Msc. Thesis SIANI

Test and Evaluation of the FastSLAM Algorithm in a Mobile Robot

❧ Evaluación y Prueba del Algoritmo FastSLAM sobre un Robot Móvil ❧

University of Las Palmas de Gran Canaria

University Institute of Sistemas Inteligentes y
Aplicaciones Numéricas en Ingeniería

Enrique Fernández Perdomo

A Thesis Submitted for the Degree of
MSc. Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería (SIANI)

October 16, 2009

To Sila and Iniesta

The fundamental cause of the trouble is that in the modern world the stupid are cocksure while the intelligent are full of doubt.

The Triumph of Stupidity
Bertrand Russell

Abstract

This work presents the test and assessment of the FastSLAM method, an algorithm to solve the SLAM problem. The SLAM (*Simultaneous Localization And Mapping*) problem is that of acquiring an enviroment map with a roving robot, while simultaneously localizing the robot relative to this map.

FastSLAM [93] is one of the most modern approaches to the SLAM problem that is based on particle filtering, showing several advantages over classical methods based on Kalman Filters (KF). Using a clever factorization of the SLAM problem, the complexity of the FastSLAM method is linear, or even logarithmic, with the number of features in the environment, where the complexity of KF-based methods is exponential. Moreover, the method is robust, since it can recover from wrong data associations, a problem that causes KF-based methods to diverge.

A testbed of different environments has been defined to evaluate the performance and results of FastSLAM. With a proper configuration [38], experiments have been done in both simulated and real environments using a mobile robot equipped with a range laser sensor. In both cases we analyze the applicability of the method to build sufficiently accurate maps in real time.

This document also provides a profuse literature review of SLAM methods and their ramifications in Robotics. We introduce the theoretical and mathematical foundations, to later describe contemporary approaches to solve the SLAM problem. The results and conclusions obtained are equivalent to those described by the authors of the method, hightlighting its applicability for real *office-like* environments operating in real time.

Resumen

En el presente trabajo se prueba y evalúa el método FastSLAM, un algoritmo para resolver el problema de SLAM. El problema de SLAM (*Simultaneous Localization And Mapping*) consiste en construir un mapa del entorno en el que se mueve un robot, localizando al mismo tiempo al robot dentro del mapa que se está construyendo.

FastSLAM es una de las últimas aportaciones dentro del ámbito de investigación de SLAM [93]. Haciendo uso de ciertas propiedades del problema, se consigue factorizarlo de modo que la complejidad del método es lineal, o incluso logarítmica, con respecto al número de características detectadas en el entorno, frente a la complejidad exponencial de otros métodos basados en filtros de Kalman. Además, el método es robusto, siendo capaz de recuperarse frente a asociaciones de datos incorrectas, un hecho que provoca la divergencia de los métodos basados en filtros de Kalman.

Se han definido varios entornos de prueba con la finalidad de evaluar el funcionamiento y los resultados del algoritmo FastSLAM. Partiendo de una configuración adecuada del algoritmo [38], se han realizado experimentos tanto simulados como en entornos reales con un robot móvil equipado con un sensor de rango láser. En ambos casos se analiza la aplicabilidad del método para construir mapas suficientemente precisos en tiempo real.

Este documento también aporta una profusa revisión bibliográfica de los métodos de SLAM y sus ramificaciones dentro del campo de la Robótica. Se introducen los fundamentos teóricos y la base matemática, para posterioremente describir las técnicas más modernas empleadas para resolver el problema de SLAM. Los resultados y conclusiones obtenidos son equivalentes a los expuestos por los autores del método, pudiendo constatarse la aplicabilidad del método en entornos de oficina reales operando en tiempo real.

Brief Contents

Contents

Acknowledgements

First and foremost, I want to thank the tutors and the members of the Robotics Research Group at SIANI, Antonio, Daniel, Jorge and Josep, for their supervision.

This work would have not been possible without the previous work of other mates, specially Julio, who studied most of the SLAM insights upon which this work is built.

I am also very greatful to the University Institute SIANI for the great effort to carry out the Msc. SIANI, and for the financial aid I was given. Thanks as well to the ULPGC for the postdegree scholarship they award me.

This work has also been partially supported by the research projects *PI2007/039* funded by the *Consejería de Educación, Cultura y Deportes*, *Gobierno de Canarias*, with FEDER funding, and *TIN2008-06068* funded by the *Ministerio de Ciencia e Investigación*, *Gobierno de España*.

Finally, I would like to thank my family and friends for their support and everything else, although some of them are scattered around the world. ✍

Notation

$x_t = (x\ y\ \theta)^T$	pose of the robot with location $(x\ y)^T$ and orientation θ at time t
$u_t = (v\ \omega)^T$	robot control with translational v and rotational ω velocities comanded
z_t	sensor observation at time t
$\mathcal{Y}_t$	FastSLAM particle set at time t
$\mathcal{Y}_t^{[k]}$	k-th FastSLAM particle at time t
M	number of particles
$N_t^{[k]}$	number of landmarks detected up to time t of the k-th particle
$\mu_{j,t}^{[k]}, \Sigma_{j,t}^{[k]}$	j-th landmark EKF (mean, covariance) of the k-th particle
$i_{j,t}^{[k]}$	j-th landmark counter (times seen) of the k-th particle
$w_t^{[k]}$	importance weight of the k-th particle
$\hat{c}$	index of Maximum Likelihood landmark
$z_t - \hat{z}_j$	measurement innovation of j-th landmark
$h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$	measurement model
$h'(\mu_{j,t-1}^{[k]}, x_t^{[k]})$	Jacobian of measurement model
$h^{-1}(z_t, x_t^{[k]})$	inverse measurement model
$g(x_{t-1}^{[k]}, u_t)$	motion model
$x_t[k] \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$	sampling from motion model probability distribution
Q_t	linearized measurement noise
R_t	linearized motion noise

Chapter 1

Introduction

The problem of Simultaneous Localization and Mapping (SLAM) has attracted immense attention in the Robotics literature. SLAM addresses the problem of a mobile robot moving through an environment of which no map is available *a priori*. The robot makes relative observations of its ego-motion and of objects in the environment, both corrupted by noise. The goal of SLAM is to reconstruct a map of the world and the path taken by the robot. SLAM is considered by many to be a key prerequisite to truly autonomous robots [93].

The estimation of the robot path is a straightforward localization problem if the true map of the environment is available. Similarly, mapping is relatively simple if the robot path is known. When both the robot path and the map are unknown, localization and mapping must be considered concurrenty. Hence the problem is coined *Simultaneous Localization and Mapping* (SLAM), and less frequently *Concurrent Mapping and Localization* (CML).

1.1 The SLAM Problem

First and foremost, *robot environments* are inherently unpredictable. While the degree of uncertainty in well-structured environments such as assembly lines is small, environments such as highways and private homes are highly dynamic and in many ways highly unpredictable. Furthermore, sensors are limited in what they can perceive. Limitations arise from several factors, e.g. the range and resolution of a sensor is subject to physical limitations. The uncertainty of the information obtained from the measurement sensor and the robot odometry is the main factor that complicate the SLAM process. For this reason, most SLAM techniques are probabilistic.

The *chicken-or-egg* relationship between localization and mapping is a consequence of how errors in the robot's sensor readings are corrupted by errors in the robot's motion. Since the robot pose estimate is corrupted by motion noise, the perceived locations of objects in the environment ar, in turn, corrupted by both measurement noise and the erro in the estimated pose of the robot. Montemerlo states it plainly in [93]: *error in the robot's path correlates errors in the map*. Consequently, the true map cannot be estimated accurately without also estimating the true path of the robot. The relationship between localization and mapping was first identified by Smith and Cheeseman [134] in 1987.

If robots are to operate autonomously extreme environments underwater, underground or on the surface of other planets, they must be capable of building maps and navigating reliably according to these maps. Unfortunately, the relationship between robot path error and map error does make the SLAM problem harder to solve in principle. Clearly, as the robot pose becomes more uncertain, the uncertainty in the estimated positions of observed landmarks also increases. Motion uncertainty is due to typical odometry noise, which is the consequence of wheel imperfections, drifting, slipping, etc. Whereas range laser are very precise but they perform poor with reflective and transparent surfaces, such as mirrors and glasses.

The ability to transform raw measurements into high level semantic concepts is still an open problem. Fortunately, it is possible to extract simple geometric elements, such as points and lines. These features cannot represent all types of environments though. They are suitable for *office* environments, but they are inappropiate for forests or the land. The type of landmarks to make a map of, is a topic that has attracked the interest of the research community [26, 27, 30].

The features detected in the environment with the measurement sensors must be compared with the landmarks in the map that is being built. This problem is known as *data association* and it is probably the most complex problem in SLAM. Observed features must be transformed form the robot frame to the global frame of the map. The uncertainty in the robot pose may cause incorrect data associations. Measurement information is also used to estimate the robot pose, which in sum may lead to divergence in the estimation of both the robot path and the location of landmarks in the map.

1.1.1 The FastSLAM algorithm

There exist a number of approaches to solve the SLAM problem. In the present work we will focus in a family of algorithms known as FastSLAM [93]. FastSLAM exploits the relationship between the robot path error and map error to factor the SLAM problem into a set of much smaller problems that can be solved efficiently. Especially, it uses $N+1$ estimators, one estimator over robot paths and N independent estimators over landmark positions, each conditioned on the path estimate. The algorithm uses a particle filter ot approximated the factored SLAM problem with M particles, each of them carrying a robot path hypothesis, along with its map.

Put in brief, the first step of the FastSLAM is to propose a new robot pose x_t for each particle that is consistent with the previous pose x_{t-1} and the new control u_t. Next, a landmark filter in each particle that corresponds with the latest observation is updated. Each particle is given an importance weight, and a new set of samples is drawn according to these weights. Such importance resampling step corrects for the fact that it selects particles with *good* estimations and discard the rest. As a result, the algorithm converges asymptotically to the true robot pose and map, while it still manage the uncertainty in both.

There exist two versions of FastSLAM. In the present work we focus on the first version, coined FastSLAM 1.0. Meanwhile, FastSLAM 2.0 is a modified version that corrects some of the problems of the former. Both versions scale efficiently to large maps and is robust to significant ambiguity in data association. However, these and other state-of-the-art SLAM algorithms assume a static environment. Unfortunately, the world is highly dynamic because of non-structural elements such as people or moving objects, and structural elements such as doors or any furniture

that may change its position. SLAM research in dynamic environment is still an open problem.

1.2 Thesis Statement

This dissertation involves the following thesis:

> It is possible to map structured indoor environments by means of the detection of basic geometric features, such as lines and corner points, using a robot equiped with a range laser sensor. Both the *full* and *online* SLAM problems can be solved with the family of FastSLAM algorithms. They make real time mapping feasible with sufficient accuracy in fairly large environments and abundance of landmarks.

The field of SLAM is very extense, so we have to constraint it under some assumptions to make it tractable. More details will be given in the description of the experiments performed, in Section 11.2.

Motion We will use a robotic differential drive platform that moves in a plane. The motion is denoted by the translational v and rotational ω velocities, which can be infered using the odometry.

Perception The robot will obtain measurements of the environment using a range laser sensor. It provides precise information that suffices to detected features in structured indoor environments.

Environment The experiments will take place in structured indoor environments that can be described with basic geometric features. In particular, we will only manage lines and corner points between them. We assume the environment is static, *not* dynamic.

Landmark As mentioned above, the landmarks considered to construct the map of the environment are lines and corner points. Such features are easily extracted from range laser scan with state of the art feature extraction methods.

Data It is possible to test the SLAM algorithm with real and simulated data. We consider both in *online* and *offline*, i.e. for real time or batch mode using data logs, respectively. Furthermore, the environment might be real or synthetic, if they are designed *ad hoc* for the quantitative evaluation of the algorithm.

1.3 Thesis Outline

This thesis will present the SLAM problem and a detailed description of FastSLAM algorithms. The document provides an unified review of a number of topics related with SLAM research. Recursive state estimators are described as they are the mathematical foundation of most SLAM algorithms and particularly of FastSLAM. The FastSLAM algorithm presented is based on features extracted from range laser scans, thus an important part of this thesis is concerned with feature extraction methods. An implementation of the FastSLAM 1.0 algorithm is evaluated with a number of experiments and metrics, leading to conclusions equivalent to those mentioned in the literature. The text is organized in five major parts:

- Chapters 2 through 4 introduce the basic mathematical framework for recursive state estimation. After a general introduction to Bayes filtering, most common state estimators are described, both parametric and nonparametric. They are the basis of the FastSLAM family of algorithms. These chapters are the mathematical foundation of the SLAM algorithms described throughout this thesis.

- Chapters 5 and 6 present probabilistic models of mobile robots as described in [149]. They cover the motion and measurement models, which are essential components of filter algorithms. In many ways, these chapters are the probabilistic generalization of classical robotic models and they form the robotic foundation for the material that follows.

- The basic feature extraction methods for range laser scans aimed to detect both lines and corner points are analyzed in Chapter 7. Next, Chapter 8 discusses the data association problem, which is a crucial step in the process of incorporating new features into the map from robot observations.

- Chapters 9 and 10 introduce the mapping field in Robotics and discuss some of the classical solutions. Chapter 9 describes the SLAM problem with the perspective of the joint estimation of the map and the path of the robot. Although some classical SLAM algorithms will be presented, the thesis will focus in a particular family of algorithms, the FastSLAM algorithms, discussed in detail throughout Chapter 10.

- Chapter 11 enumerates the experiments and metrics applied to evaluate the FastSLAM algorithm. A thoroughly evaluation yields some results and conclusions already observed in other works, which are discussed in Chapter 12. Similarly, Chapter 13 discusses some of the possible improvements and current fields open to research.

Most of the first chapters of this document supply the mathematical foundations and theorical concepts that constitute the basis of the SLAM algorithms later discussed. Each chapter is self-explantory, but it might employ concepts introduced in previous chapters. For this reason, the thesis is best read in order. Depending on the reader background some chapters, enumerated in the sequel, are not compulsory notwithstanding. The dissertation includes the mathematical derivation of some topics. Sections with that name might be skipped on first reading without compromising the coherence of the overall material.

The introduction to recursive state estimation may be skipped if the reader has a good probabilistic and filtering background. The same applies for the discussion about parametric and nonparametric filters, which are focused on Kalman and Particle filters, respectively. Contrary, robot motion and measurement models are encourage to be read because a probabilistic generalization of classical models is devised.

Although SLAM dissertations are rarely concerned with feature extraction, we present state-of-the-art feature extractors for laser scans aimed to detect lines and corner points. This chapter tries to summarize and gather most common algorithms, that the reader might already known. The problem of pairing observations and landmarks, known as data association problem, introduces the basics to understand one of the main problems concerning SLAM, so only experts on the subject may skip this chapter.

Therefore, the expert reader may go directly to the chapters about mapping and SLAM. Even more, those who are familiar with SLAM might start directly with the FastSLAM chapter, but this is generally not recommended.

Chapter 2

Recursive State Estimation

2.1 Introduction

At the core of probabilistic robotics is the idea of estimating state from sensor data. Stata estimation addresses the problem of estimating quantities from sensor data that are not directly observable, but that can be inferred. In most robotic applications, determining what to do is relatively easy if one only knew *certain* quantities. For example, moving a mobile robot is relatively easy if the exact location of the robot and all nearby obstacles are known. Unfortunately, these variables are not directly measurable. Instead, a robot has to rely on its sensors to gather this information. Sensors carry only partial information about those quantities, and their measurements are corrupted by noise. State estimation seeks to recover state variables from the data. Probabilistic state estimation algorithms compute belief distributions over possible world states. An example of probabilistic state estimation is mobile robot localization.

The goal of this chapter is to introduce the basic concepts and mathematical tools for estimating state from sensor data.

- Section 2.2 introduces basic probabilistic concepts used throughout the thesis.
- Section 2.3 describes our formal model of robot environment interaction, setting forth some of the key terminology used throughout the thesis.
- Section 2.4 introduces *Bayes filters*, the recursive algorithm for state estimation that forms the basis of virtually every technique presented in this thesis.

2.2 Probability

The theory presented in this chapter summarizes the basic notation and probabilistic facts used in this thesis, as appears in [149]. In probabilistic robotics, quantities such as sensor measurements, controls, robot pose and even the environment are modeled as random variables. A *random variable* can take on multiple values, and it does so according to specific probabilistic laws. Let X denote a random variable and x a specific value that X might assume. If the space of all values that X can take on is discrete, the probability that the random variable X has value x is

RANDOM VARIABLE

denoted with

$$p(X = x) \tag{2.1}$$

Discrete probabilities sum to one

$$\sum_x p(X = x) = 1 \tag{2.2}$$

and are always non-negative, i.e. $p(X = x) \geq 0$. To simplify the notation we will use the common abbreviation $p(x)$ instead of writing $p(X = x)$.

Most mapping techniques address estimation in continuous spaces, which are characterized by random variables that can take on a continuum of values. We assume that all continuous random variables possess a *probability density function* (PDF), unless explicitly stated. A common density function is that of the one-dimensional *normal distribution* with mean μ and variance σ^2. The PDF of a normal distribution is given by the following *Gaussian* function

PROBABILITY DENSITY FUNCTION
NORMAL DISTRIBUTION
GAUSSIAN

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \tag{2.3}$$

$$= \left(2\pi\sigma^2\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \tag{2.4}$$

We will frequently abbreviate them as $\mathcal{N}(x; \mu, \sigma^2)$, which specifies the random varible x, its mean μ and variance σ^2.

The normal distribution in (2.3) assumes that x is scalar. Often, x will be a multi-dimensional vector. Normal distributions over vectors are called *multivariate*. Multivariate normal distributions are characterized by density functions of the form

MULTIVARIATE

$$p(x) = \left|2\pi\Sigma\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{2.5}$$

Here μ is the mean vector and Σ is a *positive semidefinite* and *symmetric* matrix known as the *covariance matrix* —its determinant is therefore non-negative, i.e. $\left|\Sigma\right| \geq 0$.

COVARIANCE MATRIX

Just as discrete probability distributions always sum up to 1, a PDF always integrates to 1, that is

$$\int p(x)\, dx = 1 \tag{2.6}$$

Throughout this document we will use the terms *probability*, *probability density* and *probability density function* interchangeably. We will assume that all continuous random variables are measurable and that all continuous distributions possess densities.

The *joint distribution* of two random variables X and Y is given by

JOINT DISTRIBUTION

$$p(x, y) = p(X = x \text{ and } Y = y) \tag{2.7}$$

If X and Y are *independent*, we have

INDEPENDENT

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

Sometimes, random variables carry information about other random variables. If we already know that Y's value is y and we would like to know the probability that X's value is x conditioned on that fact, such probability is denoted

$$p(x \mid y) = p(X = x \mid Y = y) \tag{2.9}$$

and is called *conditional probability*. If $p(y) > 0$, then the conditional probability is defined as

CONDITIONAL PROBABILITY

$$p(x \mid y) = \frac{p(x)\, p(y)}{p(y)} \tag{2.10}$$

Actually, if X and Y are independent, we have

$$p(x \mid y) = \frac{p(x)\, p(y)}{p(y)} = p(x) \tag{2.11}$$

In such a case, Y tells us nothing about he value of X. Independence, and its generalization known as conditional independence, plays a major role in the SLAM algorithm discussed in the present work.

An interesting fact, which follows from the definition of conditional probability and the axioms of probability measures, is often referred to as *Theorem of total probability*

THEOREM OF TOTAL PROBABILITY

$$p(x) = \sum_y p(x \mid y)\, p(y) \quad \text{(discrete)} \tag{2.12}$$

$$p(x) = \int p(x \mid y)\, p(y)\, dy \quad \text{(continuous)} \tag{2.13}$$

Equally important is *Bayes rule*, which relates a conditional of the type $p(x \mid y)$ to its *inverse* $p(y \mid x)$.

BAYES RULE

$$p(x \mid y) = \frac{p(y \mid x)\, p(x)}{p(y)} = \frac{p(y \mid x)\, p(x)}{\sum_{x'} p(y \mid x')\, p(x')} \quad \text{(discrete)} \tag{2.14}$$

$$p(x \mid y) = \frac{p(y \mid x)\, p(x)}{p(y)} = \frac{p(y \mid x)\, p(x)}{\int p(y \mid x')\, p(x')\, dx'} \quad \text{(continuous)} \tag{2.15}$$

If x is a quantity that we would like to infer from y, the probability $p(x)$ will be referred to as *prior probability distribution* and y is called the *data* —e.g. sensor measurement. The distribution $p(x)$ summarizes the knowledge we have regarding X prior to incorporating the data y. The probability $p(x \mid y)$ is called the *posterior probability distribution* over X. As (2.15) suggests, Bayes rule provides a convenient way to compute a posterior $p(x \mid y)$ using the *inverse* conditional probability $p(y \mid x)$ along with the prior probability $p(x)$. In Robotics, the probability $p(y \mid x)$ is often coined *generative model*, since it describes how variables X *cause* sensor measurement Y.

PRIOR PROBABILITY DISTRIBUTION

POSTERIOR PROBABILITY DISTRIBUTION

GENERATIVE MODEL

The denominator $p(y)$ of Bayes rule does not depend on x. Thus, the factor $p(y)^{-1}$ in (2.14) and (2.15) will be the same for any value x in the posterior $p(x \mid y)$. For this reason, $p(y)^{-1}$ is often written as a *normalizer* in Bayes rule variable generically denoted η

$$p(x \mid y) = \eta p(y \mid x)\, p(x) \tag{2.16}$$

We will simply use the normalization symbol η to indicate that the final result has to be normalized to 1.

It is perfectly fine to condition any of the rules discussed thus far on arbitrary random variables, such as Z. For instance, conditioning Bayes rule on $Z = z$ gives us

$$p(x \mid y, z) = \frac{p(y \mid x, z)\, p(x \mid z)}{p(y \mid z)} \tag{2.17}$$

as long as $p(y \mid z) > 0$.

Similarly, we can condition the rule for combining probabilities of independent random variables (2.8) on other variables z

$$p(x, y \mid z) = p(x \mid z)\, p(y \mid z) \tag{2.18}$$

CONDITIONAL INDEPENDENCE

Such a relation is known as *conditional independence*. It is easy to verify that (2.18) is equivalent to

$$p(x \mid z) = p(x \mid z, y) \tag{2.19}$$

$$p(y \mid z) = p(x \mid z, x) \tag{2.20}$$

Conditional independence plays an important role in probabilistic robotics, since it applies whenever a variable y carries no information about a variable x if another variable's value z is known. Conditional independence does not imply absolute independence

$$p(x, y \mid z) = p(x \mid z)\, p(y \mid z) \not\Rightarrow p(x, y) = p(x)\, p(y) \tag{2.21}$$

The converse is also in general untrue: absolute independence does not imply conditional independence

$$p(x, y) = p(x)\, p(y) \not\Rightarrow p(x, y \mid z) = p(x \mid z)\, p(y \mid z) \tag{2.22}$$

In special cases, conditional and absolute independence may coincide.

EXPECTATION

A number of probabilistic algorithms require us to compute statistics of probability distributions. The *expectation* of a random variable X is given by

$$E[X] = \sum_x x\, p(x) \quad \text{(discrete)} \tag{2.23}$$

$$E[X] = \int x\, p(x)\, dx \quad \text{(continuous)} \tag{2.24}$$

The expectation is a linear function of a random variable

$$E[aX + b] = aE[X] + b \tag{2.25}$$

for arbitrary numerical values a and b. The covariance of X is obtained as follows

$$\mathrm{Cov}[X] = E[X - E[X]]^2 = E[X^2] - E[X]^2 \tag{2.26}$$

The covariance measures the squared expected deviation from the mean μ, for a normal distribution $\mathcal{N}(x; \mu, \Sigma)$.

2.3 Robot Environment Interaction

The *environment*, or *world*, is a dynamical system that possesses internal state. the robot can acquire information about its environment using sensors. However, sensors are noisy and many things cannont be sensed directly. As a consequence, the robot maintains an internal belief of the state of the environment. The robot can also influence the environment through its actuators. The effect of doing so is often somewhat unpredictable. Thus, each control action affects both the environment state and the robot's internal beleif of such state.

