(z_k|x_k)$, which is essential for the measurement update step. [22]

Our exposition focuses entirely on mobile robot kinematics for robots operating in planar environments. In this way, it is much more specific than most contemporary treatments of kinematics.[22] In theory, the goal of a proper probabilistic model may appear to accurately model the specific types of uncertainty that exist in robot actuation and perception. In practice, the exact shape of the model often seems to be less important than the fact that some provisions for uncertain outcomes are provided in the first place. In fact, many of the models that have proven most successful in practical application vastly overestimate the amount of uncertain. We will point out such findings when discussing actual implementation of probabilistic robotics algorithms. [22]

3.2 Velocity Motion Model

The velocity motion model assumes that we can control a robot through two velocities, a rotational and a translational velocity. Drive trains commonly controlled in this way include differential drives, Ackerman drives, and synchro-drives. Drive systems not covered by the model we are about to present are those without non-holonomic constraints, such as robots equipped with Mecanum wheels or legged robots.

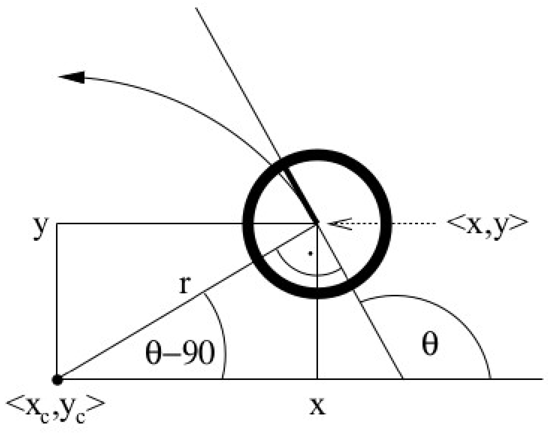


Figure 3.1: Motion carried out by a noise-free robot moving with constant velocities

We denote the translational velocity at time step k by v_k, and the rotational velocity by w_k. Hence, we have:

$$u_k = \begin{pmatrix} v_k \\ w_k \end{pmatrix} \tag{3.1}$$

We arbitrary postulate that positive rotational velocities w_k induce a counterclockwise rotation and positive translational velocities v_k correspond to forward motion.

Before turning to the probabilistic case, let us begin by stating the kinematics for an ideal, noise-free robot. Let $u_k = (v \ w)^T$ denote the control input at time step k. If both velocities are kept at a fixed value for the entire time interval $[k, k+1]$ the robot moves on a circle with radius $r = \left|\dfrac{v}{w}\right|$

Let $x_{k-1} = (x \ y \ \theta)^T$ be the initial pose of the robot, and suppose we keep the velocity constant at $(v \ w)^T$ for some time Δt. We can show with ease that the center of the circle is at :

$$x_c = x - \frac{v}{w}\sin(\theta) \tag{3.2}$$

$$y_c = y + \frac{v}{w}\cos(\theta) \tag{3.3}$$

After some time Δt of motion, our ideal robot will be at a new position $x_k = (x' \ y' \ \theta')^T$ with:

$$\begin{aligned}
\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} &= \begin{pmatrix} x_c + \frac{v}{w}\sin(\theta + w\Delta t) \\ y_c - \frac{v}{w}\cos(\theta + w\Delta t) \\ w\Delta t \end{pmatrix} && (3.4) \\
&= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{w}\sin(\theta) + \frac{v}{w}\sin(\theta + w\Delta t) \\ +\frac{v}{w}\cos(\theta) - \frac{v}{w}\cos(\theta + w\Delta t) \\ w\Delta t \end{pmatrix} && (3.5)
\end{aligned}$$

In reality, robot motion is subject to noise. The actual velocities differ from the commanded or measured ones if the robot possesses a sensor for measuring velocity. We will model this difference by a zero-centered random variable with finite variance. More precisely, let assume the actual velocities are given by:

$$\begin{pmatrix} \hat{v} \\ \hat{w} \end{pmatrix} = \begin{pmatrix} v + \epsilon_v \\ w + \epsilon_w \end{pmatrix} \tag{3.6}$$

Here ϵ is a zero-mean error variable with variance σ^2. Thus, the true velocity equals the commanded velocity plus some small additive error (noise). The most common choice for the error ϵ is the normal Gaussian distribution density given by the equation:

$$\epsilon_{\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}} \tag{3.7}$$

Therefore, a better model of the actual pose x_k after executing the motion command u_k with noisy motion at x_{k-1} is thus:

$$\begin{pmatrix} x^{'} \\ y^{'} \\ \theta^{'} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{w}} \sin(\theta) + \frac{\hat{v}}{\hat{w}} \sin(\theta + \hat{w}\Delta t) \\ +\frac{\hat{v}}{\hat{w}} \cos(\theta) - \frac{\hat{v}}{\hat{w}} \cos(\theta + \hat{w}\Delta t) \\ \hat{w}\Delta t \end{pmatrix} \tag{3.8}$$

Computation of $p(x_k|u_k, x_{k-1})$

Let us calculate the probability $p(x_k|u_k, x_{k-1})$of control action $u_k = (v\ w)^T$ carrying the robot from the pose $x_{k-1} = (x\ y\ \theta)^T$ to the new pose $x_k = (x^{'}\ y^{'}\ \theta^{'})^T$ within Δt time units. To do so, we will first determine the control $\hat{u} = (\hat{v}\ \hat{w})^T$ required to carry the robot from x_{k-1} to the position $(x^{'}\ y^{'})^T$ regardless of the robot's final orientation. Subsequently, we will determine the final rotation necessary for the robot to attain the orientation $\theta^{'}$. Based on these calculations, we can then calculate the desired probability $p(x_k|u_k, x_{k-1})$.

We already assumed that the robot travels with a fixed velocity during Δt, resulting in a circular trajectory. The center of the circle is defined as $(x^*\ y^*)^T$ and given by:

$$\begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} \frac{x + x^{'}}{2} + \mu(y - y^{'}) \\ \frac{y + y^{'}}{2} + \mu(x^{'} - x) \end{pmatrix} = \begin{pmatrix} \frac{x + x^{'}}{2} + \frac{1}{2}\frac{(x - x^{'})\cos\theta + (y - y^{'})\sin\theta}{(y - y^{'})\cos\theta + (x - x^{'})\sin\theta}(y - y^{'}) \\ \frac{y + y^{'}}{2} + \frac{1}{2}\frac{(x - x^{'})\cos\theta + (y - y^{'})\sin\theta}{(y - y^{'})\cos\theta + (x - x^{'})\sin\theta}(x^{'} - x) \end{pmatrix} \tag{3.9}$$

The radius of the circle is given by the Euclidean distance:

$$r^* = \sqrt{(x - x^*)^2 + (y - y^*)^2} \tag{3.10}$$

Furthermore, we can now calculate the change of the heading direction:

$$\Delta\theta = \arctan\Big(y^{'} - y^*, x^{'} - x^*\Big) - \arctan(y - y^*, x - x^*) \tag{3.11}$$

The motion model error is the deviation of $\hat{u}_k$ from the commanded velocity u_k as defined below:

$$v_{err} = v - \hat{v} \tag{3.12}$$
$$w_{err} = w - \hat{w} \tag{3.13}$$

Under our error model, the desired probability distribution $p(x_k|u_k, x_{k-1})$ is the product of the individual error probability distributions

$$p(x_k|u_k, x_{k-1}) = \epsilon(v_{err}) \times \epsilon(w_{err}) \tag{3.14}$$

3.3 Odometry Motion Model

The velocity motion model discussed earlier uses the robot velocity to compute posterior over poses. Alternatively, we might want to use the odometry measurements as the basis for calculating the robot's motion over time. Odometry is commonly obtained by integrating wheel encoder information.