ENVIRONMENT

2.3.1 State

Environments are characterized by *state*, which is the collection of all aspects of the robot and its environment that can impact the future. State that canges over time will be called *dynamic state*, which distinguishes if from *static state* or non-changing state. State will be denoted x, and consequently the state at time t will be referred to as x_t. Typical state variables are:

STATE

STATIC STATE

- The robot *pose*, which comprises its location and orientation relative to a global coordinate frame.

  POSE

- The robot *velocity* is commonly referred to as *dynamic state*.

  DYNAMIC STATE

- The *location and features of surrounding objects in the environment* are also state variables. Features of such objects may be their visual appearance. In this work we consider the location of objects in the environment is static. In some problems, objects will assume the form of *landmarks*, which are distinct, stationary features of the environment that can be recognized reliably. Landmarks also denote objects in the environment used for navigation.

  LANDMARKS

- The *location and velocities of moving objects and people* are also potential state variables. This entities possess their own kinematic and dynamic state.

- There are many other state variables, such as the level of the battery charge, sensor health, etc. The list of potential state variables is endless.

A state x_t will be called *complete* if it is the best predictor of the future. Completeness entails that knowledge of past states, measurements, or controls carry no additional information that would help us predict the future more accurately. The future may be stochastic, but no variables prior to x_t may influence the stochastic evolution of future states, unless this dependence is mediated through the state x_t. Temporal processes that meet these conditions are commonly known as *Markov chains*.

COMPLETE

MARKOV CHAINS

In practice, it is impossible to specify a complete state for any realistic robot system. Practical implementations single out a small subset of all state variables, such as the ones listed above. Such a state is called *incomplete state*. In most robotics applications, the state x_t is continuous, but it may be also discrete. State spaces that contain both continuous and discrete variables are called *hybrid* state spaces. In most cases, state changes over time. In this document we consider that time is discrete, i.e. all events will take take place at discrete time steps $t = 0, 1, 2, \ldots$.

INCOMPLETE STATE

2.3.2 Interaction

There are two fundamental types of interaction between a robot and its environment. The robot can influence the state of its environment through its actuators, and it can gather inforamation about the state through its sensors.

Environment measurement Perception is the process by which the robot uses its sensors to obtain information about the state of its environment. The result of such a perceptual interaction is known as a *measurement*, although we will sometimes also call it *observation* or *percept*. Measurements provide information about a momentary state of the environment. The measurement at time t will be denoted z_t. Throughout this thesis, we simply assume that the robot takes only one measurement at a time, for clarity reasons. The notation

MEASUREMENT

$$z^{t_1:t_2} = \{z_{t_1}, z_{t_1+1}, z_{t_1+2}, \ldots, z_{t_2}\} \tag{2.27}$$

denotes the set of all measurements acquired from time t_1 to time t_2, for $t_1 \leq t_2$. If $t_1 = 1$ and $t_2 = t$, we will use the more compact notation $z^t \equiv z^{1:t}$, which represent all measurements up to time t.

Control Control actions change the state of the world. Control data carry information about the *change of state* in the environment. In mobile robotics, a typical example of control data is the velocity of a robot. An alternative source of control data are *odometers*. Odometers are sensors that measure the revolution of a robot's wheels. They convey information about the change of state. Even though odometers are sensors, we will treat odometry as control data, since they measure the effect of a control action. Control data is denoted u_t, which corresponds to the change of state in the time interval $(t-1, t]$. As before, we denote sequences of control data by

ODOMETERS

$$u^{t_1:t_2} = \{u_{t_1}, u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2}\} \tag{2.28}$$

for $t_1 \leq t_2$. Similarly, if $t_1 = 1$ and $t_2 = t$, we will use the more compact notation $u^t \equiv u^{1:t}$, which represent all controls up to time t. Since the environment may change even if a robot does not execute control actions, we assume that there is exactly one control data item per time step t, including *do-nothing* as legal action.

The distinction between measurement and control is crucial. Environment perception provides information about the environment's state, hence it tends to increase the robot's knowledge. Motion, on the other hand, tends to induce a loss of knowledge due to the inherent noise in robo actuation and the stochasticity of robot environments.

2.3.3 Probabilistic Generative Laws

The evolution of state and measurements is governed by probabilistic laws. In general, the state x_t is generated stochastically from the state x_{t-1}. The probabilistic law characterizing the evolution of state might be given by a probability distribution of the form $p(x_t \mid x^{t-1}, z^{t-1}, u^t)$. Notice that we assume here that the robot executes a control action u_t first, and then takes a measurement z_t. If the state x_t is complete then it is a sufficient summary of all that happened in previous time

steps. Thus, from all variables in the expression above, only the control u_t matters if we know the state x_{t-1}

$$p(x_t \mid x^{t-1}, z^{t-1}, u^t) = p(x_t \mid x_{t-1}, u_t) \tag{2.29}$$

One might also want to model the process by which measurements are being generated. Again, if x_t is complete, we have an important conditional independence

$$p(z_t \mid x^t, z^{t-1}, u^t) = p(z_t \mid x_t) \tag{2.30}$$

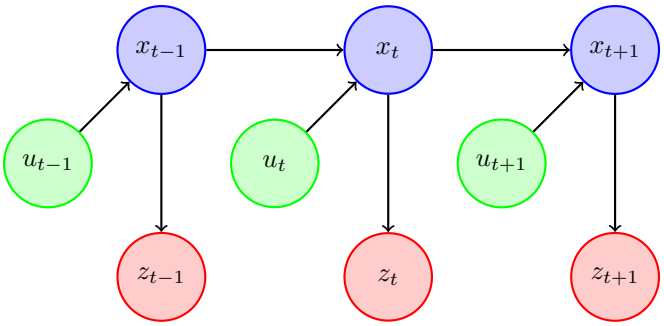


Figure 2.1: Dynamic Bayes Network that characterizes the evolution of states x_t, controls u_t and measurements z_t

Therefore, the two resulting conditional probabilities are $p(x_t \mid x_{t-1}, u_t)$ and $p(z_t \mid x_t)$. The probability $p(x_t \mid x_{t-1}, u_t)$ is the *state transition probability*. It specifies how environmental state evolves over time as a function of robot controls u_t. The probability $p(z_t \mid x_t)$ is called the *measurement probability*. It specifies the probabilitic law according to which measurements z_t are generated from the environment state x_t. Measurements are better thought as noisy projections of the state.

STATE TRANSITION PROBABILITY

MEASUREMENT PROBABILITY

The state transition probability and the measurement probability together describe the dynamical stochastic system of the robot and its environment. Figure 2.1 illustrates the evolution of states and measurements, defined through those probabilities. The state at time t is stochastically dependent on the state at time $t-1$ and the control u_t. The measurement z_t depends stochastically on the state at time t. Such a temporal generative model is alos known as *Hidden Markov Model* (HMM) or *Dynamic Bayes Network* (DBN).

DYNAMIC BAYES NETWORK

2.3.4 Belief Distributions

A *belief* reflects the robot's internal knowledge about the state of the environment. That state cannot be measured directly, so the robot must infer such state. We therefore distinguish the true state from its internal belief with regards to that state. Probabilistic robotics represents beliefs through conditional probability distributions. A belief distribution assigns a probability to each possible hypothesis with regards to the true state. Belief distributions are posterior probabilities over state variables

BELIEF

conditioned on the available data. We will denote the belief over a state variable x_t by $\mathrm{bel}\,(x_t)$, which is an abbreviation for the posterior

$$\mathrm{bel}\,(x_t) = p(x_t \mid z^t, u^t) \tag{2.31}$$

which is the probability distribution over the state x_t at time t, conditioned on all past measurements z^t and all past controls u^t.

Occasionally, it will prove useful to calculate a posterior before incorporating the last measurement z_t, just after executing the control u_t. Such a posterior will be denoted

$$\overline{\mathrm{bel}}\,(x_t) = p(x_t \mid z^{t-1}, u^t) \tag{2.32}$$

PREDICTION which is often referred to as *prediction* in the context of probabilistic filtering. This terminology reflects the fact that $\overline{\mathrm{bel}}\,(x_t)$ predicts the state at time t based on the previous state posterior, before incorporating the measurement z_t at time t.

CORRECTION Calculating $\mathrm{bel}\,(x_t)$ from $\overline{\mathrm{bel}}\,(x_t)$ is called *correction* or *measurement update*.

2.4 Bayes Filters

BAYES FILTER The most general algorithm for calculating beliefs is given by the *Bayes filter* algorithm, shown in Algorithm 1. This algorithm calculates the belief distribution $\mathrm{bel}\,(x_t)$ from measurement and control data. The Bayes filter is recursive, i.e. the belief $\mathrm{bel}\,(x_t)$ at time t is computed from the belief $\mathrm{bel}\,(x_{t-1})$ at time $t-1$. A single

UPDATE RULE iteration of the algorithm is known as the *update rule*, which is applied recursively to calculate the belief $\mathrm{bel}\,(x_t)$ from the belief $\mathrm{bel}\,(x_{t-1})$ at $t-1$, along with the most recent control u_t and measurement z_t.

Algorithm 1 Bayes Filter

Require: Belief $bel(x_{t-1})$ over state x_{t-1} in the previous temporal step and control u_t and measurement z_t obtained in the current temporal step.
Ensure: Belief $bel(x_t)$ over state x_t in the current temporal step.
Algorithm: BayesFilter$(bel(x_{t-1}),\ u_t,\ z_t)$ **return** $bel(x_t)$

1: **for all** x_t **do**
2: $\quad \overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx_{t-1}$
3: $\quad bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$
4: **end for**
5: **return** $bel(x_t)$

PREDICTION The Bayes filter algorithm has two essential steps. The first update step is called the control update or *prediction*. In line 2, the control u_t is processed by calculating a belief over the state x_t based on the prior belief over state x_{t-1} and u_t. The

MEASUREMENT UPDATE second step is called *measurement update*. In line 3, the Bayes filter algorithm multiplies the belief $\overline{\mathrm{bel}}\,(x_t)$ by the probability that the measurement z_t may have been observed. The result is normalized, leading to the final belief $\mathrm{bel}\,(x_t)$.

To compute the posterior belief recursively, the algorithm requires an initial belief $\mathrm{bel}\,(x_0)$ at time $t = 0$ as boundary condition. If the value of x_0 is known with certainty, $\mathrm{bel}\,(x_0)$ should be initialized with a point mass distribution that centers all probability mass on x_0 and assigns zero probability anywhere else. Otherwise, if x_0

is completely unknown, then $\mathrm{bel}\,(x_0)$ may be initialized using a uniform distribution over the domain of x_0. Partial knowledge of x_0 can be expressed by non-uniform distributions, but this case is uncommon in practice.

The Bayes filter algorithm can only be implemented in the form stated here for very simple estimation problems. In particular, we either need to carry out the integration in line 2 and the multiplication in line 3 in closed form, or we need to restrict ourselves to finite state spaces, so that the integral in line 2 becomes a finite sum. For this reason, the two following chapters will discuss different approaches to implement the Bayes filter in practice. In general robotics problems, beliefs have to be approximated. The nature of the approximation has important ramifications on the complexity of the algorithm. Finding a suitable approximation is usually a challenging problem, with no unique best answer for all robotics problems. One has to trade off a range of properties:

1. **Computational efficiency.** Some approximations, such as Gaussian approximations discussed in Chapter 3 make it possible to calculate beliefs in time polynomial in the dimension of the state space, while others may require exponential time. Particle-based techniques, discussed in Section 4.2, have an *any-time* characteristic, enabling them to trade off accuracy with computational efficiency.

2. **Accuracy.** Some approximations can approximate a wider range of distributions more tightly than others. For example, linear Gaussian approximations are limited to unimodal distributions, whereas histogram representations can approximate multi-modal distributions, albeit with limited accuracy. Particle representations can approximate a wide set of distributions, but the number of particles needed to attain a desired accuracy can be large.

3. **Ease of implementation.** The difficulty of implementation depends on a variety of factors, such as the form of the measurement probability $p(z_t \mid x_t)$ and the state transition probability $p(x_t \mid x_{t-1}, u_t)$. Particle representations often yield surprisingly simple implementations for complex nonlinear systems.

2.4.1 Markov Assumption

MARKOV ASSUMPTION

The *Markov assumption* postulates that past and future data are independent if one knows the current state x_t. There exist a number of factors that may have a systematic effect on sensor readings, such as unmodeled dynamics not included in x_t, inaccuracies in the probabilistic models $p(z_t \mid x_t)$ and $p(x_t \mid x_{t-1}, u_t)$, approximation errors due to approximate representations of belief functions, and so on. Thus, they induce violations of the Markov assumption.

In principle, many of these variables can be included in state representations, but incomplete state representations are often preferable to reduce the computational complexity of the Bayes filter algorithm. In practice, Bayes filters are surprisingly robust to such violations. As a general rule of thumb one should exercise care when defining the state x_t, so that the effect of unmodeled state variables has close-to-random effects [149].

Chapter 3

Parametric Filters

3.1 Introduction

This chapter describes an important family of recursive state estimators called *Gaussian filters*. Guassian techniques all share the basic idea that beliefs are represented by multivariate normal distributions

GAUSSIAN FILTERS

$$p(x) = \left|2\pi\Sigma\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{3.1}$$

This density over the variable x is characterized by two sets of parameters μ and Σ. The mean μ is a vector that possesses the same dimensionality as the state x. The covariance is a quadratic matrix that is symmetric and positive-semidefinite, with dimension equal to the dimensionality of the state x squared. Gaussians are unimodal, i.e. they possess a single maximum. Such 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. However, they are a poor match for many global estimation problems in which many distinct hypotheses exist.

The parametrization of a Gaussian by its mean μ and covariance Σ is called the *moments parametrization*, becuase the mean and covariance are the first and seconde moments of a probability distribution. There exists an alternative parametrization called *canonical parametrization* or *natural parametrization*. The moments and the canonical parametrizations are best thought of as duals: what appears to be computationally easy in one parametrization is involved in the other, and vice versa. The filters that rely on a fixed functional form of the posterior, having a parametrization, are clasified as *Parametric Filters*. We will only discuss Gaussians filters for reasons commented in the sequel. Most of the theorical foundations of such filters have been taken from [149], that might be useful as a complementaty reading.

MOMENTS PARAMETRIZATION

CANONICAL PARAMETRIZATION

PARAMETRIC FILTERS

3.2 The Kalman Filter

In 1960, Kalman published his famous paper describing a recursive solution to the discrete-data linear filtering problem [73]. Since that time the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assited navigation. They are the classical approach to generating

maps and to a certain extent they are part of comtemporary SLAM methods [147].

KALMAN FILTER The *Kalman Filter* (KF) is a Bayes filter that the belief with a normal distribution. It estimates the state of a process, in a way that minimizes the mean of the squared error [159].

3.2.1 Linear Gaussian Systems

The Kalman filter was invented as a technique for filtering and prediction in *linear Gaussian systems*. The Kalman filter implements belief computation for continuous states and it is not applicable to discrete state spaces. It estimates the state $x_t \in \Re^n$ of a discrete-time controlled process that is governed by the a linear stochastic differential equation. Posteriors are Gaussian if the following three properties hold, in addition to the Markov assumptions of the Bayes filter.

1. The state transition $p(x_t \mid u_t, x_{t-1})$ must be a *linear* function in its arguments with added Gaussian noise. This is expressed by the equation

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t \tag{3.2}$$

where x_t and x_{t-1} are state vectors, and u_t is the control vector at time t. A_t is a square matrix of size $n \times n$, where n is the dimension of the state vector x_t. B_t is a matrix of size $n \times m$, with m being the dimension of the control vector u_t. By multiplying the state and control vector with the matrices A_t and B_t, respectively, the state transition function becomes *linear* in its arguments. Thus, Kalman filters assume linear system dynamics. For this reason, the filter is also called *Linear Kalman Filter* (LKF) in some texts.

LINEAR KALMAN FILTER

The random variable ε_t in (3.2) is a Gaussian random vector that models the uncertainty introduced by the state transition. It is of the same dimension as the state vector, with mean 0 and covariance R_t. A state transition probability $p(x_t \mid u_t, x_{t-1})$ of the form (3.2) is called a *linear Gaussian* to reflect that it is linear in its arguments with additive Gaussian noise

LINEAR GAUSSIAN

$$p(x_t \mid u_t, x_{t-1}) = \left|2\pi R_t\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x_t - A_t x_{t-1} - B_t u_t)\right\} \tag{3.3}$$

2. The process must be observable and the measurement probability $p(z_t \mid x_t)$ must also be *linear* in its arguments, with added Gaussian noise

$$z_t = C_t x_t + \delta_t \tag{3.4}$$

where C_t is a matrix of size $k \times n$, with k being the dimension of the measurement vector z_t. The vector δ_t describes the measurement noise with a multivariate Guassian distribution with zero mean and covariance Q_t. The measurement probability $p(z_t \mid x_t)$ is thus given by the following multivariate normal distribution

$$p(z_t \mid x_t) = \left|2\pi Q_t\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\right\} \tag{3.5}$$

3. Finally, the initial belief $\mathrm{bel}\,(x_0)$ must be normally distributed. We will denote the mean of this belief by μ_0 and the covariance by Σ_0

$$\mathrm{bel}\,(x_0) = p(x_0) = \left|2\pi\Sigma_0\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_0-\mu_0)^T\Sigma_0^{-1}(x_0-\mu_0)\right\} \quad (3.6)$$

The three assumptions above are sufficient to ensure that the posterior $\mathrm{bel}\,(x_t)$ is always a Gaussian for any point in time t. The reader is invited to consult the proof of this non-trivial result in [149].

3.2.2 The KF Algorithm

Kalman filters represent the belief $\mathrm{bel}\,(x_t)$ at time t by the mean μ_t and the covariance Σ_t. The input of the Kalman filter is the belief $\mathrm{bel}\,(x_{t-1})$ at time $t-1$, the control u_t and the measurement z_t, as shown in Algorithm 2. The output is the belief at time t. The Kalman filter estimates a process by using a form of *feedback control* since it first estimates the state $\bar{x}_t$ and then obtains feedback in the form of the measurement z_t, which is used to correct the estimation. Therefore, the equations of the Kalman filter fall into two groups. The *time update* or *prediction* PREDICTION equations, that are responsible for projecting forward in time the current state mean μ_{t-1} and covariance Σ_{t-1} estimates to obtain the *a priori* estimates $\bar{\mu}_t$ and $\bar{\Sigma}_t$ for the next time step. And the *measurement update* or *correction* equations, that are CORRECTION responsible for the feeback, i.e. for incorporating a new measurement z_t into the *a priori* estimate to obtain an improve *a posterior* estimate represented by μ_t and Σ_t. The final estimation algorithm resembles that of a *predictor-corrector* algorithm [159].

Algorithm 2 Kalman Filter

Require: Mean μ_{t-1} and covariance Σ_{t-1} obtained in the previous temporal step, that will be corrected using the control u_t and observation z_t of the current step.
Ensure: Mean μ_t and covariance Σ_t filtered.
Algorithm: **KF**(μ_{t-1}, Σ_{t-1}, u_t, z_t) **return** μ_t, Σ_t

▷ Prediction:
1: $\bar{\mu}_t = A_t\mu_{t-1} + B_tu_t$
2: $\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$
▷ Correction:
3: $K_t = \bar{\Sigma}_tC_t^T\left(C_t\bar{\Sigma}_tC_t^T + Q_t\right)^{-1}$ ▷ Kalman Gain
4: $\mu_t = \bar{\mu}_t + K_t\left(z_t - C_t\bar{\mu}_t\right)$
5: $\Sigma_t = \left(I - K_tC_t\right)\bar{\Sigma}_t$
6: **return** μ_t, Σ_t

In lines 1 and 2, the predicted belief $\overline{\mathrm{bel}}\,(x_t)$, represented by $\bar{\mu}_t$ and $\bar{\Sigma}_t$, is calculated before incorporating the measurement z_t. This belief is obtained by incorporating the control u_t. The mean is updated using the state transition function (3.2). The update of the covariance considers the fact that states depend on previous states through the linea matrix A_t.

The belief $\overline{\mathrm{bel}}\,(x_t)$ is then transformed into the desired belief $\mathrm{bel}\,(x_t)$ in lines 3 through 5, by incorporating the measurement z_t. The variable K_t computed in

KALMAN GAIN line 3 is called *Kalman gain*. It specifies the degree to which the measurement is incorporated into the new state estimate, in a way that the reader may consult in the mathematical derivation given in [149]. Line 4 manipulates the mean by
INNOVATION adjusting it in proportion to K_t and the *innovation*, which is the difference between the actual measurement z_t and the expected $C_t\bar{\mu}_t$. The new covariance of the posterior belief is computed in line 5, adjusted for the information gain resulting from the measurement.

The Kalman filter is computationally quite efficient, since after each predictor-corrector step, the process is repeated with the previous *a posteriori* estimates used to project the new *a priori* estimates. This recursive nature is one of the very appealing features of the Kalman filter, that makes it more feasible than other filters like the Weiner filter, which is designed to operate on *all* of the data *directly* for each estimate [21]. The Kalman filter instead recursively conditions the current estimate on all of the past measurements.

3.3 The Extended Kalman Filter

The assumptions that the observations are linear functions of the state and that the next state is a linear function of the previous state are crucial for the correctness of the Kalman filter. Unfortunately, state transitions and measurements are rarely
EXTENDED KALMAN FILTER linear in practice [149]. The *Extended Kalman filter* (EKF) relaxes the linearity assumption.

3.3.1 Linearization

Here the assumption is that the state transition probability and the measurement probabilities are governed by *nonlinear* functions g and h [33, 147, 149, 159], respectively

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t \tag{3.7}$$

$$z_t = h(x_t) + \delta_t \tag{3.8}$$

This model strictly generalizes the linear Gaussian model underlying Kalman filters. The function g replaces the matrices A_t and B_t in (3.2), and h replaces the matrix C_t in (3.4). Unfortunately, with arbitrary functions g and h, the belief is no longer a Gaussian. The key idea underlying the EKF approximation is called
LINEARIZATION *linearization*. It approximates the nonlinear function g by a linear function that is tangent to g at the mean of the Gaussian. A similar linearization is applied to h. There exist many techniques for linearizing nonlinear functions. EKFs utilize a
TAYLOR EXPANSION method called (first order) *Taylor expansion*. It constructs a linear approximation to a function g from its value and slope. The slope is given by the partial derivate with repect to the state. Hence, the nonlinear functions g and h have the form

$$\begin{aligned} g(u_t, x_{t-1}) &\approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1}) \\ &= g(u_t, \mu_{t-1}) + G_t(x_{t-1} - \mu_{t-1}) \end{aligned} \tag{3.9}$$

$$\begin{aligned} h(x_t) &\approx h(\bar{\mu}_t) + h'(\bar{\mu}_t)(x_t - \bar{\mu}_t) \\ &= h(\bar{\mu}_t) + H_t(x_t - \bar{\mu}_t) \end{aligned} \tag{3.10}$$

where the G_t is the Jacobian of the state transition model and H_t is the Jacobian of the measurement model. G_t is a matrix of size $n \times n$, with n denoting the

dimension of the state. The value of G_t depends on u_t and μ_{t-1}, hence it differs for different points in time. Similarly, H_t is a matrix of size $n \times n$ and depends on $\bar{\mu}_t$. The reader interested in a more detailed explanation of the linearization and the mathematical derivation of the EKF might consult [149].

3.3.2 The EKF Algorithm

The Algorithm 3 computes the Extended Kalman filter, that replaces the state transition and measurement funcions (3.2) and (3.4) with the nolinear functions (3.9) and (3.10), respectively. This is the main modification introduced in the EKF algorithm. The mean $\bar{\mu}_t$ of the predicted state is computed using g in line 1. And the mean μ_t of the corrected state is computed using h in line 4.

Algorithm 3 Extended Kalman Filter

Require: Mean μ_{t-1} and covariance Σ_{t-1} obtained in the previous temporal step, that will be corrected using the control u_t and observation z_t of the current step.

Ensure: Mean μ_t and covariance Σ_t filtered.

Algorithm: EKF$(\mu_{t-1},\ \Sigma_{t-1},\ u_t,\ z_t)$ **return** $\mu_t,\ \Sigma_t$

▷ Prediction:

1: $\bar{\mu}_t = g\left(u_t, \mu_{t-1}\right)$
2: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$

▷ Correction:

3: $K_t = \bar{\Sigma}_t H_t^T \left(H_t \bar{\Sigma}_t H_t^T + Q_t\right)^{-1}$ ▷ Kalman Gain
4: $\mu_t = \bar{\mu}_t + K_t \left(z_t - h\left(\bar{\mu}_t\right)\right)$
5: $\Sigma_t = \left(I - K_t H_t\right) \bar{\Sigma}_t$
6: **return** $\mu_t,\ \Sigma_t$

The Jacobians G_t and H_t are used to compute the values of the covariances. The covariance $\bar{\Sigma}_t$ of the predicted state uses G_t instead of A_t in line 2. And the covariance Σ_t of the corrected state uses H_t instead of C_t in line 5. The Kalman gain K_t is also computed using the Jacobian H_t.

Chapter 4

Nonparametric Filters

4.1 Introduction

A popular alternative to parametric techniques are *nonparametric filters*. Nonparametric filters do not rely on a fixed functional form of the posterior, such as parametric filters. Instead, they approximate posteriors by a finite number of values, each corresponding to a region in state space. The quality of the approximation depends on the number of parameters used to represent the posterior. As the number of parameters goes to infinity, nonparametric techniques tend to converge to the correct posterior.

NONPARAMETRIC FILTERS

This chapter describes a parametric technique that represents posteriors by finitely many samples, known as *particle filter*. We also introduce in brief another approach that decomposes the state into finitely many regions, and represents the posterior by a histogram. The reader might consult [149] for further reading, since the contents of the following sections have been taken mostly from it.

Both techniques, histograms and particle filters, do not make strong parametric assumptions on the posterior density. Thus, they are well-suited to represent multimodal beliefs. For this reason, they are often the method of choice when a robot has to cope with global uncertainty or hard data association problems that yield separate, distinct hypotheses.

However, the representational power of these techniques comes at the price of a higher computational complexity. Fortunately, it is possible to adapt the number of parameters to the complexity of the posterior. Techniques that can adapt the number of parameters to represent the posterior online are called *adaptive*. In particular, if they can adapt based on the computational resources available for belief computation, they are called *resource-adaptive*. They play an important role in Robotics, since they enable robots to make decisions in real time, regardless of the computational resources available. In fact, particle filters are often implemented as a resource-adaptive algorithm, by adapting the number of particles online [54].

RESOURCE-ADAPTIVE

4.2 The Particle Filter

The *Particle filter* is a nonparametric implementation of the Bayes filter that approximate the posterior by a finite number of parameters [149]. Particle filters are

PARTICLE FILTER

sequential Monte Carlo (MC) methods based on point mass or *particle* representations of probability densities [5, 51, 66]. The key idea of the particle filter is to represent the posterior $\mathrm{bel}(x_t)$ by a set of random state samples drawn from this posterior, with associated weights [5, 42]. As the number of particles M becomes very large, this MC characterization becomes an equivalent representation to the usual functional description of the posterior and the filter approaches the optimal Bayesian estimate.

The FastSLAM algorithm discussed in the present work based on a particle filter to compute the SLAM posterior. Although this will be later discussed in Chapter 9 and 10, note that the basic FastSLAM algorithm applies most of the fundamentals introduced here. Furthermore, the FastSLAM 2.0 version uses an

EXTENDED PARTICLE FILTER

Extended Particle filter (EPF), that incorporates the current measurement z_t into the proposal distribution, not just the importance weights, in order to better match the posterior.

4.2.1 Basic Algorithm

PARTICLES

In particle filters, the samples of a posterior distribution are called *particles* and are denoted

$$\mathcal{X}_t = \left\{ x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]} \right\} \tag{4.1}$$

Each particle $x_t^{[k]}$, with $k = 1, \ldots, M$, is a concrete instantiation of the state at time t. Put differently, a particle is a hypothesis as to what the true world state may be at time t. Here M denotes the number of particles in the particle set $\mathcal{X}_t$.

The most basic variant of the particle filter algorithm is stated in Algorithm 4. The algorithm first constructs a temporary particle set $\bar{\mathcal{X}}_t$ that represents the belief $\overline{\mathrm{bel}}(x_t)$. It does this by samping a hypothetical state $x_t^{[k]}$ from the state transition distribution $p(x_t \mid x_{t-1}, u_t)$ in line 4. Then, line 5 calculates the so-called

IMPORTANCE FACTOR

importance factor $w_t^{[k]}$ for each particle state $x_t^{[k]}$. The importance is the probability of the measurement z_t under the state $x_t^{[k]}$, given by $p(x_t \mid x_t^{[k]})$. If we interpret $w_t^{[k]}$ as the *weight* of a particle, this yields the set of weighted particles $\bar{\mathcal{X}}_t$.

The real *trick* of the particle filter algorithm starts at line 9. These lines implement the *resampling* step. It draws with replacement M particles from the temporary set $\bar{\mathcal{X}}_t$. The probability of drawing each particle is given by its importance weight. After the resampling, the particles are distributed approximately according to the posterior $\mathrm{bel}(x_t)$. The resampling step is a probabilistic implementation of the Darwinian idea of *survival of the fittest*, since 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.

RESAMPLING

The particle set distribution is depicted in Figure 4.1. The target distribution $p(x)$ is approximated by a set of particles $\hat{x}^{[k]}$ drawn according with Algorithm 4 discussed above. Initially, samples are drawn from a proposal distribution and then they are resampled according with the importance weight computed for each temporary particle. In regions where the target distribution is larger than the proposal distribution, the samples receive higher weights. As a result, samples in this region will be picked more often. Contrary, in regions where the target distribution is smaller, the samples will be given lower weights. In the limit of infinite samples, this procedure will produce samples distributed according to the target distribution.

Algorithm 4 Particle Filter

Require: Particle set $\mathcal{X}_{t-1}$ in previous temporal step and control u_t and observation z_t obtained in current temporal step.

Ensure: Sample new particles $x_t^{[k]}$ to construct particle set $\bar{\mathcal{X}}_t$ in current temporal step. A resampling process is applied to this set to obtain the final particle set $\mathcal{X}_t$.

Algorithm: ParticleFilter($\mathcal{X}_{t-1}$, u_t, z_t) **return** $\mathcal{X}_t$

1: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \varnothing$
2: $\bar{\mathcal{W}}_t = \varnothing$
3: **for all** particle $x_t^{[k]} \in \mathcal{X}_{t-1}$ **do**
4: sample $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$
5: $w_t^{[k]} = p(z_t \mid x_t^{[k]})$
6: add $x_t^{[k]}$ to $\bar{\mathcal{X}}_t$
7: add $w_t^{[k]}$ to $\bar{\mathcal{W}}_t$
8: **end for**
9: **for all** particle $x_t^{[k]} \in \bar{\mathcal{X}}_t$ **do** ▷ resampling
10: draw i with probability $\propto w_t^{[i]}$
11: add $x_t^{[i]}$ to $\mathcal{X}_t$
12: **end for**
13: **return** $\mathcal{X}_t$

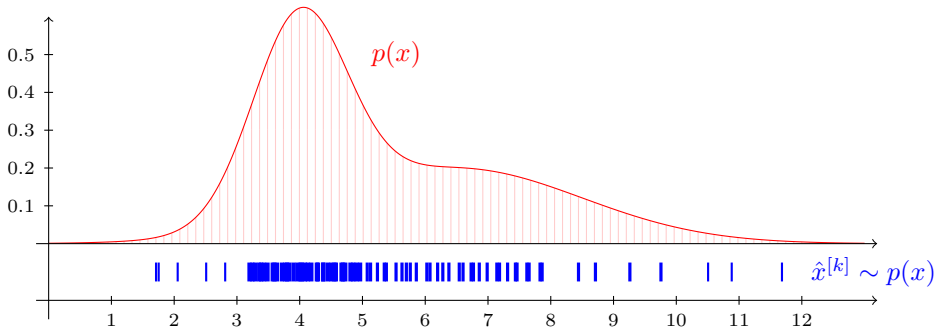


Figure 4.1: Particle filter distribution. Importance sampling draws samples from the proposal distribution and the resampling step draws particles with probability proportional to their importance weights

4.2.2 Importance Sampling

SEQUENTIAL IMPORTANCE SAMPLING

The *Sequential Importance Sampling* (SIS) algorithm is a MC method that forms the basis for most sequential MC filters developed over the past decades [5, 42, 43, 66]. This sequential MC (SMC) approach is known variously as *bootstrap filtering*, the *condensation* algorithm, *particle filter*, interacting particle approximations, and *survival of the fittest* [5].

Let the state x_t of a stochastic process governed by the function

$$x_t = f_t(x_{t-1}, \varepsilon_{t-1}) \tag{4.2}$$

possibly nonlinear, where ε_{t-1} is an iid (independent and identically distributed) noise sequence. The process can be observed according with

$$z_t = h_t(x_t, \delta_t) \tag{4.3}$$

also possibly nonlinear, where δ_t is an iid noise sequence.

In particular, we seek filtered estimates of x_t based on the set of all available measurements z^t up to time t. From a Bayesian perspective, the problem is to recursively calculate some degreee of belief in the state x_t at time t given the measurements z^t up to time t. Thus, it is required to construct the posterior $p(x_t \mid z^t)$.

Let $\left\{x^{t,[k]}, w_t^{[k]}\right\}_{k=1}^{M}$ denote a set of M particles that characterizes the posterior $p(x_t \mid z^t)$, where $\left\{x^{t,[k]}\right\}_{k=1}^{M}$ is a set of states with associated weights $\left\{w_t^{[k]}\right\}_{k=1}^{M}$, and let the set of all states x^t and all measurements z^t up to time t. The weights are normalized such that $\sum_{k=1}^{M} w_t^{[k]} = 1$. Then, the posterior at time t can be approximated as

$$p(x^t \mid z^t) \approx \sum_{k=1}^{M} w_t^{[k]}\, \delta(x^t - x^{t,[k]}) \tag{4.4}$$

where $\delta(\,\cdot\,)$ is the Dirac delta function.

We therefore have a discrete weighted approximation to the true posterior $p(x^t \mid z^t)$. The weights are chosen using the principle of *importance sampling* [41, 42, 43], which relies on the following. Suppose $p(x) \propto \pi(x)$ is a probability density from which it is difficult to draw samples but for which $\pi(x)$ can be evaluated. In addition, let $x^k \sim q(x)$ for $k = 1, \ldots, M$ be samples that are easily generated from a proposal $q(\,\cdot\,)$ called an *importance density*. Then, a weighted approximation to the density $p(\,\cdot\,)$ is given by

IMPORTANCE SAMPLING

IMPORTANCE DENSITY

$$p(x) \approx \sum_{k=1}^{M} w^{[k]}\, \delta(x - x^{[k]}) \tag{4.5}$$

where

$$w^{[k]} \propto \frac{\pi(x^{[k]})}{q(x^{[k]})} \tag{4.6}$$

is the normalized weight of the k-th particle.

Therefore, if the samples $x^{t,[k]}$ were drawn from an importance density $p(x^{t,[k]} \mid z^t)$, then the weights in (4.4) are defined by (4.6) to be

$$w_t^{[k]} \propto \frac{p(x^{t,[k]} \mid z^t)}{q(x^{t,[k]} \mid z^t)} \tag{4.7}$$

Returning to the sequential case, at each iteration one could have samples constituting an approximation of $p(x^{t-1,[k]} \mid z^{t-1})$ and want to approximate $p(x^{t,[k]} \mid z^t)$ with a new set of samples. If the importance density is chosen to factorize such that

$$q(x^{t,[k]} \mid z^t) = q(x_t^{[k]} \mid x^{t-1,[k]}, z^t)\; q(x^{t-1,[k]} \mid z^{t-1}) \tag{4.8}$$

then one can obtain samples $x^{t,[k]} \sim q(x^{t,[k]} \mid z^t)$ by augmenting each of the existing samples $x^{t-1,[k]} \sim q(x^{t-1,[k]} \mid z^{t-1})$ with the new state $x_t^{[k]} \sim q(x_t^{[k]} \mid x^{t-1,[k]}, z^t)$.

After some operations [5], this yields the following expression to compute the weights

$$w_t^{[k]} \propto w_{t-1}^{[k]} \frac{p(z_t \mid x_t^{[k]})\, p(x_t^{[k]} \mid x_{t-1}^{[k]})}{q(x_t^{[k]} \mid x^{t-1,[k]}, z^t)} \tag{4.9}$$

Furthermore, if $q(x_t^{[k]} \mid x^{t-1,[k]}, z^t) = q(x_t^{[k]} \mid x_{t-1}^{[k]}, z_t)$, then the importance density becomes only dependent on $x_{t-1}^{[k]}$ and z_t. This is particularly useful when only a filtered estimate of $p(x_t \mid z^t)$ is required. In such scenarios, only x_t^k need be stored and therefore one can discard the path $x^{t-1,[k]}$ and history of measurements z^{t-1}. The modified weight is then

$$w_t^{[k]} \propto w_{t-1}^{[k]} \frac{p(z_t \mid x_t^{[k]})\, p(x_t^{[k]} \mid x_{t-1}^{[k]})}{q(x_t^{[k]} \mid x_{t-1}^{[k]}, z_t)} \tag{4.10}$$

and the posterior filtered density $p(x_t \mid z^t)$ can be approximated as

$$p(x_t \mid z^t) \approx \sum_{k=1}^{M} w_t^{[k]}\, \delta(x_t - x_t^{[k]}) \tag{4.11}$$

where the weights are defined in (4.10).

Algorithm 5 Sequential Importance Sampling

Require: Particle set $\mathcal{X}_{t-1}$ with M elements and associated weights $\mathcal{W}_{t-1}$, and observation z_t.

Ensure: Sequential Importance Sampling (SIS) leaves the sampled particles in the set $\mathcal{X}_t$ with associated weights in $\mathcal{W}_t$.

Algorithm: SIS$(\mathcal{X}_{t-1},\, \mathcal{W}_{t-1},\, z_t)$ **return** $\langle \mathcal{X}_t, \mathcal{W}_t \rangle$

1: **for** $k = 1$ **to** M **do**
2: draw $x_t^{[k]} \sim q(x_t^{[k]} \mid x_{t-1}^{[k]}, z_t)$ ▷ sample particle
3: $w_t^{[k]} \approx w_{t-1}^{[k]} \dfrac{p(z_t \mid x_t^{[k]})\, p(x_t^{[k]} \mid x_{t-1}^{[k]})}{q(x_t^{[k]} \mid x_{t-1}^{[k]}, z_t)}$ ▷ assign the particle a weight
4: add $x_t^{[k]}$ to $\mathcal{X}_t$
5: add $w_t^{[k]}$ to $\mathcal{W}_t$
6: **end for**
7: **return** $\langle \mathcal{X}_t, \mathcal{W}_t \rangle$

The SIS algorithm thus consists of recursive propagation of the weights and state support points as each measurement is received sequentially. The description of this algorithm is summarized in Algorithm 5.

An important ramification of the SIS filter in SLAM is the *Sampling Importance Resampling* (SIR) filter. The SIR filter proposed in [60] is a Monte Carlo method that can be applied to recurvise Bayesian filtering problems. The assumptions required to use the SIR filter are very weak:

SAMPLING IMPORTANCE RESAMPLING

1. The state dynamics and measurement functions f_t and h_t in (4.2) and (4.3), need to be known.

2. It is required to be able to sample from the process noise distritubtion ε_{t-1} and from the prior.

3. The likelihood function $p(z_t \mid x_t)$ needs to be available for pointwise evaluation.

The SIR filter uses the importance density

$$q(x_t \mid x^{t-1,[k]}, z^t) = p(x_t \mid x^{t-1,[k]}) \tag{4.12}$$

and it incorporates a resampling step, which is to be applied at every time index. For this particular choice of importance density, the weights are given by

$$w_t^{[k]} \propto w_{t-1}^{[k]}\, p(z_t \mid x_t^{[k]}) \tag{4.13}$$

4.2.3 Practical Considerations and Properties

Density Extraction

DENSITY ESTIMATION

The sample sets maintained by particle filters represent discrete approxiamations of continuous beliefs. The problem of extracting a continuous density from such samples is called *density estimation*. There exist several approaches to density estimation. The decision of which density extraction technique should be used depends on the problem at hand.

A simple and highly efficient approach is to compute a *Gaussian approximation*. A Gaussian approximation captures only basic properties of a density, and it is only appropiate if the density is unimodal. Multimodal sample distributions require more complex techniques such as *k-means clustering*, which approximates a density using mixtures of Gaussians.

In an alternative approach, a discrete *histogram* is superimposed over the state space and the probability of each bin is computed by summing the weights of the particles that fall into its range. An important shortcoming of this technique is the fact that the space complexity is exponentical in the number of dimensions. On the other hand, histograms can represent multimodal distributions, they can be computed efficiently, and the density at any state can be extracted in time independent of the number of particles. The space complexity of histogram representations can be reduced significantly by generating a *density tree* from the particles.

Kernel density estimation is another way of converting a particle set into a continuous density. Each particle is used as the center of a so-called kernel, and the overall density is given by a mixture of the kernel densities. The advantage of this approach is the smoothness and algorithmic simplicity. However, the complexity of computing the density at any point is linear in the number of particles, or kernels.

Sampling Variance

VARIANCE

An important source of error in the particle filter relates to the variation inherent in random sampling. Whenever a finite number of samples is drawn from a density, statistics extracted from these samples differ slightly from those of the original density. Variability due to random sampling is called the *variance* of the sampler. Fortunately, the sampling variance decreases with the number of samples. Obviously, the higher the number of samples results in more accurate approximations with less variability.

Resampling

The *resampling* process is a technique frequently used in conjunction with particle filters to mitigate the degeneracy problem [5, 41, 42, 43]. A common problem with the SIS particle filter is the *degeneracy problem*, where after a few iterations, all but one particle will have negligible weight. It has been shown [42] that the variance of the importance weights can only increase over time, and thus, it is impossible to avoid the degeneracy phenomenon. This degeneracy implies that a large computational effort is devoted to updating particles whose contribution to the approximation $p(x_t \mid z^t)$ is almost zero. A suitable measure of the degeneracy of the algorithm is the effective sample size N_{eff} [5] defined as

RESAMPLING

DEGENERACY PROBLEM

$$N_{\text{eff}} = \frac{M}{1 + \text{Var}(\hat{w}_t^{[k]})} \tag{4.14}$$

where $\text{Var}(\hat{w}_t^{[k]})$ is the sampling variance, and

$$\hat{w}_t^{[k]} = \frac{p(x_t^{[k]} \mid z^t)}{q(x_t^{[k]} \mid x_{t-1}^{[k]}, z_t)} \tag{4.15}$$

is referred to as the *true weight*. This cannot be evaluated exactly, but an estimate of N_{eff} can be obtained by

TRUE WEIGHT

$$\text{ESS} = \frac{1}{\sum_{k=1}^{M} \left(w_t^{[k]} \right)^2} \tag{4.16}$$

which is known as the *Effective Sample Size* (ESS), where $w_t^{[k]}$ is the normalized weight obtained using (4.9).

EFFECTIVE SAMPLE SIZE

There exist other measure of the degeneracy, which are the *Coefficient of Variation* (CV)

COEFFICIENT OF VARIATION

$$\text{CV} = \sqrt{\frac{1}{M} \sum_{k=1}^{M} \left(M w_t^{[k]} - 1 \right)^2} \tag{4.17}$$

and even the *entropy* of the weights $H(\mathcal{W})$, defined as

ENTROPY

$$H(\mathcal{W}) = -\sum_{k=1}^{M} w_t^{[k]} \log_2 w_t^{[k]} \tag{4.18}$$

Such metrics are outside the scope of this thesis. The reader might consult [41, 43] for further insight on them.

Returning to the ESS, notice that ESS $\leq M$ and that small values of ESS indicates severe degeneracy. There exist a variety of methods to reduce the effect of degeneracy. The brute force approach uses a very large M, but this is computationally undesirable. More effective methods are a good choice of the importance density and the use of resampling [5], which is the preferred technique in the present work, since the FastSLAM method impose an importance density function and recomends resampling [92, 93, 95, 96, 149].

The basic idea of resampling is to eliminate particles that have small weights and to concentrate on particles with large weights. The resampling step involves generating a new set $\left\{\hat{x}_t^{[k]}\right\}_{k=1}^{M}$ by resampling with replacement M times from an approximate discrete representation of $p(x_t \mid z^t)$ given by

$$p(x_t \mid z^t) \approx \sum_{k=1}^{M} w_t^{[k]}\, \delta(x_t - x_t^{[k]}) \tag{4.19}$$

so that the probability of drawing a particular particle j is proportional to its weight

$$P(\hat{x}_t^{[k]} = x_t^{[j]}) = w_t^{[j]} \tag{4.20}$$

The resulting sample is in fact an iid sample from the discrete density (4.19), and therefore the weights are now reset to $x_t^{[k]} = \frac{1}{M}$. There exist two important aspects regarding resampling. Firstly, it is possible to determine when to resample or not. The resampling step might be performed always or only when the number of effective particles N_{eff} goes down a fixed threshold. Secondly, one may choose the type of resampling. There exist a variety of resampling methods, such as *sequential resampling*, which is proposed by the authors of FastSLAM [93, 149], and *stratified resampling*. In Section 4.2.4 both resampling methods are discussed in brief. Resampling methods based on probabilitic foundations outside the scope of this work, which are developed by Madow in a series of papers [85, 86, 87].

Sampling Bias

BIAS The fact that only finitely many particles are used introduces a systematic *bias* in the posterior estimate. Consider the extreme case of $M = 1$ particle. The key insight is that the resampling step *deterministically* accepts the sample, regardless of its importance factor $w_t^{[k]}$. Thus, the particle filter flatly ignores all measurements z^t. The culprit is the normalization implicit in the resampling step. When sampling in proportion to the importance weights, in line 10 of Algorithm 4, $w_t^{[k]}$ becomes its own normalizer if $M = 1$

$$p(\text{draw } x_t^{[k]}) = \frac{w_t^{[k]}}{w_t^{[k]}} = 1 \tag{4.21}$$

In general, the problem is that the non-normalized weights $\mathcal{W}_t$ are drawn from an M-dimensional space, but after normalization they reside in a space of dimension $M - 1$. This is because after normalization, the k-th weight can be recovered from the $M - 1$ other weights by substracting those from 1. Fortunately, for large values of M, the effect of loss of dimensionality becomes less pronounced.

Particle Deprivation

Even with a large number of particles, it may happen that there are no particles in the vicinity of the correct state. This is known as the *particle deprivation* problem. It occurs mostly when the number of particles is too small to cover all relevant regions with high likelihood. However, this ultimately can happen in any particle filter, regardless of the particle set size M.

Particle deprivation is a consequence of the variance in random sampling. If we run the particle filter long enough, it will eventually generate an estimate that is arbitrarily incorrect. In practice, problems of this nature only tend to arise when M is small relative to the space of all states with high likelihood.

The quality of the sample based representation increases with the number of samples. An important question is therefore how many samples should be used for a specific estimation problem. Unfortunately, there is no perfect answer and it is often left to the user to determine the required number of samples. As a rule of thumb, the number of samples strongly depends on the dimensionality of the state space and the uncertainty of the distributions approximated by the particle filter.

4.2.4 Resampling Methods

We only consider two resampling methods. The sequential resampling, which is explained in [149], and the stratified resampling. Both resampling techniques yield similar results. The authors of FastSLAM propose the sequential resampler [93, 149], although they recommend the stratified resampling method for tracking multiple, distinct hypotheses, which is mostly the case of the SLAM problem.

Sequential Resampling

SEQUENTIAL RESAMPLING

The *sequential resampling* method proposed by Madow [85, 86, 87] is an efficient probabilistic sampling method easy to implement. It is alos known as *systematic resampling* and *low variance sampling* [149]. The resampling method, when applied to the particle filter, combines the systematic resampling with the sampling based on the weights of each particle. Sequential resampling performs a systematic sampling in the space of probabilities and seeks the a particle with an accumulated probability greater than the sampled probability.