Practical experience suggests that odometry, while still erroneous is usually more accurate than velocity. Both suffer from drift and slippage, but velocity additionally suffers from the mismatch between the actual motion controllers and its mathematical model. However, odometry is only available in retrospect, after the robot moved. This poses no problem for filter algorithms, such as localization and mapping algorithms discussed in the previous chapter. But it makes this information unusable for accurate motion planning and control.

The mathematical derivation of the algorithm is relatively straightforward. To derive a probabilistic motion model using odometry, we recall that the relative difference between any two poses is represented by a concatenation of three basic motions: a rotation, a straight-line motion (translation), and another rotation. The equations below show how to calculate the values of the two rotations and the translation from the odometry reading $u_k = (x_{k-1}\ x_k)^T$ with $x_{k-1} = (x\ y\ \theta)^T$ and $x_k = (x^{'}\ y^{'}\ \theta^{'})^T$:

$$\hat{\delta}_{rot1} = \arctan\Big(y^{'} - y, x^{'} - x\Big) - \theta \tag{3.15}$$

$$\hat{\delta}_{trans} = \sqrt{(x - x')^2 + (y - y')^2} \tag{3.16}$$

$$\hat{\delta}_{rot2} = \theta^{'} - \theta - \delta_{rot1} \tag{3.17}$$

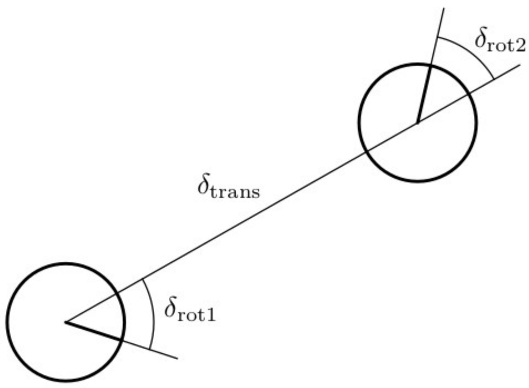


Figure 3.2: Odometry Motion Model

We assume that the true values of the rotation and translation are obtained from the measured ones by subtracting independent noise ϵ_{σ^2} with zero mean and variance σ^2.

Consequently, the true position x_k is obtained from x_{k-1} by an initial rotation with angle δ_{rot1}, followed by a translation with distance δ_{trans}, followed by another rotation with angle δ_{rot2}. Thus:

$$\begin{pmatrix} x^{'} \\ y^{'} \\ \theta^{'} \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \delta_{trans}\cos\theta + \delta_{rot1} \\ \delta_{trans}\sin\theta + \delta_{rot1} \\ \delta_{rot1} + \delta_{rot2} \end{pmatrix} \tag{3.18}$$

with $\delta = \hat{\delta} - \epsilon_{\sigma^2}$ is the true robot measurements and $\hat{\delta}$ is the noisy measurement collected from sensors.

The odometry motion model is obtained by computing the difference between the hypothesized pose x_k, relative to the initial pose x_{k-1}, assuming that x_k is the true final pose. The probability $p(x_k|u_k, x_{k-1})$ is computed by multiplying the probability distributions of the errors in odometry data:

$$p(x_k|u_k, x_{k-1}) = \epsilon(\delta_{rot1} - \hat{\delta}_{rot1}) \times \epsilon(\delta_{trans} - \hat{\delta}_{trans}) \times \epsilon(\delta_{rot2} - \hat{\delta}_{rot2}) \tag{3.19}$$

3.4 Measurement Model

Environment measurement models comprise the second domain-specific model in probabilistic robotics. Measurement models describe the formation process by which sensors measurements are generated in the physical world. Today's robots use a variety of different sensor modalities, such as tactile sensors, range sensors, or cameras. The specifics of the model depends on the sensor: Imaging sensors are best modeled by projective geometry, whereas sonar sensors are best modeled by describing the sound wave and its reflection on surfaces in the environments.

To illustrate the basic problem of mobile robots that use their sensors to perceive their environment, the figure [3.3] below shows a typical sonar range scan obtained in a corridor with a mobile robot equipped with a cyclic array of 24 ultrasound sensors. The distance measured by the individual sensors are depicted in light gray and the map of the environment is shown in black.

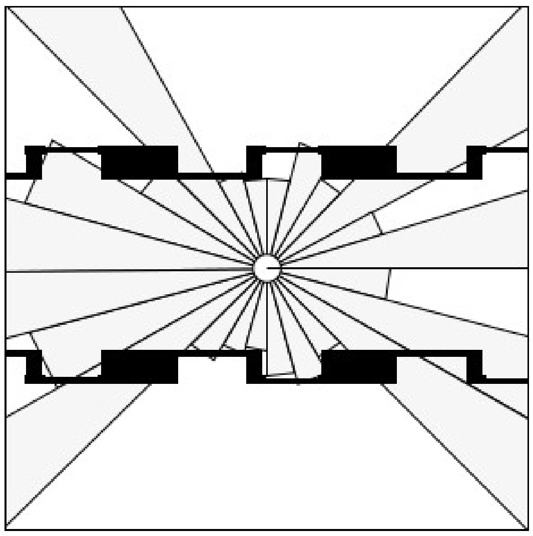


Figure 3.3: Typical ultrasound scan of a robot in its environment.

Many sensors generate more than one numerical measurement values when queried. For example, cameras generate entire arrays of values of brightness, saturation and color, similarly, range finders usually generate entire scans of ranges. We will denote the number of such measurement values within a measurement z_k by N, hence we can write:

$$z_k = \{z_k^1, z_k^2, ..., z_k^N\} \tag{3.20}$$

We will use z_k^n to refer to an individual measurement for example one range value. The probability $p(z_k|x_k,m)$ is obtained as the product of the individual measurement likelihoods [3.21]. Technically, this amounts to an Independence assumption between the noise in each individual measurement beam. This assumption is only true in the ideal case, however many dependencies exist due to a range of factors such as people who often corrupt measurements, error in the model, approximations in the posterior and so on.

$$p(z_k|x_k,m) = \prod_{n=1}^{N} p(z_k^n|x_k,m) \tag{3.21}$$

Beam Model of Range Finders

Range finders are among the most popular sensors used in modern robotics. This measurement model is therefore an approximative physical model of range finders. Range finders as the name suggest, measure the range to nearby objects and them often used along with a beam, which is a good model of the workings of laser range finders, which is the preferable model of ultrasonic sensors.