Algorithm 6 depicts an implementation of the sequential resampling algorithm. The basic idea is that instead of selecting samples independently of each other in the resampling process, the selection involves a sequential stochastic process. This algorithm computes a single random number and selects samples according to this number but still with a probability proportional to the sample weight. This is achieved by drawing a random number r in the interval $[0, M^{-1}]$, where M is the number of samples to be drawn at time t. Then particles are selected by repeatedly adding the fixed amount M^{-1} to r and by choosing the particle that corresponds to the resulting number. Any number $U = r + (m-1)M^{-1}$ in $[0, 1]$ points to exactly one particle, namely the particle i for which

$$i = \arg\min_{j} \sum_{k=1}^{j} w_t^{[k]} \geq U \tag{4.22}$$

The while loop in Algorithm 6 serves two taks, it computes the sum in the right-hand side of this equation and additionally checks wheter i is the index of the first particle such that the corresponding sum of weights exceeds U. This process is also illustrated in Figure 4.2.

The advantage of the sequential resampler is threefold. First, it covers the space of samples in a more systematic fashion that the independent random sampler. Second, if all samples have the same importance factors, the resulting sample set $\bar{\mathcal{X}}_t$ is equivalent to $\mathcal{X}_t$. Third, the low-variance sampler has a complexity of $\mathcal{O}(M)$.

Algorithm 6 Sequential Resampling

Require: Particle set $\mathcal{X}_t$ with M elements and associated weights $\mathcal{W}_t$.
An uniform random number generator **rand**(a, b) that generates a random number $r \in [a, b]$ is used to sample from $\mathcal{X}_t$.

Ensure: Sequential, systematic or low variance resampling left the sampled particles in the set $\bar{\mathcal{X}}_t$ such that the probability of a particle to be resampled is proportional to its weight.
Sampling M particles requires $\mathcal{O}(M)$ time.

Algorithm: Sequential$(\mathcal{X}_t, \mathcal{W}_t)$ **return** $\bar{\mathcal{X}}_t$

1: $\bar{\mathcal{X}}_t = \varnothing$
2: $r = \text{rand}(0, M^{-1})$
3: $i = 1$
4: $c = w_t^{[i]}$
5: **for all** particle $x_t \in \mathcal{X}_t$ **do**
6: **while** $r > c$ **do**
7: $i = i + 1$
8: $c = c + w_t^{[i]}$
9: **end while**
10: add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$
11: $r = r + M^{-1}$
12: **end for**
13: **return** $\bar{\mathcal{X}}_t$

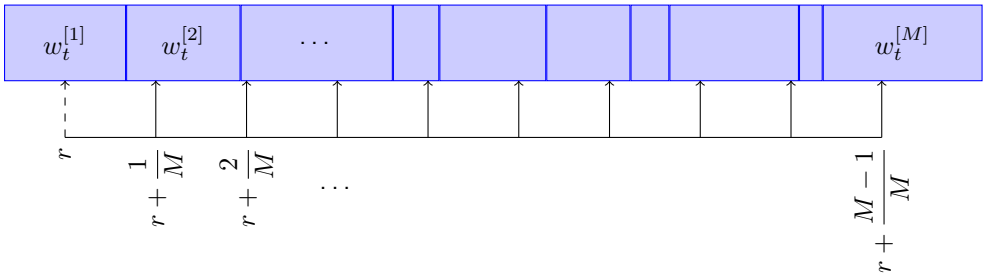


Figure 4.2: Sequential Resampler

Computation time is of essence when using particle filters, and often an efficient implementation of the resampling process makes a huge diffrence in the practical performance.

Stratified Resampling

STRATIFIED RESAMPLING

Another popular option is *stratified resampling*, in which particles are grouped into subsets. Sampling from these sets is performed in a two stage procedure, as show in Algorithm 7. First, the number of samples drawn from each subset is determined based on the total weight of the particles contained in the subset. In the second stage, individual samples are drawn randomly from each subset using, for example, low variance resampling or simply random sampling.

Stratified resampling has lower sampling variance and tends to perform well

Algorithm 7 Stratified Resampling

Require: Particle set $\mathcal{X}_t$ with M elements and associated weights $\mathcal{W}_t$.
An uniform random number generator **rand**(a, b) that generates a random number $r \in [a, b]$ is used to sample from $\mathcal{X}_t$.

Ensure: Stratified or residual resampling left the sampled particles in the set $\bar{\mathcal{X}}_t$ such that the probability of a particle to be resampled is proportional to its weight within a stratum.
Sampling M particles requires $\mathcal{O}(M)$ time.

Algorithm: Stratified$(\mathcal{X}_t, \mathcal{W}_t)$ **return** $\bar{\mathcal{X}}_t$

$k = \frac{1}{M}$ ▷ stratum separation
$d_1 = \frac{k}{2}$ ▷ first stratum end
for $i = 2$ **to** M **do** ▷ construct rest of stratum set
  $d_i = d_{i-1} + k$
end for
for $i = 1$ **to** M **do** ▷ compute stratified random variables
  $u_i = d_i +$ **rand**$(-d_1, d_1)$
end for

$c = 1$ ▷ index of current stratified random variable
for $i = 1$ **to** M **do**
  while $u_c < \sum_{m=1}^{i} w_t^{[m]}$ **do**
    add $x_t^{[i]}$ to $\bar{\mathcal{X}}_t$
    $c = c + 1$ ▷ step to next stratified random variable
  end while
end for
return $\bar{\mathcal{X}}_t$

when a robot tracks multiple, distinct hypotheses with a single particle filter [149]. This is precisely the case of the FastSLAM algorithm discussed in the present work. Nevertheless, the results obtained with both sequential and stratified are very similar in practice.

Chapter 5

Robot Motion

5.1 Introduction

This and the next chapter describe the two remaining components for implementing the filter algorithms described thus far: the motion and the measurement models. This chapter focuses on the motion model. *Motion models* comprise the state transition probability $p(x_t \mid x_{t-1}, u_t)$, which plays an essential role in the prediction step of the Bayes filter. This chapter provides in-depth examples of probabilistic motion models as they are being used in actual robotics implementations. The subsequent chapter will describe probabilistic models of sensor measurements $p(z_t \mid x_t)$, which are essential for the measurement update step. The material presented here will be essential for implementing any of the algorithms described in subsequent chapters, and it has been taken from [149], which is recommeded as a complementary reading.

MOTION MODELS

5.2 Kinematics

Robot kinematics, which is the central topic of this chapter, has been studied thoroughly in past decades. However, it has almost exclusively been addressed in deterministic form. Probabilistic robotics generalizes kinematic equations to the fact that the outcome of a control is uncertain, due to control noise or unmodeled exogenous effects. The outcome of a control will be described by a posterior probability. In doing so, the resulting models will be amenable to the probabilistic state estimation techniques described in the previous chapters.

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. No model of manipulator kinematics will be provided, neither will be discuss models of robot dynamics. However, this restricted choice of material is by no means to be interpreted that probabilistic ideas are limited to simple kinematic models of mobile robots. Rather, it is descriptive of the present state of the art, as probabilistic techniques have enjoyed their biggest successes in mobile robotics using relatively basic models of the types described in this chapter. The use of more sophisticated probabilistic models —e.g. probabilistic models of robot dynamics— remains largely unexplored in the literature. Such extensions, however, are not infeasible. As this chapter illustrates, deterministic robot actuator

models are "probilified" by adding noise variables that characterize the types of uncertainty that exist in robotic actuation.

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 uncertainty outcomes are provided in the first place. In fact, many of the models that have proven most successful in practical applications vastly overestimate the amount of uncertainty. By doing so, the resulting algorithms are more robust to violations of the Markov assumptions, such as unmodeled state and the effect of algorithmic approximations (see Section 2.4.1). We will point out such findings in later chapters, when discussing actual implementations of probabilisitc robotic algorithms.

5.2.1 Kinematic Configuration

KINEMATICS

Kinematics is the calculus describing the effect of control actions on the configuration of a robot. The *configuration* of a rigid mobile robot operating in planar environments is described by three variables that represent the kinematic state, referred to as pose in this text. As a consequence of this restriction, the material of this thesis is restricted to two dimensions.

CONFIGURATION

POSE

The *pose* of a mobile robot operating in a plane is illustrated in Figure 5.1. It comprises its two-dimensional planar coordinates relative to an external coordinate frame, along with its angular orientation. Denoting the former as x and y, and the latter by θ, the pose of the robot is described by the vector

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{5.1}$$

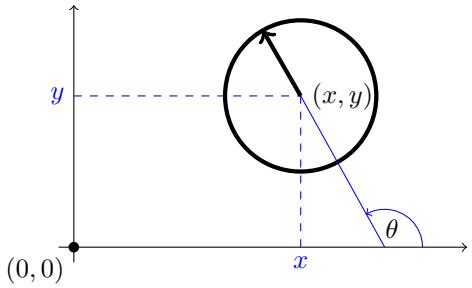


Figure 5.1: Robot Kinematic Configuration, where pose $x_t = (x, y, \theta)^T$

ORIENTATION

The *orientation* of a robot is often called *bearing* or *heading direction*. As shown in Figure 5.1, we postulate that a robot with orientation $\theta = 0$ points into the direction of its x-axis, while a robot with orientation $\theta = \frac{\pi}{2}$ points into the direction of its y-axis.

LOCATION

Pose without orientation will be called *location*. The concept of location will be important in which refer to xy coordinates of an object:

$$\begin{pmatrix} x \\ y \end{pmatrix} \tag{5.2}$$

The pose and the locations of objects in the environment may constitute the kinematic state x_t of the robot-environment system.

5.2.2 Motion Model

The probabilistic kinematic model or *motion model* plays the role of the state transition model in mobile robotics. This model is the familiar conditional density MOTION MODEL

$$p(x_t \mid x_{t-1}, u_t) \tag{5.3}$$

Here x_t and x_{t-1} are both robot poses and u_t is a motion command or *control*. This model describes the posterior distribution over kinematic states that a robot assumes when executing the control u_t at x_{t-1}. For conceptual reasons we will refer to u_t as control, although it is sometimes provided by the robot's odometry. CONTROL

This chapter provides in detail two specific probabilistic motion models $p(x_t \mid x_{t-1}, u_t)$, both for mobile robots operating in the plane but complementary in the type of motion information that is being processed. The first assumes that the control u_t specifies the velocity commands given to the robot's motors, while the second model assumes that one has access to odometry information, resulting a probabilistic model somewhat different from the velocity one.

In practice, odometry models tend to be more accurate than velocity models, for the simple reason that most commercial robots do not execute velocity commands with the level of accuracy that can be obtained by measuring the revolution of the robot's wheels. However, odometry is only available after executing a motion command. Thus, odometry models are usually applied for estimation, whereas velocity models are used for probabilistic motion planning, which is outside the scope of this thesis.

5.3 Velocity Motion Model

The *velocity motion model* assumes that we can control a robot through two velocities, a rotational and a translational velocity.

We will denote the *translational velocity* at time t by v, and the *rotational velocity* by ω. Hence, we have the control vector

$$u_t = \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{5.4}$$

We arbitrarily postulate that positive rotational velocities ω induce a counterclockwise rotation $\circlearrowleft$. Positive translational velocities v correspond to forward motion.

For kinematic state estimation, it suffices to sample from the motion model $p(x_t \mid x_{t-1}, u_t)$ instead of computing the posterior for arbitrary x_t, u_t and x_{t-1}. *Sampling* from a conditional density is different than calculating the density. In sampling, one is given u_t and x_{t-1} and seeks to generate a random x_t drawn according to the motion model $p(x_t \mid x_{t-1}, u_t)$. When calculating the density, one is also given x_t generated through other means, and one seeks to compute the probability of x_t under $p(x_t \mid x_{t-1}, u_t)$.

The Algorithm 8 generates random samples from $p(x_t \mid x_{t-1}, u_t)$ for a fixed control u_t and pose x_{t-1}. It accepts $x_{t-1} = (x \ y \ \theta)^T$ and $u_t = (v \ \omega)^T$ as

Algorithm 8 Velocity Motion Model

Require: Pose $x_{t-1} = (x\ y\ \theta)^T$ and control $u_t = (v\ \omega)^T$.
Motion noise parameters $\alpha_1, \ldots, \alpha_6$ and function **sample**(σ^2) that generates a random sample from a zero-centered distribution with variance σ^2.
Ensure: Sample pose $x_t = (x'\ y'\ \theta')^T$ from $p(x_t \mid u_t, x_{t-1})$ applying a pure velocity model without odometry information.
Final orientation is perturbed by an additional random term $\hat{\gamma}$.
Algorithm: VelocityMotionModel$(u_t,\ x_{t-1})$ **return** x_t

1: $\hat{v} = v + \textbf{sample}(\alpha_1 v^2 + \alpha_2 \omega^2)$
2: $\hat{\omega} = \omega + \textbf{sample}(\alpha_3 v^2 + \alpha_4 \omega^2)$
3: $\hat{\gamma} = \textbf{sample}(\alpha_5 v^2 + \alpha_6 \omega^2)$
4: $x' = x + \frac{\hat{v}}{\hat{\omega}}\Big(\sin(\theta + \hat{\omega}\Delta t) - \sin\theta\Big)$
5: $y' = y + \frac{\hat{v}}{\hat{\omega}}\Big(\cos\theta - \cos(\theta + \hat{\omega}\Delta t)\Big)$
6: $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$
7: **return** $x_t = (x', y', \theta')^T$

input and generates a random pose $x_t = (x'\ y'\ \omega')^T$ according to the distribution $p(x_t \mid x_{t-1}, u_t)$. Line 1 through 3 "perturb" the commanded control parameters by noise drawn from the error parameters of the kinematic motion model. The noise values are then used to generate the sample's new pose in lines 4 through 7. Thus, the samping procedure implements a simple physical robot motion model that incorporates control noise in its prediction, in just about the most straightforward way.

A null rotational velocity $\hat{\omega} = 0$ produces a *division-by-zero* indetermination when computing x' and y' at lines 4 and 5. In this case it is safe to take the limit when $\hat{\omega} \to 0$ for the problematic expressions

$$\lim_{\hat{\omega}\to 0} \frac{\hat{v}}{\hat{\omega}}\Big(\sin(\theta + \hat{\omega}t) - \sin\theta\Big) \overset{\text{l'Hôpital}}{=} t\hat{v}\cos\theta \tag{5.5}$$

$$\lim_{\hat{\omega}\to 0} \frac{\hat{v}}{\hat{\omega}}\Big(\cos\theta - \cos(\theta + \hat{\omega}t)\Big) \overset{\text{l'Hôpital}}{=} t\hat{v}\sin\theta \tag{5.6}$$

5.3.1 Mathematical Derivation

We will now derive the Algorithm 8. As usual, the reader not interested in the mathematical details is invited to skip this section at first reading. The derivation begins with a generative model of robot motion, and then derives formulae for sampling $p(x_t \mid x_{t-1}, u_t)$ for arbitrary x_t, u_t and x_{t-1}.

Exact Motion

Before turning to the probabilistic case, let us begin by stating the kinemtics for an ideal, noise-free robot. Let $u_t = (v\ \omega)^T$ denote the control at time t. If both velocities are kept at a fixed value for the entire time interval $(t-1, t]$, the robot

moves on a circle with radius

$$r = \left|\frac{v}{\omega}\right| \tag{5.7}$$

This follows from the general relationship between the translational and rotational velocities v and ω for an arbitrary object moving on a circular trajectory with radius r,

$$v = \omega \cdot r \tag{5.8}$$

Equation (5.7) encompasses the case where the robot does no turn at all ($\omega = 0$), in which case the robot moves on a straight line. A straight line corresponds to a circle with infinite radius, hence we note thar r may be infinite.

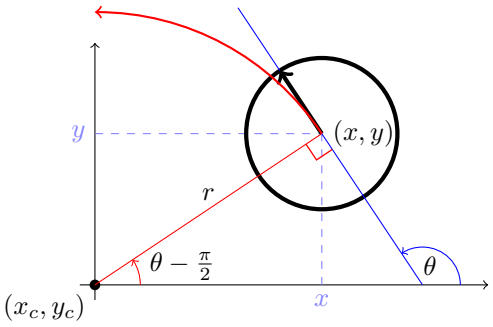


Figure 5.2: Exact Motion

Let $x_{t-1} = (x\ y\ \theta)^T$ be the initial pose of the robot, and suppose we keep the velocity constant at $(v\ \omega)^T$ for some time Δt. As one easily shows, the center of the circle is at

$$x_c = x - \frac{v}{\omega}\sin\theta \tag{5.9}$$

$$y_c = y - \frac{v}{\omega}\cos\theta \tag{5.10}$$

The variables $(x_c\ y_c)^T$ denote this coordinate. After Δt time of motion, our ideal robot depicted in Figure 5.2 will be at $x_t = (x'\ y'\ \theta')^T$ with

$$\begin{aligned}\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} &= \begin{pmatrix} x_c + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ y_c - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \theta + \omega\Delta t \end{pmatrix} \\ &= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{v}{\omega}\sin\theta + \frac{v}{\omega}\sin(\theta + \omega\Delta t) \\ \frac{v}{\omega}\cos\theta - \frac{v}{\omega}\cos(\theta + \omega\Delta t) \\ \omega\Delta t \end{pmatrix} \\ &= \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \frac{v}{\omega}\Big(\sin(\theta + \omega\Delta t) - \sin\theta\Big) \\ \frac{v}{\omega}\Big(\cos\theta - \cos(\theta + \omega\Delta t)\Big) \\ \omega\Delta t \end{pmatrix}\end{aligned} \tag{5.11}$$

The derivation of this expression follows from simple trigonometry. After Δt units of time, the noise-free robot has progressed $v \cdot \Delta t$ along the circle, which caused its heading direction to turn by $\omega \cdot \Delta t$. At the same time, its x and y coordinate is given by the intersection of the circle about $(x_c\ y_c)^T$ and the ray starting at $(x_c\ y_c)^T$ at the angle perpendicular to $\omega \cdot \Delta t$. The second transformation simply substitutes (5.9) and (5.10) into the resulting motion equations.

Of course, real robots cannot jump from onew velocity to another and keep velocity constant in each time interval. To compute the kinematics with non-constant velocities, it is therefore common practice to use small values for Δt and to approximate the actual velocity by a constant within each time interval. The approximate final pose is then obtained by concatenating the corresponding cyclic trajectories using the mathematical equations just stated.

Real Motion

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

$$\begin{pmatrix} \hat{v} \\ \hat{\omega} \end{pmatrix} = \begin{pmatrix} v \\ \omega \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1 v^2 + \alpha_2 \omega^2} \\ \varepsilon_{\alpha_3 v^2 + \alpha_4 \omega^2} \end{pmatrix} \tag{5.12}$$

Here ε_{σ^2} is a zero-mean error variable with variance σ^2. Thus, the true velocity equals the commanded velocity plus some small, additive error (noise). In our model, the standard deviation of the error is proportional to the commanded velocity. The parameters α_1 to α_4 (with $\alpha_i \geq 0$ for $i = 1, \ldots, 4$) are robot-specific error parameters. They model the accuracy of the robot. The less accurate a robot, the larger these parameters.

Two common choices for the error ε_{σ^2} are the normal and the triangular distribution. The reader interested in the density function and the sampling process over such distributions is invited to see Appendix A, which treats the details of the sampling processes beyond random variables drawn from different distributions.

A better model of the actual pose $x_t = (x'\ y'\ \theta')^T$ after executing the motion command $u_t = (v\ \omega)^T$ at $x_{t-1} = (x\ y\ \theta)^T$ is thus

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \frac{\hat{v}}{\hat{\omega}} \Big(\sin(\theta + \hat{\omega}\Delta t) - \sin\theta \Big) \\ \frac{\hat{v}}{\hat{\omega}} \Big(\cos\theta - \cos(\theta + \hat{\omega}\Delta t) \Big) \\ \hat{\omega}\Delta t \end{pmatrix} \tag{5.13}$$

This equation is obtained by substituting the commanded velocity $u_t = (v\ \omega)^T$ with the noisy motion $(\hat{v}\ \hat{\omega})^T$ in (5.11). However, this model is still not very realistic, for reasons discussed in turn.

Final Orientation

The two equations given above exactly describe the final location of the robot given that the robot actually moves on an exact circular trajectory with radius $r = \dfrac{\hat{v}}{\hat{\omega}}$.

While the radius of this circular segment and the distance traveled is influenced by the control noise, the very fact that the trajectory is circular is not. The assumption of circular motion leads to an important degeneracy. In particular, the support of the density $p(x_t \mid x_{t-1}, u_t)$ is two-dimensional, within a three-dimensional embedding pose space. The fact that all posterior poses are located on a two-dimensional manifold within the three-dimensional pose space is a direct consequence of the fact thar we used only two noise variables, one for v and one for ω. Unfortunately, this degeneracy has important ramifications when applying Bayes filters for state estimation.

In reality, any meaningful posterior distribution is of course not degenerate, and poses can be found within a three-dimensional space of variations in x, y and θ. To generalize our motion model accordingly, we will assume that the robot performs a rotation $\hat{\gamma}$ when it arrives at its final pose. Thus, instead of computing θ' according to (5.13), we model the final orientation by

$$\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t \tag{5.14}$$

with

$$\hat{\gamma} = \varepsilon_{\alpha_5 v^2 + \alpha_6 \omega^2} \tag{5.15}$$

Here α_5 and α_6 are additional robot-specific parameters that determine the variance of the additional rotational noise. Thus, the resulting motion model is as follows,

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \frac{\hat{v}}{\hat{\omega}}\Big(\sin(\theta + \hat{\omega}\Delta t) - \sin\theta\Big) \\ \frac{\hat{v}}{\hat{\omega}}\Big(\cos\theta - \cos(\theta + \hat{\omega}\Delta t)\Big) \\ \hat{\omega}\Delta t + \hat{\gamma}\Delta t \end{pmatrix} \tag{5.16}$$

5.4 Odometry Motion Model

The velocity model discussed thus far uses the robot's velocity to compute posteriors over poses. Alternatively, one 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. This lead to a second motion model, the *odometry motion model*, which uses odometry measurements in lieu of controls.

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 crude mathematical model.

Technically, odometric information are sensor measurements, not controls. To model odometry as measurements, the resulting Bayes filter would have to include the actual velocity as state variables —which increases the dimension of the state space. To keep the state space small, it is therefore common to consider odometry data as if it were control signals. In this section, we will treat odometry measurements just like controls. The resulting model is at the core of many of today's best probabilistic robot systems.

Let us define the format of our control information. At time t, the correct pose of the robot is modeled by the random variable x_t. The robot odometry estimates this pose; however, due to drift and slippage there is no fixed coordinate

transformation between the coordinates used by the robot's internal odometry and the physical world coordinates. In fact, knowing this transformation would solve the robot localization problem.

The odometry model uses the *relative motion information*, as measured by the robot's internal odometry. More specifically, in the time interval $(t-1, t]$, the robot advances from a pose x_{t-1} to pose x_t. The odometry reports back to us a related advance from $\bar{x}_{t-1} = (\bar{x}\ \bar{y}\ \bar{\theta})^T$ to $\bar{x}_t = (\bar{x}'\ \bar{y}'\ \bar{\theta}')^T$. Here the bar indicates that these are odometry measurements embedded in a robot internal coordinate whose relation to the global world coordinates is unknown. The key insight for utilizing this information in state estimation is that the relative difference between $\bar{x}_{t-1}$ and $\bar{x}_t$, under and appropiate definition of the term "difference", is a good estimator for the difference of the true poses $\bar{x}_{t-1}$ and $\bar{x}_t$. The motion information u_t is, thus, given by the pair

$$u_t = \begin{pmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{pmatrix} \tag{5.17}$$

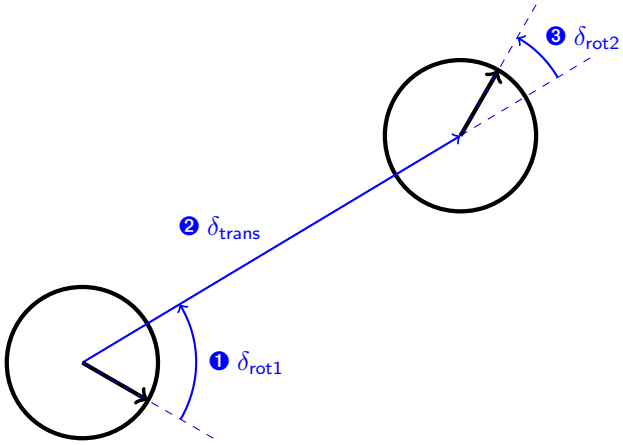


Figure 5.3: Odometry Model

To extract relative odometry, u_t is transformed into a sequence of three steps: ❶ a rotation, followed by ❷ a straight line motion (translation), and ❸ another rotation. Figure 5.3 illustrates this decomposition: ❶ the initial turn is called δ_{rot1}, ❷ the translation δ_{trans}, and ❸ the second rotation δ_{rot2}. Each pair of positions $(\bar{s}\ \bar{s}')^T$ has a unique parameter vector $(\delta_{\text{rot1}}\ \delta_{\text{trans}}\ \delta_{\text{rot2}})^T$, and these parameters are sufficient to reconstruct the relative motion between $\bar{s}$ and $\bar{s}'$. Thus, δ_{rot1}, δ_{trans}, δ_{rot2} form together a sufficient statistics of the relative motion encoded by the odometry.