Our model incorporates four types of measurement errors, all of which are essential to making this model work: small measurement noise, errors due to unexpected objects, errors due to failures to detect objects, and random unexplained noise. The desired model probability $p(z_k|x_k,m)$ is therefore a mixture of four densities [3.4], each of which corresponds to a particular type of error:

- **Correct range with local measurement noise** In an ideal world, a range finder would always measure the correct range to the nearest object in its measurement field. We will use z_k^{n*} to denote the true range of the object measured by z_k^n. Even if the sensor correctly measures the range of the nearest object, the value it returns is subject to error. This error arises from the limited resolution of range sensors. This measurement noise is usually modeled by a narrow Gaussian p_{hit} with mean z_k^{n*} and standard deviation σ_{hit}:

$$p_{hit}(z_k^n|x_k,m) = \begin{cases} \eta\ \mathcal{N}(z_k^n, z_k^{n*}, \sigma_{hit}) & \text{if } 0 \le z_k^n \le z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3.22}$$

  where $\mathrm{z}_k^{n*} is calculated from \mathrm{x}_k$ and m via ray-casting, and z_{max} is the maximum sensor range, and η is a normalizing constant.

- **Unexpected Objects** Environments of mobile robots are dynamics, whereas maps m are statics. As a result, objects not contained in the map can cause range finders to produce surprisingly short ranges. A typical example of moving objects are people that share the operational space of the robot. One way to deal with such objects is to treat them as part of the state vector and estimate their location. Much simpler approach is to treat them as sensor noise which causes ranges to be shorter than z_k^{n*}. The likelihood of sensing unexpected objects decreases with range. Mathematically, the probability of range measurements are described by an exponential distribution. The parameter λ_{short} is an intrinsic parameter of the measurement model:

$$p_{short}(z_k^n|x_k,m) = \begin{cases} \eta\ \lambda_{short}\ e^{-\lambda_{short} z_k^n} & \text{if } 0 \le z_k^n \le z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3.23}$$

- **Failures** Sometimes, obstacles are missed altogether. For example, this happens frequently for sonar sensors as a result of specular reflections. Failure also occur with laser range finders when sensing black, light-absorbing objects. A typical result if a sensor failure is a max-range measurement: the sensor returns its maximum allowable value z_{max}.

  Wel will model this with a point-mass distribution centered at z_{max}:

  $$p_{max}(z_k^n|x_k, m) = \mathcal{I}(z = z_{max}) = \begin{cases} 1 & \text{if } z = z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3.24}$$

  where $\mathcal{I}$ denotes the indicator function that takes on the value 1 if its argument it true, and 0 otherwise.

- **Random Measurements**. Finally, range finders occasionally produce entirely unexplainable measurements. For example, sonars often generate phantom readings when they bounce off walls, or when they are subject to cross-talk between different sensors. We will model such measurements with a uniform distribution spread over the entire sensor measurement range:

  $$p_{rand}(z_k^n|x_k, m) = \begin{cases} \dfrac{1}{z_{max}} & \text{if } 0 \leq z_k^n \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{3.25}$$

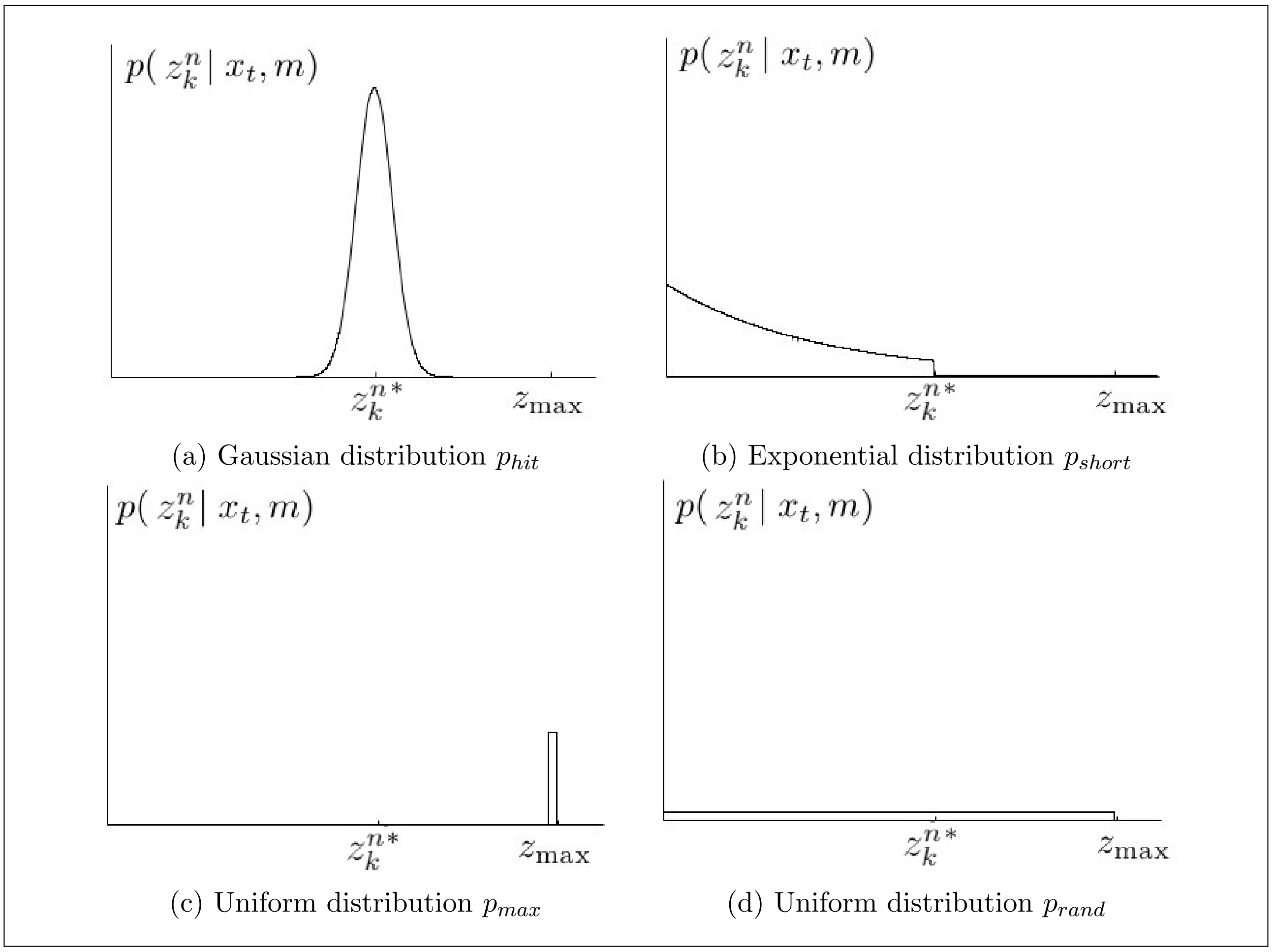
  These four different distributions are mixed by a weighted average defined by the parameters: z_{hit}, z_{short}, z_{max} and z_{rand} with $z_{hit} + z_{short} + z_{max} + z_{rand} = 1$:

  $$p(z_k^n|x_k, m) = \begin{pmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{pmatrix}^T \cdot \begin{pmatrix} p_{hit}(z_k^n|x_k, m) \\ p_{short}(z_k^n|x_k, m) \\ p_{max}(z_k^n|x_k, m) \\ p_{rand}(z_k^n|x_k, m) \end{pmatrix} \tag{3.26}$$

3.5 Conclusion

In this chapter, we derived two principal probabilistic motion models for mobile robots operating on the plane, as well as a probabilistic measurement model. We derived an algorithm for the probabilistic motion model $p(x_k|x_{k-1}, u_k)$ that represents control u_k by a translational and angular velocity, executed over a fixed time interval. In implementing this model, we realized the existing of two control parameters, one for translational and one for rotational velocities. Also, we presented an alternative motion model that uses the robot's odometry as input. Odometry measurements were expressed by three parameters, an initial rotation, followed by a translation, and a final rotation. The probabilistic motion model was implemented by assuming all three of these parameters are subject to noise.

Later on, we placed a strong emphasis on models for range finders, due to their great importance in robotics. However, the models discussed here are only representatives of a much broader class of probabilistic models. In choosing the right model, it is important to trade off physical realism with properties that might be desirable for an algorithm using these models. For example, we noted that a physically realistic model of range sensors may yield



(a) Gaussian distribution p_{hit} (b) Exponential distribution p_{short}

(c) Uniform distribution p_{max} (d) Uniform distribution p_{rand}

Figure 3.4: Components of the range finder sensor model

probabilities that are not smooth in the alleged robot pose which in turn causes problems for SLAM algorithms.

We started with models for range finders and lasers in particular, we discussed measurement models $p(z_k|x_k, m)$. We also devised a maximum likelihood technique for identifying the intrinsic noise parameters of the measurement model. Since the measurement model is a mixture model.

CHAPTER

4 TEST SIMULATION AND RESULTS

4.1 Introduction

The SLAM algorithms evaluated in this simulation are the ones we have talk about in the last chapter. The simulation consist of a moving planer robot with absolute measurements with different velocities: $v_1 = 0.9$ m/s, $v_2 = 0.7$ m/s and $v_3 = 0.5$ m/s. The landmarks featured in the map are motionless, and their coordinates are: $[-5\ 20, 3\ 15, 10\ 3, 15\ 10]$. For the real trajectory at a given position is having a moderate measurement noise as shown in the figure [4.1]. The state equation is a diagonal, which ensures that the next state's estimate or prediction is equal to the present state. The primary covariance matrix is well-defined by a higher diagonal uncertainty mutually in the position of the landmark and the robot state and by a comparable uncertainty, which means that none prevails over the other.

4.2 EKF-SLAM Simulation

The process noise matrix represented by Q and the measurement noise matrix represented by R are computed in which the landmarks are motionless. For the next state prediction, the measurement is done at the prediction position. The maximum range for the measurement sensor is set to be 20. The robot state denoted by $\boldsymbol{x}_k$ has 3 components: the planer position x, y, and orientation θ. The true state vector at any time step is given by $\boldsymbol{x}_k = [x_k\ y_k\ \theta_k]^T$. The robot has a speed sensor and a gyroscope, so the input vector used at each time step is $\boldsymbol{u}_k = [v_k\ \omega_k]^T$. Also, the robot has a GNSS sensor, which means that the robot can observe its x-y position at each time step: $\boldsymbol{z}_k = [x_k\ y_k]^T$.

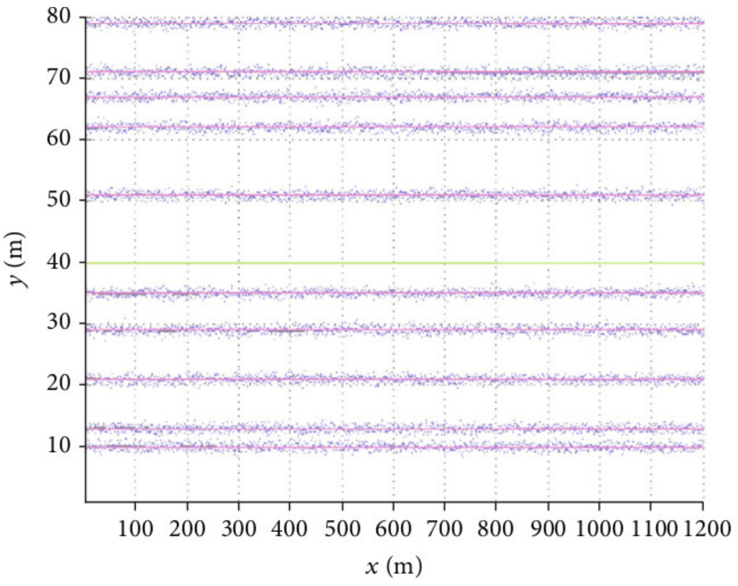


Figure 4.1: A moving robot with absolute measurements and a process noise

The robot motion model is given by the following equations:

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{4.1}$$

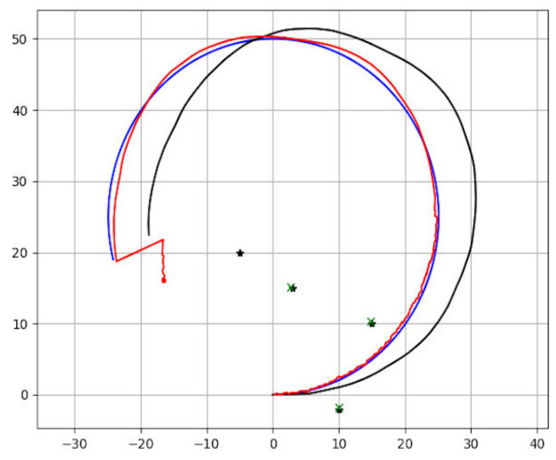
The motion model in state space representation is as follows:

$$\dot{\boldsymbol{x}}_{k+1} = \boldsymbol{f}(x_k, y_k, \theta_k) \tag{4.2}$$

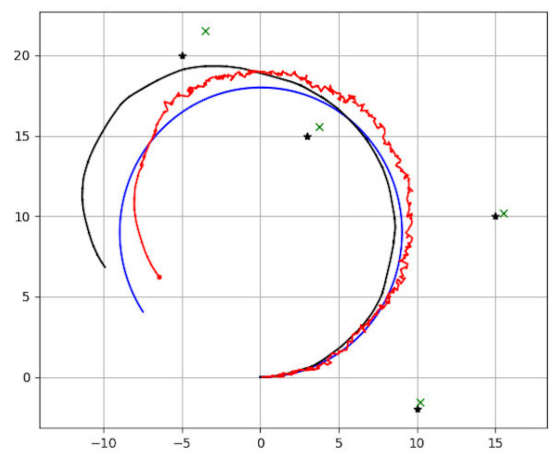
In order to predict and update the mean μ_k and the covariance Σ_k, the Extended Kalman Filter needs to compute the Jacobian G_k and H_k by replacing the system state $\boldsymbol{x}_k$ by the mean μ_k. Then the Kalman Gain is calculated and used in the correction step as described by the algorithm 2.