The probabilistic motion model assumes that these three parameters are corrupted by independent noise. The reader may note that odometry motion uses one more parameter than the velocity vector defined in the previous section, for which reason we will not face the same degeneracy that led to the definition of a "final rotation".

The kinematic state estimation requires samples of $p(x_t \mid x_{t-1}, u_t)$ rather than a closed-form expression for computing $p(x_t \mid x_{t-1}, u_t)$ for any x_{t-1}, u_t and x_t. The Algorithm 9 accepts an initial pose x_{t-1} and an odometry reading u_t as input,

Algorithm 9 Odometry Motion Model

Require: Pose $x_{t-1} = (x\ y\ \theta)^T$ and control $u_t = (\bar{x}_{t-1}\ \bar{x}_t)^T$ with $\bar{x}_{t-1} = \left(\bar{x}\ \bar{y}\ \bar{\theta}\right)^T$ and $\bar{x}_t = \left(\bar{x}'\ \bar{y}'\ \bar{\theta}'\right)^T$, that is a differentiable set of two pose estimates obtained by the robot's odometer.
Motion noise parameters $\alpha_1, \ldots, \alpha_4$ and function **sample**(σ^2) that generates a random sample from a zero-centered distribution with variance σ^2.

Ensure: Sample pose $x_t = (x'\ y'\ \theta')^T$ from $p(x_t \mid u_t, x_{t-1})$ using odometry information.

Algorithm: OdometryMotionModel$(u_t,\ x_{t-1})$ **return** x_t

1: $\delta_{\text{rot1}} = \operatorname{atan2}\left(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}\right) + \bar{\theta}$
2: $\delta_{\text{trans}} = \sqrt{\left(\bar{x} - \bar{x}'\right)^2 + \left(\bar{y} - \bar{y}'\right)^2}$
3: $\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$

4: $\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1\delta^2_{\text{rot1}} + \alpha_2\delta^2_{\text{trans}})$
5: $\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3\delta^2_{\text{trans}} + \alpha_4\delta^2_{\text{rot1}} + \alpha_4\delta^2_{\text{rot2}})$
6: $\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1\delta^2_{\text{rot2}} + \alpha_2\delta^2_{\text{trans}})$

7: $x' = x + \hat{\delta}_{\text{trans}} \cos\left(\theta + \hat{\delta}_{\text{rot1}}\right)$
8: $y' = y + \hat{\delta}_{\text{trans}} \sin\left(\theta + \hat{\delta}_{\text{rot1}}\right)$
9: $\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$

10: **return** $x_t = (x', y', \theta')^T$

and outputs a random x_t distributed according to $p(x_t \mid x_{t-1}, u_t)$. It randomly guesses a pose x_{t-1} in lines 4-6, instead of computing the probability of a given x_t.

5.4.1 Mathematical Derivation

The derivation of the algorithm is relatively straightforward, and may be skipped at first reading. 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 translation and another rotation. The following equations show how to calculate the values of the two rotations and the translation from the odometry reading $u_t = (\bar{x}_{t-1}\ \bar{x}_t)^T$ with $\bar{x}_{t-1} = (\bar{x}\ \bar{y}\ \bar{\theta})$ and $\bar{x}_t = (\bar{x}'\ \bar{y}'\ \bar{\theta}')$,

$$\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) + \bar{\theta} \tag{5.18}$$

$$\delta_{\text{trans}} = \sqrt{\left(\bar{x} - \bar{x}'\right)^2 + \left(\bar{y} - \bar{y}'\right)^2} \tag{5.19}$$

$$\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}} \tag{5.20}$$

To model the motion error we assume that the "true" values of the rotation and translation are obtained from the measured ones by substracting independent noise

ε_{σ^2} with zero mean and variance σ^2,

$$\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \varepsilon_{\alpha_1\delta^2_{\text{rot1}}+\alpha_2\delta^2_{\text{trans}}} \tag{5.21}$$

$$\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \varepsilon_{\alpha_3\delta^2_{\text{trans}}+\alpha_4\delta^2_{\text{rot1}}+\alpha_4\delta^2_{\text{rot2}}} \tag{5.22}$$

$$\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \varepsilon_{\alpha_1\delta^2_{\text{rot2}}+\alpha_2\delta^2_{\text{trans}}} \tag{5.23}$$

The parameters α_1 to α_4 are robot-specific error parameters, which specify the error accrued with motion.

Consequently, the true position x_t is obtained from x_{t-1} by an initial rotation with angle $\hat{\delta}_{\text{rot1}}$, followed by a translation with distance $\hat{\delta}_{\text{trans}}$, followed by another rotation with angle $\hat{\delta}_{\text{rot2}}$. Thus,

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot1}}) \\ \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}}) \\ \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}} \end{pmatrix} \tag{5.24}$$

Notice that Algorithm 9 implements (5.18) through (5.24).

5.5 Discussion

The odometry motion model is generally preferable to the velocity motion model. Nevertheless, both models perfom well with noisy motion, allowing SLAM algorithms to estimate the robot pose with sufficient precision. The odometry model gives better results because the odometric information retreived from the wheel encoders used to be more precise than the control commanded. Indeed, for rotational movements the odometry reduces error propagation with repect to the real robot pose.

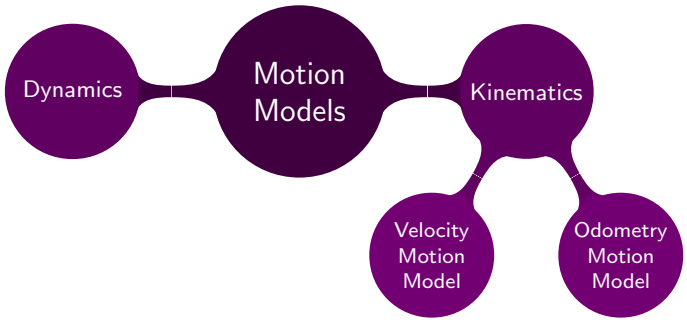


Figure 5.4: Motion Models

Alternatively, it is possible to use more sophisticated motion models that might cover the robot dynamics. Recall that the velocity and odometry motion models discussed thus far based on the robot kinamtics only, as shown in Figure 5.4. In any case, they are outside the scope of the present work. The reader interested might consult the works commented in the literature. Similarly, it is also invited to consult other works concerning robots that do not operate in a plane and have a kinematic configuration different than the one discussed in this chapter.

Chapter 6

Robot Perception

6.1 Introduction

The *Environment measurement models* comprise the second domain-specific model in probabilistic robotics, next to motion models. Measurements models describe the formation process by which sensor measurements are generated in the physical world. Today's robots use a variety of different sensor modalities, such as range sensors or cameras. The specifics of the model depends on the sensor. Hereafter we discuss the fundamentals of robot perception as described in [149]. In this work we focus in feature-based models, whose description is very detailed. Other measurement models are later skimmed to give a view of the variety of alternatives.

ENVIRONMENT MEASUREMENT MODELS

Probabilistic robotics explicitly models the noise in sensor measurements. Such models accounts for the inherent uncertainty in the robot's sensors. Formally, the measurement model is defined as a conditional probability distribution $p(z_t \mid x_t, m)$, where x_t is the robot pose, z_t is the measurement at time t and m is the map of the environment. Although we mainly address range-sensors throughout this chapter, the underlying principles and equations are not limited to this type of sensors. Instead the basic principle can be applied to any kind of sensor.

Figure 6.1 shows a typical *laser range scan*, acquired with a 2-D laser range finder. A laser actively emits a signal and records its echo. The signal is a light beam, which make lasers provide much more focused beams than other range sensors, like a *sonar*. The specific laser in Figure 6.1 is based on a time-of-flight measurement, and these are spaced in $d\theta$ increments.

LASER RANGE SCAN

SONAR

As a rule of thumb, the more accurate a sensor model, the better the results —though there are some important caveats that were already discussed in Section 2.4.1. In practice, however, it is often impossible to model a sensor accurately, primarily due to the complexity of physical phenomena.

Often, the response characteristics of a sensor depends on variables we prefer not to make explicit in a probabilistic robotics algorithm —such as the surface material of walls. Probabilistic robotics accommodates inaccuracies of sensor models in the stochastic aspects. By modeling the measurement process as a conditional probability density $p(z_t \mid x_t)$ instead of a deterministic function $z_t = f(x_t)$, the uncertainty in the sensor model can be accommodated in the non-deterministic aspects of the model. Herein lies a key advantage of probabilistic techniques over classical robotics: in practice, we can get away with extremely crude models. However, when

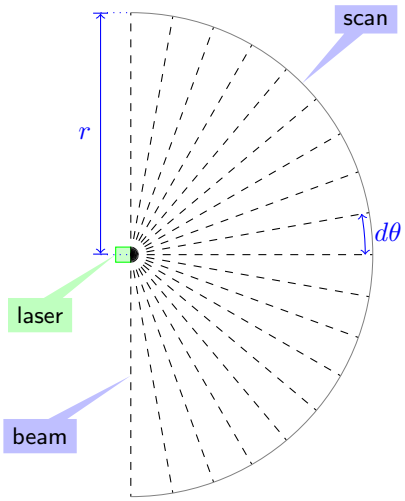


Figure 6.1: Laser range scan

devising a probabilistic model, care has to be taken to capture the different types of uncertainties that may affect a sensor measurement.

Many sensors generate more than one numerical measurement value when queried. For example, camaeras generate entire arrays of values; similarly, range finders usually generate entire scans of ranges. We will denote the number of such measurement values within a measurement z_t by K, hence we can write

$$z_t = \left\{ z_t^1, z_t^2, \ldots, z_t^K \right\} \tag{6.1}$$

We will use z_t^k to refer to an individual measurement —e.g. one range value.

The probability $p(z_t \mid x_t, m)$ is obtained as the product of the individual measurement likelihoods

$$p(z_t \mid x_t, m) = \prod_{k=1}^{K} p(z_t^k \mid x_t, m) \tag{6.2}$$

Technically, this amounts to an *independence assumption* between the noise in each individual measurement beam —just as our Markov assumption assumes independent noise over time (see Section 2.4.1). This assumption is only true in the ideal case, since possible causes of dependencies are errors in the model m, approximations in the posterior and so on. For now, we will simply not worry about violations of the independence assumption.

6.2 Maps

MAP

To express the process of generating measurements, we need to specify the environment in which a measurement is generated. A *map* of the environment is a list of objects in the environment and their locations. Formally, a map m is a list of objects in the environment along with their properties

$$m = \left\{ m_1, m_2, \ldots, m_N \right\} \tag{6.3}$$

Here N is the total number of objects in the environment, and each m_i with $1 \leq i \leq N$ specifiees a property. Maps are usually indexed in one of two ways, known as *feature-based* and *location-based*. In feature-based maps, i is a feature index. The value of m_i contains the properties of a feature and its Cartesian location. In location-based maps, the index i corresponds to a specific location. In planar maps, it is common to denote a map element by $m_{x,y}$ instead of m_i, to make explicit that $m_{x,y}$ is the property of a specific world coordinate $(x\ y)$.

Both types of maps have advantages and disadvantages. Location-based maps are *volumetric*, in that they offer a label for any location in the world. Volumetric maps contain information not only about objects in the environment, but also about the absence of objects —e.g. free-space. This is quite differente in feature-based maps, since they only specify the shape of the environment at the specific locations, namely the locations of the objects contained in the map. Feature representation makes it easier to adjust the position of an object —e.g. as a result of additional sensing. For this reason, feature-based maps are popular in the robotic mapping field, where maps are constructed from sensor data.

VOLUMETRIC

A classical map representation is known as *occupancy grid map*, which is location-based. They assign to each $(x\ y)$ coordinate a binary occupancy value that specifies whether or not a location is occupied with an object. Occupancy grid maps are great for mobile robot navigation, since they make it easy to find paths through the unoccupied space.

OCCUPANCY GRID MAP

Throughout this dissertation, we will drop the distinction between the physical world and the map. Technically, sensor measurements are caused by physical objects, not the map of those objects. However, it is tradition to condition sensor models on the map m. Hence we will adopt a notation that suggests measurements depend on the map.

6.3 Feature-Based Measurement Models

6.3.1 Feature Extraction

If we denote the feature extractor as a function f, the *features* extracted from a range measurement are given by $f(z_t)$. Most feature extractors extract a small number of features from high-dimensional sensor measurements. A key advantage of this approach is the enormous reduction of computational complexity. Therefore, inference in the low-dimensional feature space can be orders of magnitude more efficient than in the high-dimensional measurement space.

FEATURES

For range sensors, it is common to identify lines, corners or local minima in range scans, which correspond to walls, corners or objects such as tree trunks. When cameras are used for navigation, the processing of camera images falls into the realm of Computer Vision. Computer Vision has devised a myriad of feature extraction techniques from camera images. Popular features include edges, corners, distinct patterns and objects of distinct appearance. In robotics, it is also common to define places as features, such as hallways and intersections. The Chapter 7 that follows discusses some feature extraction algorithms and the features that will form the feature-based maps that the SLAM algorithms of the present work manage.

6.3.2 Landmark Measurements

Features correspond to distinct objects in the physical world. For example, in indoor environments features may be door posts or windowsills; outdoors they may correspond to tree trunks or corners of buildings. In Robotics, it is common to call those physical objects *landmarks*, to indicate that they are being used for robot navigation.

LANDMARKS

The most common model for processing landmarks assumes that the sensor can measure the range and the bearing of the landmark relative to the robot's local coordinate frame. Such sensors are called *range and bearing sensors*. The existence of a range-bearing sensor is not an implausible assumption, since any local feature extracted from range scans come with range and bearing information, as do visual features detected by stereo vision. In addition, the feature extractor may generate a *signature*. We assume a signature is a numerical value that may equally be an integer that characterizes the type of the observed landmark, or a multidimensional vector characterizing a landmark —e.g. line equation parameters.

RANGE AND BEARING SENSORS

SIGNATURE

If we denote the range by r, the bearing by ϕ, and the signature by s, the feature vector is given by a collection of triplets

$$f(z_t) = \left\{ f_t^1, f_t^2, \ldots \right\} = \left\{ \begin{pmatrix} r_t^1 \\ \phi_t^1 \\ s_t^1 \end{pmatrix}, \begin{pmatrix} r_t^2 \\ \phi_t^2 \\ s_t^2 \end{pmatrix}, \ldots \right\} \tag{6.4}$$

The number of features identified at each time step is variable. However, many probabilistic robotic algorithms assume conditional independence between features

$$p(f(z_t) \mid x_t, m) = \prod_i p(r_t^i, \phi_t^i, s_t^i \mid x_t, m) \tag{6.5}$$

Conditional independence applies if the noise in each individual measurement $(r_t^i \ \phi_t^i \ s_t^i)^T$ is independent of the noise in other measurements $(r_t^j \ \phi_t^j \ s_t^j)^T$ for $i \neq j$. Under the conditional independence assumption, we can process one feature at a time, which makes it much easier to develop algorithms that implement probabilistic measurement models.

6.3.3 Sensor Model with Known Correspondence

The measurement model requires a mechanism that establishes the correspondence between a feature f_t^i and a landmark m_j in the map. The variable c_t^i will designate the identity of f_t^i uniquely. Therefore, $j = c_t^i$ denotes that f_t^i corresponds to the landmark m_j. Given that the correspondence between features and landmarks is unknown initially, a method to build the array of correspondences c_t must be developed. Such problem is known as *data association* and it is discussed in Chapter 8. Here, we will consider that c_t have been computed in advance and we can procede to work with the sensor model, its Jacobian or the inverse model given below.

DATA ASSOCIATION

Let us now devise a sensor model for features. In Section 6.2 we distinguished between two types of maps: *feature-based* and *location-based*. Landmark measurement models are usually defined only for feature-based maps. The reader may recall that those maps consist of lists of features $m = \{m_1, m_2, \ldots\}$. Each feature may possess a signature and a *location coordinate*. The location of a feature, denoted $(m_{j,x} \ m_{j,y})^T$, is simply its coordinate in the global coordinate frame of the map.

The measurement model is different depending on the type of feature used. We consider just two types of features: points and lines. The Jacobian of the measurement model is used by recursive state estimators as the Kalman Filter described in Chapter 3. The Jacobian can be derived with respect to the landmark m_j or the pose x_t, and both might be used by FastSLAM algorithms. The inverse model allows to add new landmarks to the map, since it tranforms the feature's coordinates into the global coordinate frame of the map. The signature s_t^i is omitted in the following models because it is irrelevant actually.

Sensor Model for Point Features

The measurement vector for a noise-free landmark sensor is easily specified by the standard geometric laws. We will model noise in landmark perception by independent Gaussian noise on the range and bearing. The resulting measurement model is formulated for the case where the i-th feature at time t corresponds to the j-th landmark in the map. As usual, the robot pose is given by $x_t = (x\ y\ \theta)^T$.

$$\begin{pmatrix} r_t^i \\ \phi_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ \operatorname{atan2}(m_{j,y} - y, m_{j,x} - x) - \theta \end{pmatrix} + \begin{pmatrix} \varepsilon_{\sigma_r^2} \\ \varepsilon_{\sigma_\phi^2} \end{pmatrix} \tag{6.6}$$

where $\varepsilon_{\sigma_r^2}$ and $\varepsilon_{\sigma_\phi^2}$ are zero-mean Gaussian error variables with standard deviations σ_r and σ_ϕ.

The Jacobian with respect to the landmark m_j is

$$H_{m,j} = \begin{pmatrix} \frac{m_{j,x} - x}{\sqrt{q_t}} & \frac{m_{j,y} - y}{\sqrt{q_t}} \\ \frac{y - m_{j,y}}{q_t} & \frac{m_{j,x} - x}{q_t} \end{pmatrix} \tag{6.7}$$

where $q_t = (m_{j,x} - x)^2 + (m_{j,y} - y)^2$.

Analogously, the Jacobian with respect to the pose x_t is

$$H_{x,j} = \begin{pmatrix} \frac{x - m_{j,x}}{\sqrt{q_t}} & \frac{y - m_{j,y}}{\sqrt{q_t}} \\ \frac{m_{j,y} - y}{q_t} & \frac{x - m_{j,x}}{q_t} \end{pmatrix} \tag{6.8}$$

Curiously, for point features the Jacobian with respect to the landmark $H_{m,j}$ may be computed from the Jacobian with respect to the pose $H_{x,j}$ as follows

$$H_{x,j} = -H_{m,j} \tag{6.9}$$

and *vice versa*.

The inverse measurement model is defined for the cartesian coordinates x_t^i and y_t^i computed from the parameters r_t^i and ϕ_t^i of a feature in polar form as in (6.6). Hence

$$\begin{pmatrix} m_{j,x} \\ m_{j,y} \end{pmatrix} = \begin{pmatrix} x_t^i \cos\theta - y_t^i \sin\theta + x \\ x_t^i \sin\theta - y_t^i \cos\theta + y \end{pmatrix} \tag{6.10}$$

Sensor Model for Line Features

If we consider a line feature the measurement model changes [32]. Therefore, for the Hessian model

$$\rho = x\cos\theta + y\sin\theta \tag{6.11}$$

of a line feature defined by the parameters ρ_t^i and θ_t^i, the measurement model turns into

$$\begin{pmatrix} \rho_t^i \\ \theta_t^i \end{pmatrix} = \begin{pmatrix} m_{j,\rho} \\ m_{j,\theta} \end{pmatrix} - \begin{pmatrix} x \cos m_{j,\theta} + y \sin m_{j,\theta} \\ \theta \end{pmatrix} + \begin{pmatrix} \varepsilon_{\sigma_\rho^2} \\ \varepsilon_{\sigma_\theta^2} \end{pmatrix} \tag{6.12}$$

The Jacobian with respect to the landmark m_j is

$$H_{m,j} = \begin{pmatrix} 1 & x \sin m_{j,\theta} - y \cos m_{j,\theta} \\ 0 & 1 \end{pmatrix} \tag{6.13}$$

and with respect to the pose x_t yields

$$H_{x,j} = \begin{pmatrix} -\cos m_{j,\theta} & -\sin m_{j,\theta} \\ 0 & 0 \end{pmatrix} \tag{6.14}$$

Finally, the inverse measurement model is

$$\begin{pmatrix} m_{j,\rho} \\ m_{j,\theta} \end{pmatrix} = \begin{pmatrix} \rho_t^i \\ \theta_t^i \end{pmatrix} + \begin{pmatrix} x \cos(\theta_t^i - \theta) + y \sin(\theta_t^i - \theta) \\ \theta \end{pmatrix} \tag{6.15}$$

6.4 Other Measurement Models

In [149] some alternative measurement models are discussed in detail. Here we describe such models in brief with special attention to the advantages and drawbacks of each. The reader interested in these measurement models is invited to consult [149] and other references given below, for further reading.

6.4.1 Beam Models

BEAM MODEL

The *beam model* is an approximation of the physical model of range finders. Range finders measure the range to nearby objects. Range may be measure along a beam, which is a good model of the workings of laser range finders [149]. The model incorporates four types of measurement erros: small measurement noise, unexpected objects, failures to detect objects and random unexplained noise. The desired model $p(z_t \mid x_t, m)$ is therefore a mixture of four densities, each of which corresponds to a particular type of error. The algorithm applies ray casting to compute the noise-free range for a particular measurement and the likelihood of each individual range measurement z_t^k, which implements the mixing rule for densities [23, 97].

The various parameters of the beam sensor model are named intrinsic parameters Σ and an algorithm for adjusting these model parameters must be applied.

MAXIMUM LIKELIHOOD

Such algorithm that maximizes the likelihood of the data is known as a *Maximum Likelihood* (ML) estimator [149].

The beam-based sensor model, while closely linked to the geometry and physics of range finders, suffers two major drawbacks [149]. The model exhibits a lack of smoothness, particularly in cluttered environments with many small obstacles. As a consequence, any approximate belief might miss the correct state and methods for finding the most likely state are prone to local minima. Additionally, the model is computational involved, since evaluating $p(z_t \mid x_t, m)$ for each single sensor measurement z_t^k involves ray casting, which is computationally expensive. Fortunately, this problem can be partially remedied by *pre-caching* the ranges over a discrete grid in pose space [55].

6.4.2 Likelihood Fields

LIKELIHOOD FIELD

An alternative method called *likelihood field* overcomes some of the beam model limitations [12, 146]. It is an *ad hoc* algorithm that does not necessarily compute a conditional probability relative to any meaningful generative model of the physics of the sensors. However, the resulting posteriors are much smoother and the computation is more efficient.

The algorithm projects the end points of a sensor scan z_t into the global coordinate space of the map. The projection coordinates are only meaningful when the sensor detects an obstacle. If the range sensor takes on its maximum value $z_t^k = z_{\max}$, this measurement model simply discards such max-range readings. Three types of sources of noise and uncertainty are assumed: measurement noise, failures and unexplained random measurements. Just as for the beam-based sensor model, the probability $p(z_t \mid x_t, m)$ integrates all three distributions of noise.