4.2.1 Interpretation

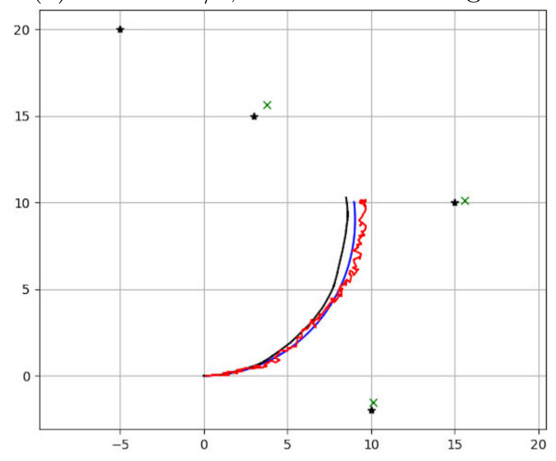
The obtained results with different velocities and measurements range are presented in the figure [4.2]. By varying the robot velocity, the robot is diverging from its route and therefore, the coverage area is reduced as seen in figure 4.2a, 4.2f. The EKF-SLAM in red is pretty accurate as estimating the true robot position in blue, however is the case of $v = 2.5$ m/s, the EKF-SLAM tripped at ones, but it quickly recovers. The error between the dead-reckoning in dark line and the true robot pose is intolerable. The estimate location of the landmarks featured in the map is surprisingly precise in the case of a maximum range of 20, however when it is set to 10, the error between the true location of landmarks and the estimated one is increasing 4.2b, 4.2c. Initially, due to the low range of the sensor, the robot did not estimate all the landmarks featured in the map 4.2a, 4.2c, 4.2e, but after sometime discovering the environment, all the landmarks were located with ranging error depending of on the maximum range of the sensor.



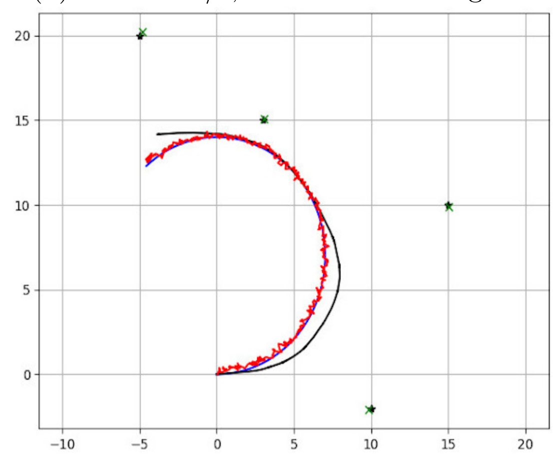
(a) $v = 2.5$ m/s, measurement range 20



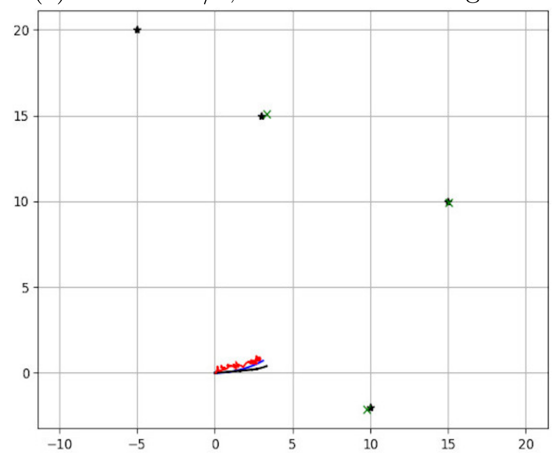
(b) $v = 0.9$ m/s, measurement range 20



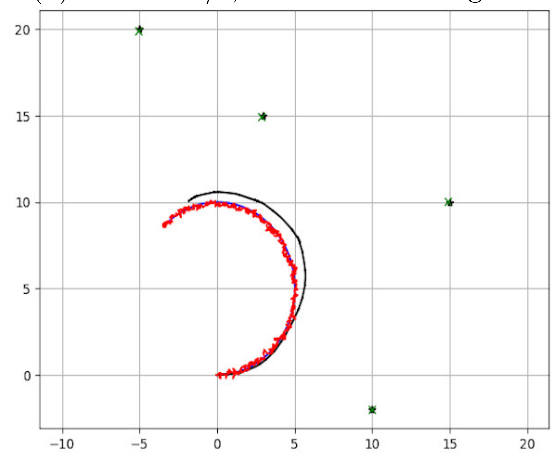
(c) $v = 0.9$ m/s, measurement range 10



(d) $v = 0.7$ m/s, measurement range 20



(e) $v = 0.7$ m/s, measurement range 10



(f) $v = 0.5$ m/s, measurement range 20

Figure 4.2: EKF-SLAM Simulation

4.3 Particle Filter SLAM Simulation

The Particle Filter present a big improvement over the standard Extended Kalman Filter for using a set of particles to estimate the robot's belief $bel(x_k)$ instead of using parametric representation consisting of Gaussian function density with mean μ_k and covariance Σ_k. Each particle in Particle Filter contains a hypothesis of the robot pose that assume its position is correct. Then, by using this assumption, every particle will maintain its own map.

There are several steps to implement particle filter algorithm. These steps are mentioned below:

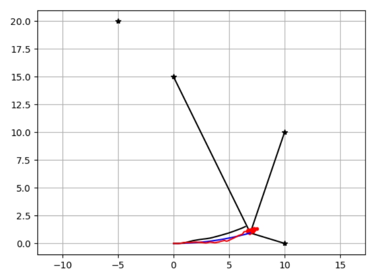
- Draw a distribution of even weighted particles.
- Update the robot's state x_k in each particle with control command u_k.
- Get observation data z_k from sensors.
- Update the robot's state estimate by incorporating the robot's observation.
- Calculate the importance weight w_n of each particle using the difference between actual observation and predicted observation.
- Resample particles proportional to their weights.

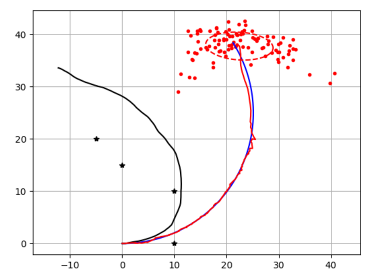
These steps are put into the prediction and correction steps of the Bayes Filter in order to implement the Particle Filter.

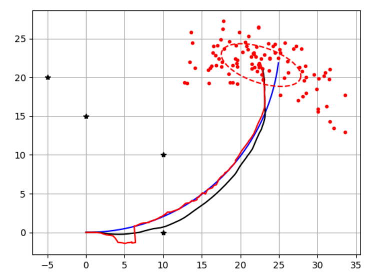
4.3.1 Interpretation

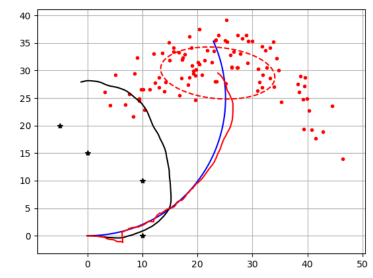
After implementing the Particle Filter algorithm described is the section [3], and varying the robot velocity ($v = 2.5$ m/s, $v = 0.9$ m/s and $v = 0.5$ m/s) as well as the maximum sensor range, we obtained the result presented in the figure 4.3. Using 100 particle. Initially, and due to the fact that all particles generated have the same importance weight, the error in the robot position estimate was pretty high for all cases, and also not all landmarks in the map where located, then the algorithm slowly converges toward the robot ground true marked in blue line in the case of $v = 0.9$ m/s for maximum range of 20 and 10, and $v = 0.5$ m/s for maximum range of 20 (figure 4.3e, 4.3f and 4.3g). When the robot moves with high (2.5 m/s) or low (0.5 m/s) speed, the robot starts to diverge from its route and the coverage area decreased 4.3b, 4.3c, 4.3d and 4.3i, which leads to an increasing error in estimation, and then divergence. The dead-reckoning curve in dark colour represent the pose estimate just by using the previous estimate and the odometry data. As seen from the figure 4.3 it is clearly bad !

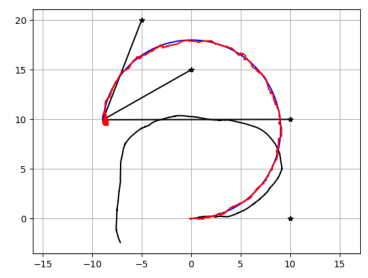
The algorithm showed promising results in some cases and terrible results in other cases. Some tweaks are needed via trial-and-error in order to obtain better results. Augmenting the number of particles at every resampling step or utilizing sensors with higher maximum range can potentially improve the robot position estimate and the landmarks location in the map.

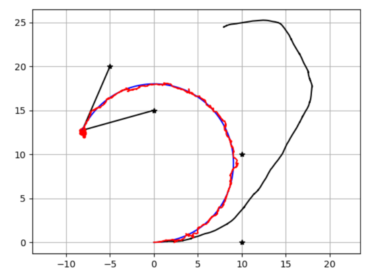

(a) v = 2.5 m/s, measurement range 20

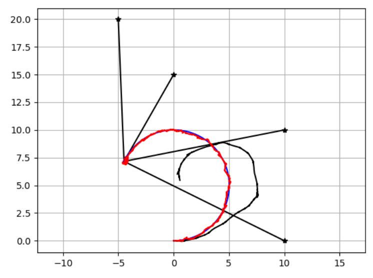

(b) v = 2.5 m/s, measurement range 20

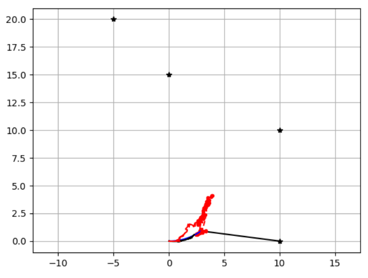

(c) v = 2.5 m/s, measurement range 10


(d) v = 2.5 m/s, measurement range 10


(e) v = 0.9 m/s, measurement range 20


(f) v = 0.9 m/s, measurement range 10


(g) v = 0.5 m/s, measurement range 20


(h) v = 0.5 m/s, measurement range 10

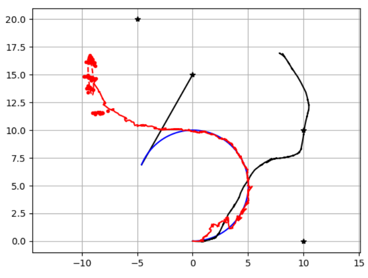

(i) v = 0.5 m/s, measurement range 10

Figure 4.3: Particle Filter SLAM applied to the moving planar robot with different velocities and measurement ranges

4.4 FastSLAM Simulation

As discussed in the section [2.5], the main advantage of the FastSLAM algorithm is the separation of the SLAM problem into a localization problem and landmark estimation problem. As a result, this algorithm can build very wide maps effectively and quicker than other existing SLAM algorithms such as Extended Kalman Filter SLAM and Particle Filter SLAM. The FastSLAM algorithm uses the Particle Filter to compute the posterior over the robot paths and each particle has k Kalman Filters that estimate the k-th landmark position conditioned on the robot path. Due to the large number of particles needed, the FastSLAM algorithm is very demanding in terms of computational power. However, some advance technique were implemented such as The sample impoverishment in the sample process and the Particle depletion in the importance resampling process which only incorporate the most recent measurement in the pose prediction.

The proposed algorithm for FastSLAM requires the following steps :

- **Step 1**: the robot path evaluation is implemented with a particle filter based on both the motion controls u_k and the sensor observations z_k. While landmark positions are estimated with k independent Kalman Filters for each particle.
- **Step 2**: Update the map: While the measurement corresponds to the observed landmark in the map previously, this landmark is updated using the standard Extended Kalman Filter update procedure.
- **Step 3**: Sample weight, The FastSLAM algorithm evaluates all particles based on their importance weight w_k to guarantee accuracy estimation.

4.4.1 Interpretation

While running the simulation, the results show a set of randomly localized, equally weighted particles trying to estimate the landmarks location in the map (figure 4.4a). As the robot moves through the environment collecting measurement data, the particles tend to have good estimates of the landmarks location, Thanks to Importance Resampling. This, however explains having very few particles compared to the initial situation, as particles are resampled in the positions of other particles having higher importance weight 4.4b, 4.4c. After sometime, the algorithm successfully located all the landmarks featured in the map as well as accurately estimating the robot position 4.4d.