A key advantage of the likelihood field model over the beam model is smoothness and that the precomputation takes place in 2D, instead of 3D, increasing the compactness of the pre-computed information. However, it has three disadvantages: it does not explicitly model dynamics that might cause short readings, it treats sensors as if they can *see through walls* and it does not take map uncertainty into account —it cannot handle unexplored areas. It is possible to extend the basic algorithm to diminish the effect of these limitations [77]. For instance, map occupancy values might be sorted into three categories: occupied, free and unknown.

6.4.3 Correlation-based Measurement Models

MAP MATCHING

In the literature there exist a number of sensor models that measure correlations between a measurement and the map [44, 123, 152]. A common method known as *map matching* transforms scans into occupancy maps [160]. It compiles small numbers of consecutive scans into local maps m_{local}. The measurement model compares m_{local} to the global map m using a map correlation function, such that the more similar m and m_{local}, the larger $p(m_{\text{local}} \mid x_t, m)$. It represents the probability of observe the local map m_{local} conditioned on the robot pose x_t and the map m, i.e. the probability that m_{local} have been observed from x_t withing the map m.

Map matching is easy to compute and it explicitly considers the free-space in the scoring of two maps, contrary to the likelihood field method that only considers the end point of the scans, which correspond to occupied space. On the other hand, map matching does not possess a plausible physical explanation, since correlations are the normalized quadratic distance between maps, which is *not* the noise characteristic of range sensors [149].

6.5 Discussion

The measurement models discussed thus far are illustrated in the mindmap of Figure 6.2. These models might be improved to represent accurately the physics of the measurement sensor or reduce the computational cost of the algorithm. There is a number of works of interst in the literature for each of them, as the reader might see in Section 6.6.

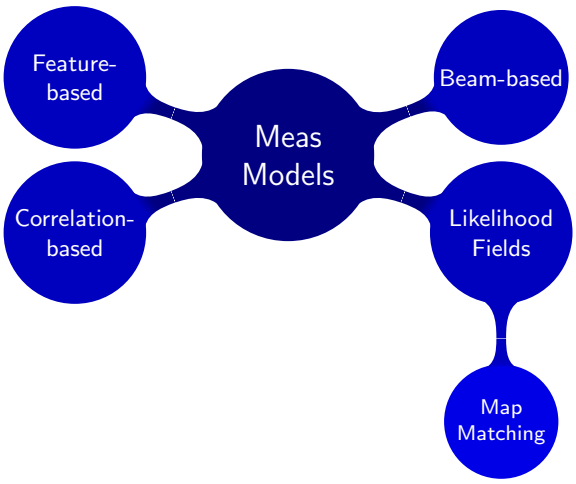


Figure 6.2: Measurement Models

In the present work we only use the feature-based measurement model devised for both point and line features in Section 6.3.3 because we will employ a featured-based SLAM algorithm fed by feature extraction methods. Therefore, the measurement models, their Jacobians and inverse models are at the core of the estimation techniques of the SLAM method. Furthermore, feature extraction methods simplify and modularize the problem, giving a computationally efficient solution that suits with the real time requirements demanded. The subsequent chapter is concerned with such methods.

6.6 Bibliographical and Historical Remarks

The contents of this chapter is essentially taken from [149], whose models are extremely crude relative to the models of laser range finders described in other works [116, 163], according to theyself. Nevertheless, those models are outside the scope of this thesis, that discusses feature-based measurement models in detail and describes other alternative models briefly, since they will not be employed in the present work. The feature-based model presented here includes the Jacobians and inverse models for both point and line feature, using the model proposed by Crowley in [32] for the former. Such models are commonplace in the SLAM literature, mostly for point landmarks [80].

An early work on beam models for range sensors can be found in a work of Moravec [97]. Fox et al. described in [55] a beam-based model with pre-caching of range measurement like the one summarized here. The likelihood fields were first published by Thrun [146] notwithstanding that they are related to the rich literature on scan matching [12]. Schiele and Crowley present a comparison of different models including correlation-based approaches [123]. The robustness of the map matching variant is analyzed by Yamauchi and Langley in [160], while Ducket and Nehmzow transform local occupancy grids into histograms that can be matched more efficiently [44].

Chapter 7

Feature Extraction

7.1 Introduction

In both localization and mapping, the measurement data or observations must be put in correspondence with the information stored in the map of the environment, a process known as *data association*. There are two techniques commonly employed to achieve this matching process: point and line based. The data association based on points works directly in the space of sensor raw data —e.g. laser scan points, image pixels. The data association based on landmarks or key *features*, which are obtained or extracted through the transformation of raw data into geometric features. The extracted features are used to solve the data association problem. This alternative is more compact, it demands lower storage and computational resources, and provides rich and precise information [4, 17, 30, 102, 113, 122, 127].

DATA ASSOCIATION

FEATURES

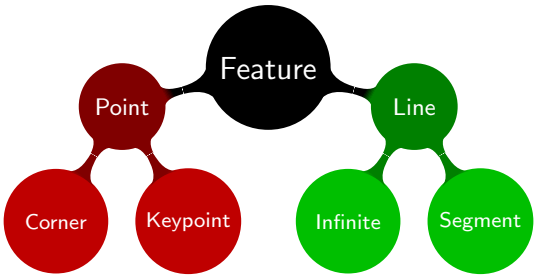


Figure 7.1: Common landmarks for feature-based SLAM

There exists many types of geometric primitives that might be used as features for mapping. The *infinite line* is meant to be the most simple one [102]. It is possible to describe a large number of interior environments using infinite lines only. Furthermore, *corner* detection from the extracted lines is straightforward and enrich the environment representation at a low computational cost.

INFINITE LINE

CORNER

Figure 7.1 shows those and other common features used in robotic mapping. Corners are just a kind of point feature that relies on the lines extracted, but there are algorithms that extract *keypoints* directly. That is the case of feature detectors as Harris [67], SIFT [82] and SURF [11], which provide a set of keypoints from an image —typically a frame retrieved by a camera onboard. It is also possible to work

KEYPOINTS

SEGMENTS with more complex geometric primitives, like line *segments* [26, 27, 90] or even curves such as circles and ellipses [57]. The data association problem is slightly more difficult with such features, but the resulting map is more representative of the environment and might be more accurate.

The ideal feature does not only depend on the data association, the extraction process must be taken into account too. The feature extraction process is tightly related with the raw data representation provided by the measurement sensor. A camera and a range laser are different by far and even more, all cameras and lasers does not provide the same format or volume of data. The present work is focused RANGE LASERS in *range lasers* exclusively. More precisely, we will manage a frontal laser of $180°$ side of view, as shown in Figure 7.2. The following feature extraction algorithms make the most of this kind of data, although the same strategies might be applied to data sampled from other sensors. Indeed, many of the algorithms shown here has been usually taken and adapted from those of the Computer Vision literature —e.g. Split & Merge [102, 109].

7.2 Line Extraction

There are many algorithms to extract lines from range laser scans. Most common LINE EXTRACTORS *line extractors* are compared in [102, 122, 127]. All of them are conceptually simple, easy to implement and are suitable to manage range laser scan data, which is particularly dense and precise. Finally, corners may be detected with a test of *real* intersections applied to all possible pairs of lines extracted. Such a corner extraction algorithm relies on a line extractor and it is analyzed in Section 7.4.

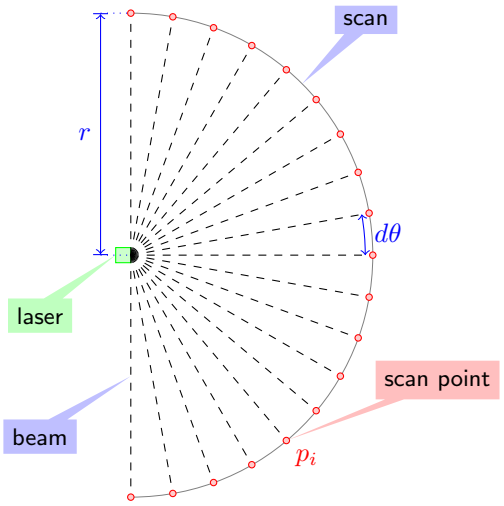


Figure 7.2: Scan points

A range laser scan can be seen as a set of n points $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$. Scan points are typically in polar form $p = (\rho\ \theta)^T$ with range ρ and bearing θ, but they are easily transformed into cartesian form $p = (x\ y)^T$ knowing the laser specifications and the acquisition configuration. The laser configuration usually depends on just two parameters, the range r and the resolution $d\theta$, as shown in Figure 7.2. The reader might recall this laser scan representation from Chapter 6, as it is also the basis of

the measurement model. For simplicity, we will manage both point representations indistinctly depending on the line extractor algorithm, because they are thought to work with a particular representation.

The line extraction problem can be subdivided into two subproblems. A *segmentation* problem that detects and separates the different lines observed —it establishes which points contribute to each line. And a *fitting* problem that adjust the parameters of a line model to obtain the best fit —it establishes how the points contribute to each line. This subproblems are discussed separately in the following sections.

Line extraction algorithms receive a list of scan points $\mathcal{P}$ and output a list of lines $\mathcal{L}$, that might be empty. Depending on the algorithm, it is possible to setup some specific parameters. In some cases, a goodness factor can be obtained to measure the quality or certainty of the lines extracted. However, for clarity reasons this factor will not be shown in the following algorithms.

The line representation is a key aspect, such as the linear fitting method used to estimate the line parameters from the set of points that presumably form it. It is crucial to use a consistent representation, one such there is no singularity or impediment to represent a particular case. A common line representation is the *Hessian model* [3] or *polar form* $\rho = x\cos\theta + y\sin\theta$ because it is consistent and compact, since only two parameters $(\rho\ \theta)^T$ are required. Furthermore, it simplifies most geometric computations —e.g. line-line intersection, point-line distance—, allowing an efficient implementation. Other line representations, like the *Intercept-Slope model* or *cartesian form* $y = mx + b$, cannot represent properly vertical lines because they suffer of a singularity —i.e. $m \to \infty$ and b does not even exists.

HESSIAN MODEL

INTERCEPT-SLOPE MODEL

Thus, the polar form has been adopted to both represent lines and compute the linear fitting parameters directly. Hence, there is no singularity or loss of precision when fitting a set of points to a line. The reader may want to consult the details of line geometry discussed in Appendix C, while Section 7.3 is concerned with linear fitting, since it is a key element of the line extraction process.

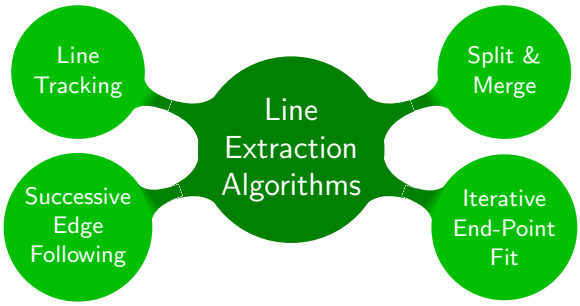


Figure 7.3: Line Extraction Algorithms

The Figure 7.3 shows the most commonly used line extraction algorithms for range lasers. This list covers classic to modern methods, that are widely used with such sensors. Although the goal and input data are shared by all theses algorithms, their operation and parametrization vary significantly, as shown in the following sections with a brief description of some of them. For more detailed information, the reader is referred to some comparative articles written by A. Siadat [127], W. Burgard [122] and R. Siegwart [102].

7.2.1 Line Tracking

LINE TRACKING

INCREMENTAL

One of the first line extraction algorithms for range laser scans was coined *Line Tracking* (LT) because it tries to track the line that best fit the subset of points from the initial p_1 to the current point p_{i-1} [127]; it is also known as *Incremental* algorithm [53, 140, 156]. It does not need all scan points at first, since it fits a running line with the leading points $p_1, \ldots, p_{i-1}$ of the scan. If the next point p_i is lower than a threshold $d_{\max}$, p_i is added to the line, otherwise a new line is began. Once a new point is added, the line parameters are fit again. Figure 7.4 illustrates this process, showing how a new scan point is added to the running line or a new line is started, for $p_i = p_5$.

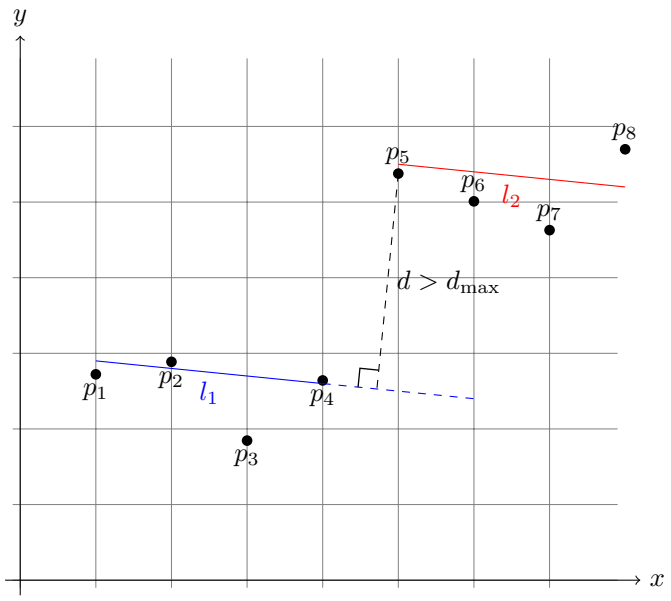


Figure 7.4: Line Tracking

The most costly step of the Line Tracking algorithm is the linear regression or fitting process, which is commonly $\mathcal{O}(n)$, where n is the number of points to fit. As the linear fitting is applied for all scan points, the total algorithm computational cost is $\mathcal{O}(n^2)$ in the worst case. Fortunately, on average n is small because it drops to 2 when a new line is started. Logically, the running line must contain at least $n = 2$ points, as done in Algorithm 10, although a greater value might result in better performance and accuracy.

Having the running line l parameters that better fit a subset of points, the distance d of the next point p_i to l is computed in line 3. If d is greater than a threshold $d_{\max}$, then p_i is meant to not belong to l. Hence l is saved and a new running line is started with p_i and p_{i+1} in line 6. Otherwise, when d is lower than $d_{\max}$, p_i is added to l, which is fit again considering p_i.

As already mentioned, a linear fitting method is at the core of the algorithm. LT performance is highly conditioned by the linear fitting method used, so a good method must be chosen. In Section 7.3 some common linear fitting methods are discussed with the focus on line features that have to be extracted from range laser

Algorithm 10 Line Tracking

Require: List $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ where $p_i = (x_i, y_i)$ are laser scan points and threshold $d_{\max}$ that represents the maximum distance allowed from a point p_i to the current estimated straight line l.
Ensure: List $\mathcal{L} = \langle l_1, \ldots, l_m \rangle$ with detected lines l_i.
Algorithm: **LT**$(\mathcal{P}, d_{\max})$ **return** $\mathcal{L}$

```
1:  l = new line with p_1, p_2                          ▷ line through p_1 and p_2
2:  for all p_i ∈ P from p_3 do
3:      d = distance of p_i to l
4:      if d > d_max then
5:          add l to L
6:          l = new line with p_i, p_{i+1}              ▷ next iteration on p_{i+2}
7:      else
8:          add p_i to l                                ▷ fit line
9:      end if
10: end for
11: return L
```

scans. Contrary to other line extraction algorithms, that fit the detected lines *a posteriori*, LT has the fitting process hardcoded. Since the line representation is also crucial, the linear fitting algorithm must be adapted to work with a particular type of line parameters. Clearly, all this applies to the rest of line extraction algorithms, LT is just more sensitive though.

7.2.2 Successive Edge Following

SUCCESIVE EDGE FOLLOWING

The *Succesive Edge Following* (SEF) method is one of the fastest and simplest line extraction algorithms [127]. It works directly with the polar form $(\rho\ \theta)^T$ of the scan points and it does not compute the linear fitting, which must be done after the set of lines have been extracted. It shares with LT the ability to extract lines without knowing the whole set of scan points in advance, but it extracts lines in a completely different manner.

In SEF, a fast distance d between a pair of consecutive scan points p_{i-1} and p_i is computed to test if it is higher than a threshold $d_{\max}$, as shown in Algorithm 11. If so, the running line l is saved and a new one is started in line 5; otherwise, the new point p_i is added to l. This way, the algorithm only have to remember the single last scan point p_{i-1} to compute the distance between it and the next scan point p_i. It is trivial to compute d as shown graphically in Figure 7.5, since scan points are in polar form $(\rho\ \theta)^T$ and the distance d is simply the absolute difference $|\rho_{i-1} - \rho_i|$, hardcoded in line 3.

SEF has a computational cost of $\mathcal{O}(n)$ for n scan points, since it only has to compute the distance between all consecutive pairs of lines, i.e. $n-1$ times. Recall that the extracted lines are not fit to the points that form them because this step is done over the resulting set of lines $\mathcal{L}$ output by the algorithm. The major drawback of this algorithm lies precisely in the fact that it does not computed the running line. Indeed, it only takes into account the last point contained in the running line. This makes SEF highly sensitive to outliers and even low measurement noise.

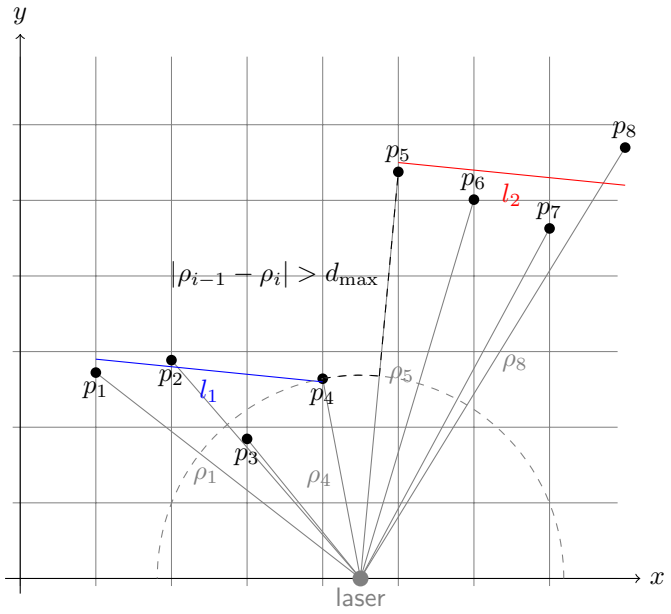


Figure 7.5: Successive Edge Following

Algorithm 11 Successive Edge Following

Require: List $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ where $p_i = (\rho_i, \theta_i)$ are laser scan points in polar form and threshold d_{max} that represent the maximum absolute difference allowed between two consecutive points p_{i-1} and p_i.
Ensure: List $\mathcal{L} = \langle l_1, \ldots, l_m \rangle$ with detected lines l_i.
Algorithm: **SEF**$(\mathcal{P}, d_{max})$ **return** $\mathcal{L}$

```
1:  l = new line with p_1, p_2                    ▷ line through p_1 and p_2
2:  for all p_i ∈ P from p_3 do
3:      if |ρ_{i-1} − ρ_i| > d_max then
4:          add l to L                             ▷ fit line
5:          l = new line with p_i, p_{i+1}         ▷ next iteration on p_{i+2}
6:      else
7:          add p_i to l
8:      end if
9:  end for
10: return L
```

7.2.3 Split & Merge

SPLIT & MERGE

The *Split & Merge* (SM) method is probably the most popular line extraction algorithm [102, 122], originally developed in the context of Computer Vision [109]. Given a set of points $\mathcal{P}$ from a range laser scan, a line l is fit to the *whole* set of points, which makes a big difference with LT and SEF because the complete scan is demanded. Line l is shown dashed (_ _ _) in Figure 7.6 as the result of the linear regression. Then the distance from each point p_i to l is computed to obtain the point p_m with the higher distance d_m. If d_m is higher than a threshold $d_{\min}$, the

set of points is split at p_m and the algorithm is called recursively to both subsets of points $\langle p_1, \ldots, p_m \rangle$ and $\langle p_m, \ldots, p_n \rangle$, as shown in the line 6 of Algorithm 12.

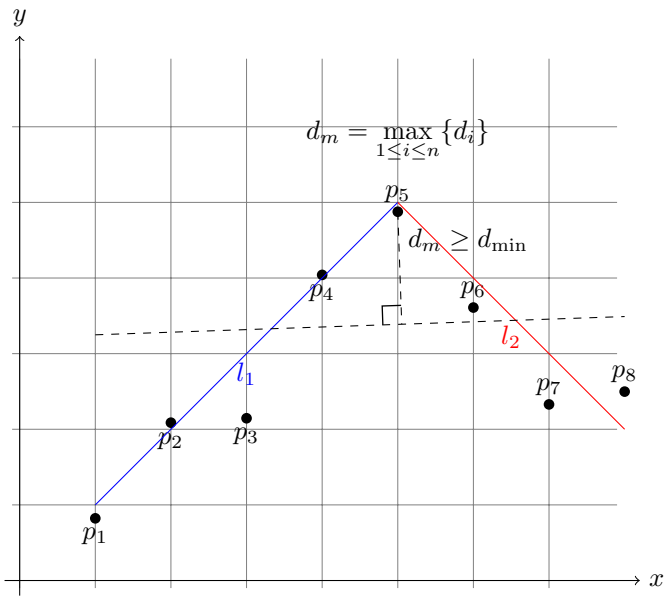


Figure 7.6: Split & Merge

The original algorithm was implemented iteratively [109]. Nevertheless, it can be implemented recursively, in such a way that it is easier to understand. The basic recursive algorithm is shown in Algorithm 12 accepting as a parameter the threshold $d_{\min}$, which establishes when to split a line. The value of this threshold depends on the environment and it is affected by the noise of the measurement sensor, since the algorithm is sensitive to outliers significantly.

Algorithm 12 Split & Merge

Require: List $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ where $p_i = (x_i, y_i)$ are laser scan points and threshold $d_{\min}$ that represents the min distance need from the most remote point p_m to the estimated straight line to split it.

Ensure: List $\mathcal{L} = \langle l_1, \ldots, l_m \rangle$ with detected lines l_i.
The extracted lines must pass the collinearity test of the merging step calling **Merge**($\mathcal{L}$, $\rho_{\max}$, $\theta_{\max}$) with the final list $\mathcal{L}$.

Algorithm: S&M($\mathcal{P}, d_{\min}$) **return** $\mathcal{L}$

1: l = new line with $p_1, \ldots, p_n$ ▷ fit line
2: p_m, d = point with max distance d to l
3: **if** $d < d_{\min}$ **then**
4: **return** $\langle l \rangle$
5: **else**
6: **return S&M**($\langle p_1, \ldots, p_m \rangle, d_{\min}$) $\cup$ **S&M**($\langle p_m, \ldots, p_n \rangle, d_{\min}$) ▷ split
7: **end if**

The basic Split & Merge algorithm is usually improved adding some additional threshold parameters, such as the minimum line length $l_{\min}$ and the minimum num-

ber of points $p_{\min}$ that must form a line. Algorithm 13 accepts these parameters, which work as base cases for the recursive calls and make the algorithm faster and more accurate, since outliers get filtered. To some extend, this process is equivalent to an *a priori* clustering used to removed outliers and noisy scan points [102].

Algorithm 13 Split & Merge Extended

Require: List $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ where $p_i = (x_i, y_i)$ are laser scan points and threshold $d_{\min}$ that represents the min distance need from the most remote point p_m to the estimated straight line to split it.
Minimum number of points $p_{\min}$ need to extracted a line and minimum line length $l_{\min}$ allowed.

Ensure: List $\mathcal{L} = \langle l_1, \ldots, l_m \rangle$ with detected lines l_i.
The extracted lines must pass the collinearity test of the merging step calling **Merge**($\mathcal{L}$, $\rho_{\max}$, $\theta_{\max}$) with the final list $\mathcal{L}$.