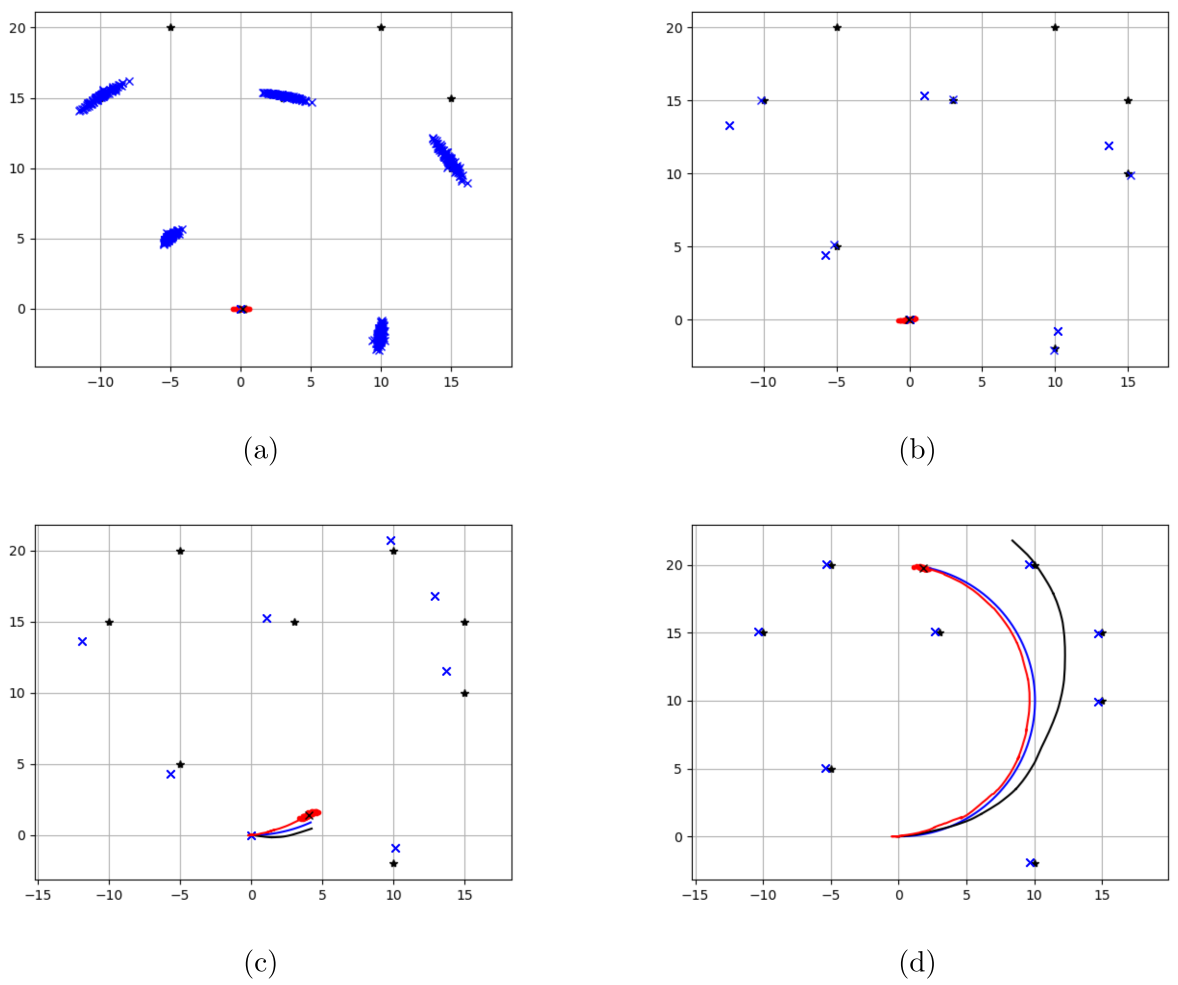


Figure 4.4: FastSLAM Simulation

4.5 Conclusion

In this experiment, we proposed three different SLAM algorithms for a moving planer robot with unknown location, EKF-based SLAM, Particle Filter and FastSLAM. The environment consist of motionless landmarks which positions are unknown to the robot. Firstly, the SLAM with EKF is implemented and an analytical expression for the the EKF-based SLAM algorithm is derived and their presentation is evaluated. The SLAM algorithm with EKF is evaluated in different scenarios, and several iterations were applied to explain the performance of EKF-based SLAM. The EKF-SLAM algorithm performance for varying velocities are presented in the figure [4.2]. Each process of localization is effective in its domain. In this analysis, many localization factors such as velocity, coverage area, localization time, maximum range of measurement are taken into consideration. The planned EKF-based SLAM algorithm present high precision and accurately estimated the robot position and landmark location for relatively low velocities. Particle Filter however, comes with computation cost and complexity due to the number of particles used (in our case 100 particle) and its non-parametric approach to the SLAM problem. The number of particles did affect the resulting map overtime, causing an increasing uncertainty caused by error accumulation from noise in sensor measurements. For relatively low or high velocities, the Particle Filter diverged and could not locate the robot.

Also, the particle depletion problem that causes the number of particles to decrease in each Importance Resampling step which affected noticeably the correction step.

FastSLAM however, solves the problem of computation by decomposing the SLAM problem into two parallel tasks, one for estimating the robot position by utilizing Particle Filter with reduced number of particles, and the other for the global mapping that utilizes EKF for each landmark featured in the map. The noticeable performance of the FastSLAM over the two other SLAM algorithms is the enhanced quality of estimation and the low latency for localization task.

CONCLUSION

The time has come to conclude our discussion of the SLAM problem and the promising solutions presented in this thesis. It should now be clear that there is no right answer to the SLAM problem. There are several ways of collecting data, and other ways of using the data after it is collected. Different environments lend themselves to different approaches, and each of the algorithms presented in this thesis have their place.

We began our discussion in the first chapter by establishing what the SLAM problem is on an intuitive level, and by discussing what a solution to the SLAM problem would accomplish. We first broke the SLAM problem down into the component problems (localization and mapping on their own), and we discovered each could be solved without too much difficulty. The problem is, in order to localize we need to start with an error free map, and to map we need to be able to localize. This is what motivated us to consider the first solution to the SLAM problem, the Kalman Filter.

In the Kalman Filter we found a solution that could take advantage of all available data statistically minimize the total squared error in our state prediction. The Kalman Filter accomplishes this task for us by maintaining a state prediction and a covariance matrix containing a confidence value for every binary relationship in the environment. We noticed the Kalman Filter specified how each observation could be used to update all feature prediction, and all covariance matrix values. Also, we noticed that the state prediction and covariance matrix had a fixed size independent of the number of iterations of the filter. Consequently, the Kalman Filter allows us to map for as long as we want without increasing the time to run each iteration. While the Kalman Filter is optimal in the sense that it yields a least squares prediction, it cannot always produce this prediction in real time. The problem is maintaining the covariance matrix which is quadratic in size with respect to the number of features. To cope with nonlinear problems, we extended the Kalman Filter. One technique described calculates a tangent to the nonlinear function. Tangents are linear, making the linear Kalman Filter applicable. The technique for finding the tangent is called the Taylor expansion. Performing this operation at a specific point yield to a matrix known as the Jacobian. The resulting Extended Kalman Filter is then used to calculate the correct posterior, assuming that the initial belief is Gaussian, the noise add to the state transition probability must also be Gaussian and the same applies to the measurement probability. A practical consideration when applying Extended Kalman Filter is to always keep the uncertainty of the state estimate as small as possible, otherwise, the robot will be less certain and the Gaussian belief will become wider because of the nonlinearities in the state transition and measurement functions.

The next type of filters presented in this thesis are the non-parametric Bayesian Filters such as the Particle Filter. Non-parametric filters approximate the posterior by a finite number of values under the assumption that both the system model and the shape of the posterior approximation errors converge uniformly to zero as the number of values used to represent the posterior goes to infinity. The Particle Filter represent the posterior by a random sample of states, drawn from the posterior. Such samples are called particles. We saw that the Particle filter is easy to implement and is the most versatile of all Bayes filters algorithms represented in this thesis.

FastSLAM approaches the SLAM problem from a Bayesian perspective and draws upon the insight that observations are conditionally independent of one another given the true robot pose. Errors in robot pose are accounted for by maintaining a set of poses represented by particles. FastSLAM also used simple Kalman Filter just like the one we saw, to update each feature prediction for each particle. This allows for an approximation of the optimal estimate while at the same time making it possible to take many more features into account. FastSLAM often outperforms Extended Kalman Filter despite being non-optimal because so many more features can be utilized in a given amount of time.