Algorithm: S&MExt($\mathcal{P}, d_{\min}, p_{\min}, l_{\min}$) **return** $\mathcal{L}$

```
if n < p_min then
    return ∅                                              ▷ no line
end if
l = new line with p_1, ..., p_n                           ▷ fit line
p_m, d = point with max distance d to l
if d < d_min then
    if ‖p_1 − p_n‖ < l_min then
        return ∅                                          ▷ no line
    else
        return ⟨l⟩
    end if
else
    return S&MExt(⟨p_1, ..., p_m⟩, d_min, p_min, l_min) ∪
           S&MExt(⟨p_m, ..., p_n⟩, d_min, p_min, l_min)   ▷ split
end if
```

MERGE

Once the set of lines $\mathcal{L}$ is obtained, the *merge* step of the algorithm is called. It iterates over all possible pairs of lines to analyzed whether they are collinear or not.

COLLINEARITY

The *collinearity* test is applied to the line parameters, which are the polar coordinates ρ and θ of the line polar form $\rho = x \cos\theta + y \sin\theta$. If the absolute difference between each line parameter $|\rho_i - \rho_j|$ and $|\theta_i - \theta_j|$ is lower than a particular threshold $\rho_{\max}$ and $\theta_{\max}$, the lines l_i and l_j are collinear and they are consequently merged into a single one. The merging step consists on fitting a new line formed by the points of both l_i and l_j; their points must be still accesible so. It yields a more compact representation space that will make the data association process easier.

The computational cost of the merging step is $\mathcal{O}\left(n^2\right)$ in the worst case, for n lines detected, since $\sum_{i=1}^{n-1} n - i = \frac{n(n-1)}{2}$ pairs of lines are compared. If a pair of lines are merged, the number of lines n is reduced by one unit, which provides a moderate computational cost reduction on avarage. Anyway, this cost is negligible because it is defined for the low-dimensional feature space representation, where n tends to be even lower than 10 per range laser scan. Although this step is part of the SM algorithm, it might be applied to other line extraction methods too. Thus, algorithms like Line Tracking and Succesive Edge Following may profit from this merging process.

Algorithm 14 Merge

Require: List $\mathcal{L} = \langle l_1, \ldots, l_n \rangle$ where l_i are lines extracted directly from laser scan points.
Collinearity thresholds $\rho_{\max}$ and $\theta_{\max}$ that represent the maximum difference allow for each line parameter; lines are represented in polar form $\rho = x\cos\theta + y\sin\theta$.
Ensure: List $\hat{\mathcal{L}} = \langle \hat{l}_1, \ldots, \hat{l}_m \rangle$ with merged collinear lines $\hat{l}_i$.
Algorithm: Merge($\mathcal{L}$, $\rho_{\max}$, $\theta_{\max}$) **return** $\hat{\mathcal{L}}$

```
1: while L not empty do
2:     delete l_0 from L                          ▷ take first line of current set of lines L
3:     for all l_i ∈ L do
4:         if |ρ_0 − ρ_i| < ρ_max ∧
              |θ_0 − θ_i| < θ_max then            ▷ modular distance within [0, 2π]
5:             l_0 = l_0 ∪ l_i                    ▷ merge lines
6:             delete l_i from L
7:         end if
8:     end for
9:     add l_0 to L̂                               ▷ fit line if it was merged
10: end while
11: return L̂
```

Split & Merge is highly customizable:

- The linear fitting algorithm used to compute the line parameters of the line that best fit a set of scan points. Several choices are discussed in Section 7.3 with methods that provides better accuracy at the same computational cost.
- Minimum number of points $p_{\min}$ that a line must have.
- Minimum length $l_{\min}$ that a line must have.
- Minimum distance $d_{\min}$ that the point p further from the line l that fit the scan points must have to split l at p.
- The algorithm to merge collinear lines.
- Collinearity conditions $\rho_{\max}$ and $\theta_{\max}$ that establish the maximum difference between the parameters ρ and θ of two lines —in polar form $\rho = x\cos\theta + y\sin\theta$— to be considered collinear.
- Furthermore, there exists some improvements that might be applied to the algorithm. In [3] it is proposed a residual analysis before split. It also advices to merge non-consecutive segments, which is actually done by Algorithm 14.

One of the major drawbacks of SM is that it must fit the *whole* set of scan points for each recursive call. This not only has a high impact on the computational cost of the algorithm, but also in the robustness. With $180°$ scans, the line that best fit all scan points usually left the leading and trailing scan points as the furthest. The split process under this cases tends to produce subsets of very few points, that are skipped thus. Some lines might be lost with this approach. A more efficient and robust variant of Split & Merge that only takes the first and last points of the scan to construct the line l —linear fitting is not needed— is known as Iterative End-Point Fit, commonly used with $180°$ scans in liu of SM.

7.2.4 Iterative End-Point Fit

ITERATIVE END-POINT FIT

The *Iterative End-Point Fit* (IEPF) algorithm is a variant of Split & Merge where the only difference is that the line fitting is not applied to the whole set of points, but only the first and last ones [45, 102, 127]. The reader may compare Figure 7.7 with 7.6 to see the difference between the line constructed by IEPF and SM, respectively. This makes IEPF faster than SM and still accurate, since the extracted lines are actually fit with all the points that form then.

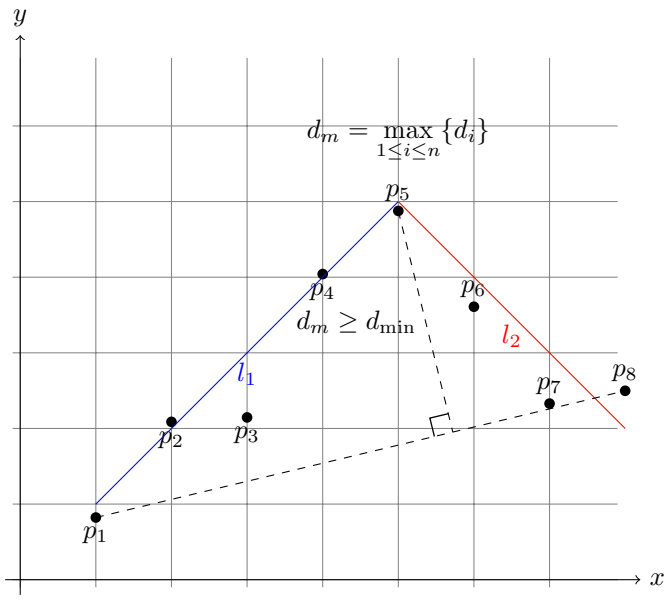


Figure 7.7: Iterative End-Point Fit

Algorithm 15 Iterative End-Point Fit

Require: List $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ where $p_i = (x_i, y_i)$ are laser scan points and threshold $d_{\min}$ that represents the min distance need from the most remote point p_m to the estimated straight line to split it.

Ensure: List $\mathcal{L} = \langle l_1, \ldots, l_m \rangle$ with detected lines l_i.
The extracted lines must pass the collinearity test of the merging step calling **Merge**($\mathcal{L}$, $\rho_{\max}$, $\theta_{\max}$) with the final list $\mathcal{L}$.

Algorithm: IEPF($\mathcal{P}, d_{\min}$) **return** $\mathcal{L}$

1: l = new line with p_1, p_n ▷ line through p_1 and p_n
2: p_m, d = point with max distance d to l
3: **if** $d < d_{\min}$ **then**
4: add $p_2, \ldots, p_{n-1}$ to l ▷ fit line
5: **return** $\langle l \rangle$
6: **else**
7: **return IEPF**($\langle p_1, \ldots, p_m \rangle, d_{\min}$) ∪ **IEPF**($\langle p_m, \ldots, p_n \rangle, d_{\min}$) ▷ split
8: **end if**

Since IEPF is simply a variant of Split & Merge, the basic algorithm shown in

Algorithm 15 is fairly the same, with the exception of the line construction in line 1. Recall that the line is constructed with the first and last points of the scan, so we get rid of the linear fitting step used in the original SM algorithm. When a line is detected, the rest of the points within the interval defined by the first and last ones are added to the line and a linear fitting method is applied. Contrary to SM, this is only done for extracted lines, which reduce the computational cost significantly.

Even more interesting is the fact that the splitting step performs better when the line l passes through the first and last scan points. This seems intuitively reasonable and can be empirically observed for 180° range laser scans. However, this is not suitable for 360° range laser scans because there is neither such first nor last scan points, since the scan is circular.

Algorithm 16 Iterative End-Point Fit Extended

Require: List $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ where $p_i = (x_i, y_i)$ are laser scan points and threshold $d_{\min}$ that represents the min distance need from the most remote point p_m to the estimated straight line to split it.
Minimum number of points $p_{\min}$ need to extracted a line and minimum line length $l_{\min}$ allowed.

Ensure: List $\mathcal{L} = \langle l_1, \ldots, l_m \rangle$ with detected lines l_i.
The extracted lines must pass the collinearity test of the merging step calling **Merge**($\mathcal{L}$, $\rho_{\max}$, $\theta_{\max}$) with the final list $\mathcal{L}$.

Algorithm: IEPFExt($\mathcal{P}, d_{\min}, p_{\min}, l_{\min}$) **return** $\mathcal{L}$

1: **if** $n < p_{\min}$ **then**
2: **return** $\varnothing$ ▷ no line
3: **end if**
4: $l =$ new line with p_1, p_n ▷ line through p_1 and p_n
5: $p_m, d =$ point with max distance d to l
6: **if** $d < d_{\min}$ **then**
7: **if** $\|p_1 - p_n\| < l_{\min}$ **then**
8: **return** $\varnothing$ ▷ no line
9: **else**
10: add $p_2, \ldots, p_{n-1}$ to l ▷ fit line
11: **return** $\langle l \rangle$
12: **end if**
13: **else**
14: **return IEPFExt**($\langle p_1, \ldots, p_m \rangle, d_{\min}, p_{\min}, l_{\min}$) $\cup$
IEPFExt($\langle p_m, \ldots, p_n \rangle, d_{\min}, p_{\min}, l_{\min}$) ▷ split
15: **end if**

Similarly to Split & Merge it is possible to extend the IEPF algorithm with some threshold parameters that filter noise and outliers. The IEPF extended algorithm is shown in Algorithm 16, for completeness.

7.2.5 Other Line Extractors

The line extraction algorithms discussed thus far are especially suitable for real time applications. They have a low computational cost and generates a sufficient robust set of line features. Nevertheless, there exists many other algorithms, but they demand higher computational resources. They are usually run *offline* and some of them can be very accurate. We present a brief list of such algorithms taken from

[102], where they are further analyzed and compared, showing the pseudo-code and specific computational cost.

LINE REGRESSION **Line Regression** Inspired on the Hough Transform, *Line Regression* first transforms the line extraction problem into a search problem in the model space, i.e. the line parameter domain [4]. Then applies the *Agglomerative Hierarchical Clustering* (AHC) algorithm to construct adjacent line segments.

AGGLOMERATIVE HIERARCHICAL CLUSTERING

RANDOM SAMPLE CONSENSUS **RANSAC** *RANdom SAmple Consensus* [50, 161] is an algorithm for robust fitting of models in the presence of outliers. It is a generic segmentation method that can be used with arbitrary features once we have their model. It is easy to implement and very popular in Computer Vision to extract features [53].

HOUGH TRANSFORM **Hough Transform** Although it tends to be most successfully applied to line finding on intensity images [53], the *Hough Transform* (HT) has been brought in to Robotics for extracting lines from range laser scans images [71, 112].

EXPECTATION MAXIMIZATION **Expectation Maximization** The *Expectation Maximization* (EM) algorithm is a probabilistic method commonly used in missing variable problems [39]. It has been used as a line extrator in Computer Vision [53] and Robotics [112], where it is significantly robust.

WINDOW SAMPLING CONSENSUS WITH GLOBAL EVALUATION **Other** There exists more line extraction methods that are extensions or modifications of those discussed thus far; for instance, the *Window SAmpling Consensus with Global Evaluation* (WSAC-GE) [120]. This algorithms begins with an empty map, and successively proposes a new map by adding or removing lines from the previous map, according to which map is best evaluated by a global function. Problems associated with outliers are handled by a probabilistic search and different segments of the same line are identified easily.

CLUSTERING It is common to apply a *clustering* algorithm prior to the line extraction process. It helps to removed outliers and spurious scan points that may lead to artifacts or *false possitives* [45, 46]. Furthermore, a fast clustering step reduce the measurement dimensionality and improve the avarage speed of the complete feature extraction process. Matter of fact, [102] makes use of a simple clustering algorithm for filtering largely noisy points and coarsely dividing a raw scan into contiguous groups, i.e. *clusters*. As a result, clusters having too few points are removed. In [120] they call *Split & Merge Split & Merge* (SMSM) to an algorithm that applies a clustering process before SM (actually IEPF variant) to make it more robust in the presence of outlying data, which is actually the same approach explained thus far. Similarly, [18] proposes *Split & Merge Fuzzy* (SMF), that employs a fuzzy clustering approach.

SPLIT & MERGE SPLIT & MERGE

SPLIT & MERGE FUZZY

The threshold parameters used by the extended versions of SM and IEPF shown thus far have an analogous goal. Recall these thresholds do limit the minimum line length $l_{\min}$ and minimum number of points $p_{\min}$. With $l_{\min}$ and $p_{\min}$ an equivalent same filtering process to clustering is achieved, because remote and loose scan points tend to be ignored.

7.2.6 Discussion

In regard to the line extraction process the most suitable algorithm for real time is Split & Merge and its variant Iterative End-Point Fit. They are sufficiently fast and robust to work with range laser scans in real time. Other algorithms are unfeasible

by far for real time or cannot deal with measurement noise and outliers. The Line Tracking and Successive Edge Following algorithms discussed thus far perform poorly under such scenario and require a thorough parametrization tunning.

It has been already mentioned that IEPF is faster than SM since it does not need to fit a line for the whole set of scan points prior to decide to split or not. Furthermore, for $180°$ scans IEPF gives better results as discussed in Section 7.2.4. Certainly, IEPF is commonly used for feature-based SLAM [4, 120, 121].[1]

7.3 Linear Fitting

LINEAR FITTING

The *linear fitting* is a mathematical procedure for finding the best-fitting line l to a given set of points $\mathcal{P}$ by minimizing the sum of the squares of the offsets (the *residuals*) of the points from the line model. In the presence of noisy data the linear fitting algorithm becomes an important election. Figure 7.8 shows the most common choices, where the robust estimation methods actually cover a great bunch of statistical solutions.

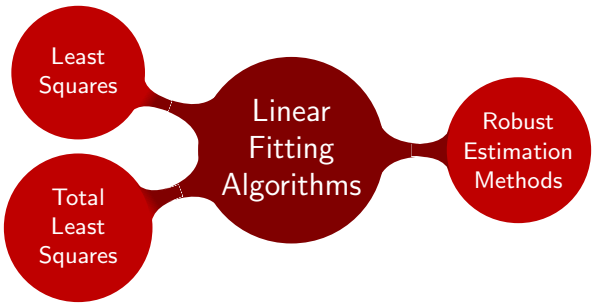


Figure 7.8: Line Fitting Algorithms

The fitting procedure and the line model employed are key aspects of the line extraction algorithms. They affect the quality of the detected line's parameters. The parameters of the model are adjusted to find the optimal value of a *merit function*, yielding the *best-fit parameters*. Typical data never exactly fit the model that is being used. Therefore, we need to test the *goodness-of-fit* against some useful statistical standard to assess whether or not the model is appropiate.

MERIT FUNCTION
BEST-FIT PARAMETERS
GOODNESS-OF-FIT

The line model used to fit the set of observations is also a design element to choose that constraints the fitting process and affects the goodness-of-fit. It is common to used the Intercept-Slope model $y = mx + b$ to derive the mathematical expressions of the best-fit parameters m and b. However, in the presence of *vertical* lines this model is not suitable. A good replacement is the Hessian model $\rho = x\cos\theta + y\sin\theta$, that can represent any kind of line properly.

Most of the linear fitting methods discussed below are taken from [157, 158]. Other methods especially aimed for robust estimation are covered by [114, 115] and several articles or technical reports [3, 30, 37, 112, 131]. In either case, they provide the analytical expressions for direct computation of the line parameters. However, they usually employed the Intercept-Slope model or a general vector notation, so

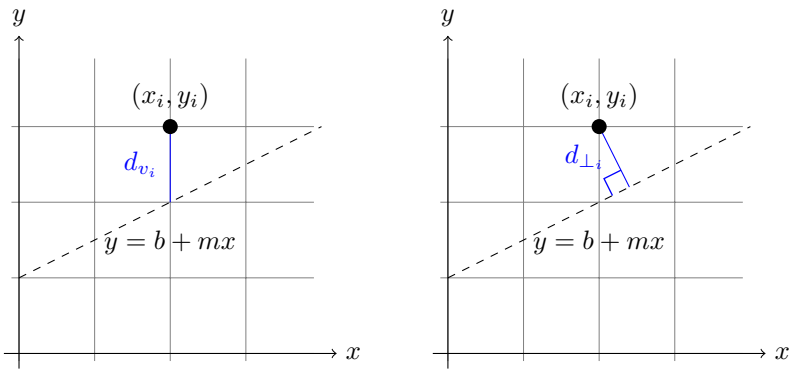
[1] It may be refered to as SM though, since IEPF is usually regarded as a simple variant.

they must be adapted. Fortunately, some works regarding line extraction from range laser scans provide equations for the Hessian model parameters ρ and θ [3, 4, 121].

7.3.1 Least Squares

LEAST SQUARES
MINIMUM SQUARED ERROR

The *Least Squares* (LS) method is the simplest approach [115, 157]. It is also refered to as the *Minimum Squared Error* (MSE) method [45, 46]. It minimizes the sum of the squares of the offsets instead of the offset absolute values because this allows the residuals to be treated as a continuous differentiable quantity. However, because squares of the offsets are used, outlying points can have a disproportionate effect on the fit, a property which may or may not be desirable depending on the problem at hand. Figure 7.9 shows the graphical difference between the plain offset or vertical distance (a) and the absolute offset or perpendicular distance (b), which are employed by the LS and Total Least Squares (TLS) algorithms, respectively.



(a) Least Squares Fitting. Vertical distance

(b) Total Least Squares Fitting. Perpendicular distance

Figure 7.9: Distance between original points and fitting line

In our case, the perpendicular distance is clearly the right choice, since scan points have errors in both variables x and y, and there are vertical lines. Indeed, LS performs poorly with lines that has a slope m greater than $m = 1$. Anyway, LS is explained now and TLS is later introduced in Section 7.3.2, since its deduction is more complex. Thus, LS uses the vertical deviations R^2 computing the vertical distances d_{v_i}, meanwhile TLS uses the perpendicular deviations $R_\perp$ computing the perpendicular distances $d_{\perp_i}$. The vertical distance d_{v_i} for each point i is shown below. Later, $d_{\perp_i}$ will be devised in terms of d_{v_i} for the TLS algorithm.

$$d_{v_i} = |y_i - (b + mx_i)| \tag{7.1}$$

Vertical Least Squares fitting proceeds by finding the sum of the squares of the vertical deviations R^2 of a set of n data points

$$R^2 = \sum_{i=1}^{n} (y_i - f(x_i \mid a_1, \ldots, a_k))^2 \tag{7.2}$$

from a function f of k variables. For a linear fit we have

$$f(x \mid b, m) = b + mx \tag{7.3}$$

which is the Intercept-Slope model with y-intercept b and slope m. Hence, the model parameters are obtained by minimizing

$$R^2(b,m) = \sum_{i=1}^{n} \left(y_i - (b + mx_i)\right)^2 \tag{7.4}$$

The best-fit parameters are then

$$m = \frac{\operatorname{cov}(x,y)}{\sigma_x^2} = \frac{SS_{xy}}{SS_{xx}} \tag{7.5}$$

$$b = \bar{y} - m\bar{x} \tag{7.6}$$

where

$$SS_{xx} = \sum_{i=1}^{n} (x_i - \bar{x})^2 = \left(\sum_{i=1}^{n} x_i^2\right) - n\bar{x}^2 \tag{7.7}$$

$$SS_{yy} = \sum_{i=1}^{n} (y_i - \bar{y})^2 = \left(\sum_{i=1}^{n} y_i^2\right) - n\bar{y}^2 \tag{7.8}$$

$$SS_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \left(\sum_{i=1}^{n} x_i y_i\right) - n\bar{x}\bar{y} \tag{7.9}$$

which might be also written as the variances σ_x^2 and σ_y^2, and the covariance $\operatorname{cov}(x,y)$

$$\sigma_x^2 = \frac{SS_{xx}}{n} \tag{7.10}$$

$$\sigma_y^2 = \frac{SS_{yy}}{n} \tag{7.11}$$

$$\operatorname{cov}(x,y) = \frac{SS_{xy}}{n} \tag{7.12}$$

and

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{7.13}$$

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{7.14}$$

are the means with respect to x and y coordinates.

If we consider the Hessian model $\rho = x\cos\theta + y\sin\theta$ the merit function turns into

$$R^2(\rho,\theta) = \sum_{i=1}^{n} \left(\rho - x_i\cos\theta - y_i\sin\theta\right)^2 \tag{7.15}$$

The model parameters ρ and θ that are obtained by minimizing (7.15) actually yields the solution for the perpendicular offsets, i.e. the TLS best-fit parameters. In fact, it is common to employ the Hessian model since it simplifies the derivation and the expressions of the TLS method [3, 4, 121]. Later, in Section 7.3.2 we will see those expressions, but for now we will devised the expressions that give the LS best-fit parameters. Although it is not found in the literature, it is straightforward to

derive the expressions for the Hessian model parameters from those of the Intercept-Slope model. We only have to apply the relations between their parameters

$$\theta = \arctan -\frac{1}{m} = \operatorname{atan2}(-1, m) \tag{7.16}$$

$$\rho = b \sin\theta \tag{7.17}$$

Hence, the best-fit parameters for the Hessian model are

$$\theta = \operatorname{atan2}(-SS_{xx}, SS_{xy}) \tag{7.18}$$

$$\rho = \bar{x}\cos\theta + \bar{y}\sin\theta \tag{7.19}$$

7.3.2 Total Least Squares

TOTAL LEAST SQUARES
EIGENVECTOR
ERRORS IN BOTH VARIABLES

The *Total Least Squares* (TLS) [37, 158] overcomes the main problem of the LS method. TLS is also known as the *Eigenvector* method [45] when derived in vector notation. The method is also known as the Least Squares for *Errors in Both Variables* or *Coordinates*, or simply the Least Squares *Orthogonal* [115, 131], since it minimizes the perpendicular distance $d_{\perp_i}$ from the points $i = 1, \ldots, n$ to the model, as mentioned above and shown in Figure 7.9(b). Such distance $d_{\perp_i}$ in terms of the vertical distance d_{v_i} for the line model (7.3) is

$$d_{\perp_i} = \frac{dv_i}{\sqrt{1+m^2}} \tag{7.20}$$

The deduction of this equivalence is shown below. The author may skip this proof at first reading. It is actually quite straightforward and it is based on the relation between the triangles shown in Figure 7.10.

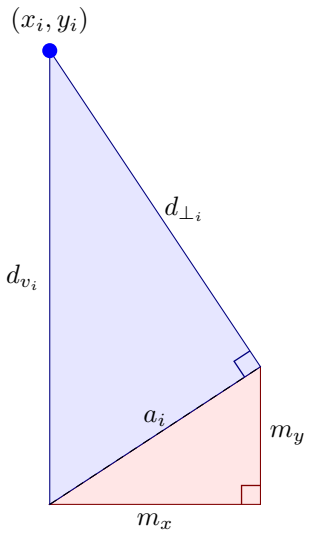


Figure 7.10: Relation between perpendicular and vertical distance

Relationship between perpendicular and vertical distance. First of all, we apply the Pythagorean theorem to the upper triangle in Figure 7.10

$$d_{v_i}^2 = d_{\perp_i}^2 + a_i^2$$
$$d_{\perp_i} = \sqrt{d_{v_i}^2 - a_i^2} \tag{7.21}$$

we take the possitive solution of the square root since $d_{\perp_i}$ is a distance, i.e. $d_{\perp_i} \geq 0$.