The simulation presented in the last chapter emphasizes the differences we already talked about in this thesis. By utilizing different scenarios for a moving non-holonomic wheelded robot, the experiment results were predictable. For higher velocities, the robot deviates from its route and the coverage area decreases, the pose estimate was terrible and the landmarks location was inaccurate. The Particle Filter needed some empirical tweaks in order to successfully estimate the robot pose and the landmarks featured in the environment. FastSLAM in the other hand performed very well at localizing the robot, but its computational constraint was noticeable.

BIBLIOGRAPHY

[1] Barraquand, J., and Latombe, J.-C. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research 10*, 6 (1991), 628–649.

[2] Brooks, R. A robust layered control system for a mobile robot. *IEEE journal on robotics and automation 2*, 1 (1986), 14–23.

[3] Canny, J., and Reif, J. New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)* (1987), IEEE, pp. 49–60.

[4] Cozman, F., and Krotkov, E. Automatic mountain detection and pose estimation for teleoperation of lunar rovers. In *Experimental Robotics V*. Springer, 1998, pp. 207–215.

[5] Dissanayake, M. G., Newman, P., Clark, S., Durrant-Whyte, H. F., and Csorba, M. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation 17*, 3 (2001), 229–241.

[6] Gauss, C. F. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*, vol. 7. FA Perthes, 1877.

[7] Golombek, M., Cook, R., Economou, T., Folkner, W., Haldemann, A., Kallemeyn, P., Knudsen, J. M., Manning, R., Moore, H., Parker, T., et al. Overview of the mars pathfinder mission and assessment of landing site predictions. *Science 278*, 5344 (1997), 1743–1748.

[8] Hammad, A., Ali, S. S., and Eldien, A. S. T. A novel implementation for fastslam 2.0 algorithm based on cloud robotics. In *2017 13th International Computer Engineering Conference (ICENCO)* (2017), IEEE, pp. 184–189.

[9] Hanebeck, U. D., and Schmidt, G. Set theoretic localization of fast mobile robots using an angle measurement technique. In *Proceedings of IEEE international conference on robotics and automation* (1996), vol. 2, IEEE, pp. 1387–1394.

[10] Huber, D. F., and Vandapel, N. Automatic 3d underground mine mapping. In *Field and Service Robotics* (2003), Springer, pp. 497–506.

[11] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation 12*, 4 (1996), 566–580.

[12] Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*. Springer, 1986, pp. 396–404.

[13] Kudriashov, A., Buratowski, T., Giergiel, M., and Małka, P. *SLAM Techniques Ap-*

plication for Mobile Robot in Rough Terrain. Springer, 2020.

[14] Kuipers, B., and Byun, Y.-T. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and autonomous systems 8*, 1-2 (1991), 47–63.

[15] Levitt, T. S., and Lawton, D. T. Qualitative navigation for mobile robots.

[16] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai 593598* (2002).

[17] Nieto, J., Guivant, J., Nebot, E., and Thrun, S. Real time data association for fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)* (2003), vol. 1, IEEE, pp. 412–418.

[18] Rimon, E. *Exact robot navigation using artificial potential functions.* PhD thesis, Yale University, 1990.

[19] Sakai, A., Ingram, D., Dinius, J., Chawla, K., Raffin, A., and Paques, A. Pythonrobotics: a python code collection of robotics algorithms. *CoRR abs/1808.10703* (2018).

[20] Thrun, S. Simultaneous localization and mapping. In *Robotics and cognitive approaches to spatial mapping.* Springer, 2007, pp. 13–41.

[21] Thrun, S., Burgard, W., and Fox, D. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots 5*, 3 (1998), 253–271.

[22] Thrun, S., Burgard, W., and Fox, D. Probalistic robotics. *Kybernetes* (2006).

[23] Thrun, S., Hahnel, D., Ferguson, D., Montemerlo, M., Triebel, R., Burgard, W., Baker, C., Omohundro, Z., Thayer, S., and Whittaker, W. A system for volumetric robotic mapping of abandoned mines. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)* (2003), vol. 3, IEEE, pp. 4270–4275.

[24] Wang, C.-C., Thorpe, C., and Thrun, S. Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)* (2003), vol. 1, IEEE, pp. 842–849.

[25] Yamauchi, B., Schultz, A., and Adams, W. Mobile robot exploration and map-building with continuous localization. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)* (1998), vol. 4, IEEE, pp. 3715–3720.

[26] Zhang, Z. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision 13*, 2 (1994), 119–152.

Abstract

In this work we provided a thorough overview of the rapidly growing topic of Simultaneous Localization and Mapping. SLAM is a subfield of Probabilistic Robotics concerned with localization, mapping, and motion control. A lot of robotic research goes into SLAM to develop self-driving cars, exploration robots, and disaster-relief robots. To begin with, we provided a mathematical description of the SLAM problem using Bayes recursive filtering techniques including linear Kalman Filter, Extended Kalman Filter, Particle Filter, and FastSLAM. We have particularly chosen these algorithms because they build off one another and they are used as a stepping stone on the way of the more state-of-the-art SLAM algorithms

Résumé

Dans ce travail, nous avons fourni un aperçu complet du sujet en pleine expansion de la localisation et de la cartographie simultanées (SLAM). Le SLAM est un sous-domaine de la robotique probabiliste qui s'intéresse à la localisation, à la cartographie et au contrôle du mouvement. De nombreuses recherches en robotique portent sur le SLAM afin de développer des voitures auto-conductrices, des robots d'exploration et des robots de secours en cas de catastrophe. Pour commencer, nous avons fourni une description mathématique du problème SLAM en utilisant des techniques de filtrage récursif de Bayes, notamment le filtre de Kalman linéaire, le filtre de Kalman étendu, le filtre à particules et FastSLAM. Nous avons particulièrement choisi ces algorithmes parce qu'ils s'appuient les uns sur les autres et qu'ils sont utilisés comme tremplin pour les algorithmes SLAM les plus avancés.

تلخيص :

في هذا العمل, قدمنا نظرة عامة شاملة على الموضوع سريع النمو المتمثل في التوطين المتزامن و رسم الخرائط SLAM .هو مجال فرعي من الروبوتات الاحتمالية يهتم بتحديد المواقع و رسم الخرائط و التحكم في الحركة . يذهب الكثير من الابحاث الروبوتية الى SLAM لتطوير سيارات ذاتية القيادة و روبوتات الإغاثة من الكوارث . في البداية قدمنا و صفًا رياضياتيكيا لمشكلة SLAM باستخدام تقنيات التصفية المتكررة Bayes في مرشح كالمان الخطي , مرشح كالمان الموسع , مشرح كالمان الموسع , مرشح الجسيمات و FastSLAM , لقد اخترنا هذه الخوارزمية بشكل متميز لأنها تبني بعضها البعض و تستخدم كحجر الأساس في طريق خوارزميات SLAM الاكثر حداثة .

Key Words

SLAM, Kalman Filter, Extended Kalman Filter, FastSLAM, Particle Filter, Python, Bayes Estimation.