The slope m of the line model can be expressed both as the quotient of the lower triangle's catheti m_x and m_y

$$m = \frac{m_y}{m_x} \tag{7.22}$$

and as the quotient of the upper triangle's catheti a_i and $d_{\perp_i}$

$$m = \frac{m_y}{m_x} = \frac{a_i}{d_{\perp_i}} \tag{7.23}$$

Hence

$$d_{\perp_i} = \frac{a_i}{m} \tag{7.24}$$

which is equal to (7.21)

$$d_{\perp_i} = \frac{a_i}{m} = \sqrt{d_{v_i}^2 - a_i^2} \tag{7.25}$$

Operating to obtain a_i yields

$$\frac{a_i}{m} = \sqrt{d_{v_i}^2 - a_i^2} \tag{7.26}$$
$$\frac{a_i^2}{m^2} = d_{v_i}^2 - a_i^2 \tag{7.27}$$
$$a_i^2 = \left(d_{v_i}^2 - a_i^2\right) m^2 \tag{7.28}$$
$$a_i^2 - \left(d_{v_i}^2 - a_i^2\right) m^2 = 0 \tag{7.29}$$
$$a_i^2 \left(1 + m^2\right) - m^2 d_{v_i}^2 = 0 \tag{7.30}$$
$$a_i = \sqrt{\frac{\left(m d_{v_i}\right)^2}{1 + m^2}} \tag{7.31}$$
$$a_i = \frac{m d_{v_i}}{\sqrt{1 + m^2}} \tag{7.32}$$

Finally, we replace the expression of a_i in (7.24), that is

$$d_{\perp_i} = \frac{a_i}{m} = \frac{m d_{v_i}}{m\sqrt{1 + m^2}} \tag{7.33}$$
$$d_{\perp_i} = \frac{d_{v_i}}{\sqrt{1 + m^2}} \tag{7.34}$$

□

The model parameters are obtained by minimizing the perpendicular deviations $R_\perp^2$ of a set of n data points from the line model (7.3)

$$R_\perp^2(b, m) = \sum_{i=1}^{n} \frac{\left(y_i - (b + m x_i)\right)^2}{1 + m^2} \tag{7.35}$$

The best-fit parameters are then

$$m = -M \pm \sqrt{M^2 + 1} \tag{7.36}$$

$$b = \bar{y} - m\bar{x} \tag{7.37}$$

where

$$M = \frac{SS_{xx} - SS_{yy}}{2SS_{xy}} \tag{7.38}$$

where SS_{xx}, SS_{yy} and SS_{xy} are those of (7.7), (7.8) and (7.9). Similarly, $\bar{x}$ and $\bar{y}$ are the means, as in (7.13) and (7.14). The reader may compare the expressions of the line model parameters obtained here with the TLS method with those of the LS method discussed in Section 7.3.1.

We return to the topic of the Hessian model $\rho = x\cos\theta + y\sin\theta$ now, that yields the merit function (7.15). As previously pointed, the minimization results in the best-fit parameters for the TLS fitting [3, 4, 121], which are

$$\theta = \frac{1}{2}\mathrm{atan2}\left(-2SS_{xy}, SS_{yy} - SS_{xx}\right) \tag{7.39}$$

$$\rho = \bar{x}\cos\theta + \bar{y}\sin\theta \tag{7.40}$$

In the literature it is common to find the TLS method as a robust estimator that includes an associated weight ω_i for each point [4, 115, 121]. That is actually a robust estimation method known as Weighted TLS, discussed in Section 7.3.3. The expressions given here might be thought as a particular case with $\omega_i = 0$ for $i = 1, \ldots, n$.

It is also possible to derive the expressions above from those of the Intercept-Slope model. Such mathematical derivation consists basically on applying the relations (7.16) and (7.17) between the Hessian model and the Intercept-Slope parameters. To obtain (7.39) the numerator and denominator of the constant M must be chosen properly and the expression

$$\mathrm{atan2}\left(y, x\right) = 2\arctan\left(\frac{y}{x + \sqrt{x^2 + y^2}}\right) \tag{7.41}$$

given by the tangent half-angle formula, must be applied at the end.

7.3.3 Robust Estimation Methods

The literature on data modeling is considerably extent and many methods have statistical foundations. The robust estimation is meant to deal with noisy data. It allows to model such data within a confidence interval, which might be taken into account by the algorithms that work with the model. Here we will discussed only some classical and common *robust regression* methods focused on line fitting with range laser scans. Other methods —even other robust ones— are briefly described in Section 7.3.4.

ROBUST REGRESSION

The following sections discussed the robust variants of the Least Squares and Total Least Squares methods shown thus far. They introduce weights associated with each raw point. Each weight ω_i describes the uncertainty of the point i based on the measurement process. The uncertainty is usually stated as the standard deviation σ_i, that must be known in advance —it is often given in the measurement

sensor data sheet or through a calibration process. For instance, the SICK LMS200 range laser has a sistematic error of $\pm 15\mathrm{mm}$ and statistical error of $5\mathrm{mm}$ [70]. If we consider the sistematic error of $\varepsilon = \pm 15\mathrm{mm}$ it is possible to assign a standard deviation σ_i to each range measurement $\rho_i \mathrm{mm}$ according with the following expression

$$\sigma_i = \frac{\rho_i}{15} \tag{7.42}$$

which states that σ_i is inversely proportional to the ε.

Weighted Least Squares. Chi-Square Fitting

In [115] is devised a Weighted Least Squares (WLS) method coined *Chi-Square* fitting. It states that the maximum likelihood estimate of the model parameters is obtained by minimizing CHI-SQUARE

$$\chi^2(b,m) = \sum_{i=1}^{n} \left(\frac{y_i - (b + m x_i)}{\sigma_i} \right)^2 \tag{7.43}$$

For linear models, the probability distribution for different values of χ^2 at its minimum can nevertheless be derived analytically, and is the chi-square distribution for $n-k$ degrees of freedom, where n is the number of points and $k=2$ is the number of parameters of the line model. This method takes into account the standard deviation σ_i of each data point i, which makes it a robust estimation technique.

The best-fit parameters m and b have the same expressions (7.5) and (7.6) as LS, but the sums SS_{xx}, SS_{yy} and SS_{xy} are different

$$SS_{xx} = \sum_{i=1}^{n} \frac{(x_i - \hat{x})^2}{\sigma_i^2} = \left(\sum_{i=1}^{n} \frac{x_i^2}{\sigma_i^2} \right) - \hat{n}\hat{x}^2 \tag{7.44}$$

$$SS_{yy} = \sum_{i=1}^{n} \frac{(y_i - \hat{y})^2}{\sigma_i^2} = \left(\sum_{i=1}^{n} \frac{y_i^2}{\sigma_i^2} \right) - \hat{n}\hat{y}^2 \tag{7.45}$$

$$SS_{xy} = \sum_{i=1}^{n} \frac{(x_i - \hat{x})(y_i - \hat{y})}{\sigma_i^2} = \left(\sum_{i=1}^{n} \frac{x_i y_i}{\sigma_i^2} \right) - \hat{n}\hat{x}\hat{y} \tag{7.46}$$

where $\hat{x}$ and $\hat{y}$ are *pseudo*-means

$$\hat{x} = \frac{1}{\hat{n}} \sum_{i=1}^{n} \frac{x_i}{\sigma_i^2} \tag{7.47}$$

$$\hat{y} = \frac{1}{\hat{n}} \sum_{i=1}^{n} \frac{y_i}{\sigma_i^2} \tag{7.48}$$

that replace the means $\bar{x}$ and $\bar{y}$, and

$$\hat{n} = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \tag{7.49}$$

Weighted Total Least Squares. Errors in Both Variables

Similarly to WLS, in [115] is also devised a Weighted Total Least Squares (WTLS) method under the name *Errors in Both Coordinates*. It extends the TLS merit function (7.35) to incorporate the uncertainties σ_{x_i} and σ_{y_i} of both variables x_i and y_i. Doing so, the weight proposed

$$\omega_i = \frac{1}{\sigma_{x_i} + m^2\sigma_{y_i}} \tag{7.50}$$

actually resembles the perpendicular offsets and yields the merit function below for the Intercept-Slope model

$$\chi^2(b, m) = \sum_{i=1}^{n} \left(\frac{y_i - (b + mx_i)}{\sigma_{x_i} + m^2\sigma_{y_i}} \right)^2 \tag{7.51}$$

The authors argue that the m^2 term in the denominator is an impediment to obtain the analytical expressions of the model parameters. Actually, b has the same analytical expression as (7.6) with *pseudo*-means

$$\hat{x} = \frac{1}{\hat{n}} \sum_{i=1}^{n} \omega_i x_i \tag{7.52}$$

$$\hat{y} = \frac{1}{\hat{n}} \sum_{i=1}^{n} \omega_i y_i \tag{7.53}$$

that replace the means $\bar{x}$ and $\bar{y}$, and

$$\hat{n} = \sum_{i=1}^{n} \omega_i \tag{7.54}$$

where ω_i is taken from (7.50). Nonetheless, m must be computed numerically.

Fortunately, other works that discuss the WTLS method for the Hessian model $\rho = x\cos\theta + y\sin\theta$ [4, 121], considering the merit function below

$$\chi^2(b, m) = \sum_{i=1}^{n} \omega_i \left(\rho - x_i\cos\theta - y_i\sin\theta\right)^2 \tag{7.55}$$

provide the same expressions (7.39) and (7.40) as TLS for the best-fit parameters ρ and θ, but the sums SS_{xx}, SS_{yy} and SS_{xy} incorporate the weight ω_i

$$SS_{xx} = \sum_{i=1}^{n} \omega_i \left(x_i - \hat{x}\right)^2 = \left(\sum_{i=1}^{n} \omega_i x_i^2\right) - \hat{n}\hat{x}^2 \tag{7.56}$$

$$SS_{yy} = \sum_{i=1}^{n} \omega_i \left(y_i - \hat{y}\right)^2 = \left(\sum_{i=1}^{n} \omega_i y_i^2\right) - \hat{n}\hat{y}^2 \tag{7.57}$$

$$SS_{xy} = \sum_{i=1}^{n} \omega_i \left(x_i - \hat{x}\right)\left(y_i - \hat{y}\right) = \left(\sum_{i=1}^{n} \omega_i x_i y_i\right) - \hat{n}\hat{x}\hat{y} \tag{7.58}$$

where $\hat{x}$ and $\hat{y}$ are the *pseudo*-means (7.52) and (7.53), and $\hat{n}$ is taken from (7.54). However, notice that they consider the weight ω_i as a single value per point, instead of one per coordinate as in (7.50). Furthermore, the nonlinear dependency on m is omitted, allowing the analytical derivation above.

7.3.4 Other Linear Fitting Methods

Some of the most common and successful methods employed for linear modeling are briefly commented below. They usually demand higher computational resources or must be solved numerically. In contrast, the methods discussed thus far provide analytical expressions to compute the best-fit parameters of the linear model efficiently. Therefore, they are suitable for real time applications, while only some variants or optimizations of the following techniques are thought for such scenario.

NORMAL EQUATIONS

Normal equations Within the general linear Least Squares topic we find the *Normal equations* of the Least Squares problem [115]. They are presented in matrix form and require to solve a linear system, analytically or numerically.

SINGULAR VALUE DECOMPOSITION

SVD In many cases the Normal equations are very close to singular, in which case you may get no solution at all. The *Singular Value Decomposition* (SVD) produces a solution that is the best approximation for an overdetermined system, and whose values are smallest for an underdetermined system, in the Least Squares sense [115].

M-ESTIMATES

M-Estimates There are various sorts of robust statistical estimators, where *M-Estimates* are usually the most relevant class for model fitting, i.e. estimation of parameters [115, 121]. To fit a model by means of an M-Estimate, you first decide with M-Estimate to employ. It is common to choose the *Lorentzian* function [131] or the median, that simplifies the process and is known as *Minimizing Absolute Deviation* [115].

MINIMIZING ABSOLUTE DEVIATION

MONTE CARLO

Monte Carlo Random sampling methods like *Monte Carlo* (MC) methods might be useful to estimate parameters since they are used to implement Bayesian methods [115] —e.g. the *Markov Chain Monte Carlo* (MCMC) method.

MARKOV CHAIN MONTE CARLO

RANDOM SAMPLE CONSENSUS

RANSAC The *RANdom SAmple Consensus* algorithm is commonly used for robust estimation, since it is able to estimate the parameters with accuracy even when outliers are present in the data set [121, 161]. There exists some variants, like MSAC [155] and MLESAC [154] that produce better results, and even optimizations like Efficient RANSAC [124], that may be suitable for real time.

WINDOW SAMPLING CONSENSUS

Other There exists other methods that are basically modifications of the ones above, like the *Window SAmpling Consensus* (WSAC) method proposed by [121], which has a random algorithm similar to MSAC.

7.3.5 Discussion

For real time applications we may choose between the Least Squares or the Total Least Squares methods basically. Other methods might be considered, but they usually have a higher computational cost or they are complex to implement. LS and TLS have fairly the same computational cost $\mathcal{O}(n)$ for n points. Thus, the decision is based on the accuracy. TLS is clearly better than LS because it manage perpendicular offsets and support vertical lines better. Furthermore, TLS must be used with the Hessian model $\rho = x\cos\theta + y\sin\theta$, since it can represent vertical lines properly and the best-fit parameters expressions (7.39) and (7.40) are simpler.

The robust variants of LS and TLS incorpore a weight ω_i for each point at no additional computational cost. However, we need to assign ω_i according with the

measurement sensor specifications, not always available. More robust methods like RANSAC might be applied, but only some variants or optimizations are suitable for real time. In general, TLS fitting performs well with range lasers, since they are very precise. Anyway, robust estimation methods are less sensitive to outliers and may model data with greater accuracy.

7.4 Corner Extraction

In order to detect corners a simple but effective method, based on the lines obtained by a line extraction algorithm, might be applied. The proposed corner extractor shown in Algorithm 17 computes all possible line-line intersections in the line 4. This approach has a computational cost of $\mathcal{O}\left(n^2\right)$, since it computes $\sum_{i=1}^{n-1} n-i = \frac{n(n-1)}{2}$ line-line interceptions, where n is the number of lines detected. Only those intersection points that are close to the intercepting lines are saved. The distances to each line d_i and d_j are computed in lines 5 and 6 as the lower distance to the scan points that form the line. Therefore, only *real* corners are retrieved in line 8, and the rest are discarded.

Algorithm 17 Corner Extraction

Require: List $\mathcal{P} = \langle p_1, \ldots, p_n \rangle$ where $p_i = (x_i, y_i)$ are laser scan points and threshold $d_{\max}$ that represents the max distance from the interception point p of a pair of lines (l_i, l_j) to the actual segments $\bar{l}_i$ and $\bar{l}_j$ that they represent.
A line extraction method **LineExtractor**$(\mathcal{P}, \ldots)$ to obtain the list of lines $\mathcal{L}$ that will be analyzed to find interceptions, i.e. corners.
Ensure: List $\mathcal{C} = \langle c_1, \ldots, c_m \rangle$ with detected corners c_i.
Algorithm: CornerExtractor$(\mathcal{P}, d_{\max})$ **return** $\mathcal{C}$

```
1:  L = LineExtractor(P, ...)                          ▷ extract lines
2:  for all pair of lines (l_i, l_j) ∈ L do            ▷ analyze all possible pairs of lines
3:      if l_i and l_j intercept then
4:          p = intersection of l_i and l_j            ▷ interception point
5:          d_i = min distance from p to l̄_i
6:          d_j = min distance from p to l̄_j
7:          if d_i, d_j ≤ d_max then
8:              add p to C                             ▷ real interception
9:          end if
10:     end if
11: end for
12: return C
```

The algorithm accepts a threshold parameter $d_{\max}$ that specifies the maximum distance allowed from an intersection p to the lines l_i and l_j that have produced it. It is meant to decide whether p is a *real* interception or not. Therefore, if the distance to at least one line is greater than $d_{\max}$, p is discarded because it is a *virtual* intersection, not present in the environment.

7.5 Sensor Data Synchronization

When extracting features or information from multiple sensors is important to synchronize the retrieved data to describe the environment at a particular instant of time. Furthermore, this allows to integrate the raw information from such sensors into semantic information as part of a process known as *sensor fusion* [26, 97]. In this work we only use one measurement sensor —a range laser—, so there is no sensor fusion performed actually. However, we also receive odometry information from the wheel encoders, that must be put in correspondence with the laser scans.

SENSOR FUSION

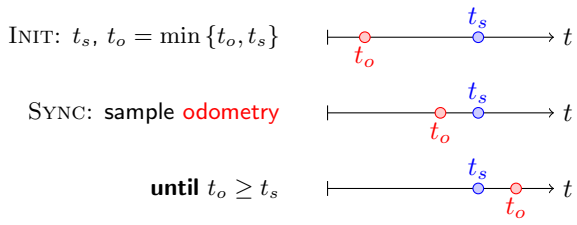


Figure 7.11: Odometry and Scan synchronization

Odometry and laser scans must be synchronized. Clearly, it is easier to interpolate the odometry at a desire time than a complete set of scan points. Therefore, we force to obtain an odometry sample with a timestamp t_o later than the laser scan one t_s, as shown in Figure 7.11. Then we can interpolate the robot pose at t_s from the odometry samples taken at t_{o-1} and t_o.

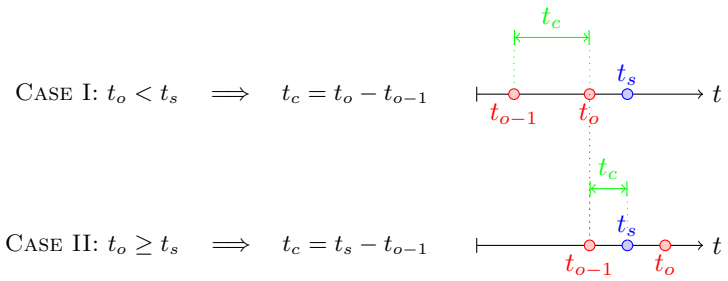


Figure 7.12: Control time computation

Additionally, the control time t_c is generally computed as the difference beween two consecutive odometry samples taken at t_{o-1} and t_o, that is the case I in Figure 7.12. The control time is a key element to compute the motion model embedded in the SLAM algorithms. When the odometry is interpolated at t_s, t_c is computed as shown for the case II in Figure 7.12, due to the odometry and laser scan synchronization.

7.6 Summary

The feature extraction process is one of the main steps of feature-based SLAM. The environment is represented by a map built with features, that bring a more

significative and low-dimensional representation space than raw data gather by the measurement equipment. The measurement sensor conditions the type of features that may be detected and the extraction process. For range lasers is common to extract lines and corner points, that are extracted using the detected lines to obtain *real* line-line interception points [45, 102, 120, 122, 127].

The lines extracted are fit to the scan points applying linear fitting methods. There are several linear fitting methods and many of them are robust in the presence of outliers. This methods are also known as estimators since they are used to estimate the best-fit parameters of a particular line model.

In this thesis we use one of the most widely used line extraction methods, Iterative End-Point Fit. As discussed in Section 7.2.4, although this variant of Split & Merge is very efficient and accurate, we employ an extended version that filters noisy points and outliers, similarly to clustering techniques [18, 102, 120]. The Total Least Squares fitting method for the Hessian model $\rho = x\cos\theta + y\sin\theta$ is used to fit the extracted lines, since it suffices for range laser scans. However, other robust estimation methods might be used provided they are suitable for real time SLAM.

Chapter 8

Data Association

8.1 Introduction

In real SLAM applications, the correspondences c^t between the observations z^t and the landmarks m^t of the map are rarely known. The number n of landmarks in the environment is also unknown. When the robot performs an observation of the environment, such observation must be associated with a landmark in the map or incorporated as a new landmark. This process is known as *data association*, which deals with the problem of the mapping between observations and landmarks [96]. The data association problem is one of the most complex in SLAM [100, 103]. The process is highly influenced both by the noise accumulated in the robot pose and the measurement error. That is, uncertainty in the SLAM posterior generates data association *ambiguity*. Two factors contribute to uncertainty in the SLAM posterior: measurement noise and motion noise, each producing a different kind of data association ambiguity [93].

DATA ASSOCIATION

AMBIGUITY

We will refer to data association ambiguity caused by measurement noise as *measurement ambiguity*. An example of measurement ambiguity is shown in Figure 8.1, where two ellipses depict the range of probable observations from two different landmarks. The observation shown as a red circle ○ plausibly could have come from either landmark. Attributing an observation to the wrong landmark due to measurement ambiguity will increase the error of the map and robot pose, but its impact will be relatively low becuase the observation could have been generated by either landmark with high probability. Therefore the effect on the landmark positions and the robot pose will be small.

MEASUREMENT AMBIGUITY

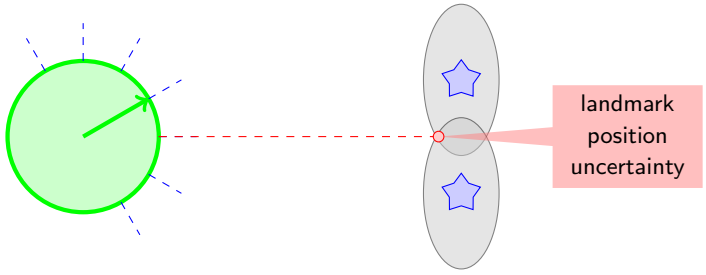


Figure 8.1: Measurement Ambiguity

Ambiguity in data association caused by motion noise can have much more severe consequences on estimation accuracy. Higher motion noise leads to higher pose uncertainty after incorporating a control. If this pose uncertainty is high enough, assuming different robot poses in this distribution will imply drastically different data association hypotheses. Such *motion ambiguity*, shown in Figure 8.2, is easily induced if there is significant rotational error in the robot's motion.

MOTION AMBIGUITY

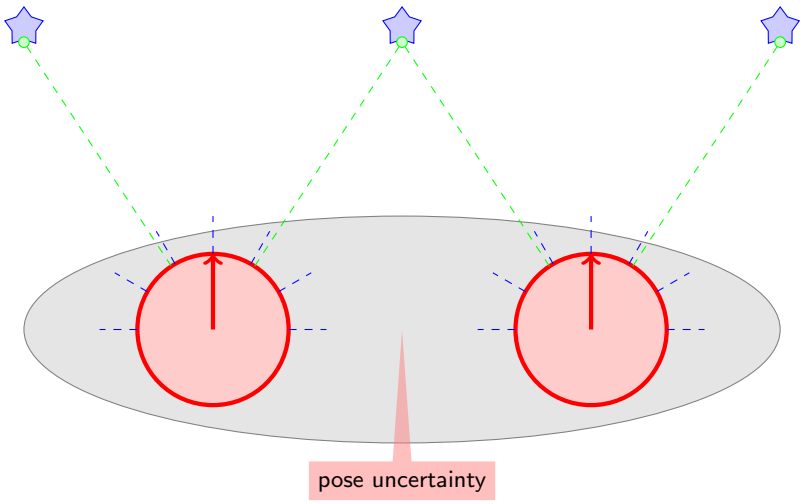


Figure 8.2: Motion Ambiguity

A data association algorithm is composed by two basic elements: a *compatibility test* between observations and landmarks for a given robot pose estimation, and a *selection criterion* to choose the best matchings among the set of compatible matchings [100]. There exists a great variety of solutions for the data association problem. In the sequel we will focus on two methods, while Section 8.4 gathers other alternatives described in short. The Maximum Likelihood (ML) method is discussed in detail since it is recommended by the authors of the FastSLAM algorithm employed in this work [69, 96, 100, 103]. This data association technique is tightly related with the classic Gated Nearest Neighbor (GNN) algorithm widely cited in the literature [92, 93, 94, 96, 148, 149]. Therefore, GNN is presented first to introduce the topic and show some of the fundamentals of the ML approach described subsequently.

COMPATIBILITY TEST

SELECTION CRITERION

8.2 Gated Nearest Neighbor

GATED NEAREST NEIGHBOR

The *Gated Nearest Neighbor* (GNN) algorithm is a classic technique usually applied to tracking problems [9]. This method is also called *Nearest Neighbor Gating* and usually referred to as *Nearest Neighbor* (NN) simply. The normalized squared innovation χ^2 test is used to determine the compatibility, and then the nearest neighbor rule, that takes the smallest Mahalanobis distance, is used to select the best matchings [100]. The great advantage of GNN is its conceptual simplicity and $\mathcal{O}(mn)$ computational complexity, where m is the number of features observed and n is the number of landmarks in the map.

GNN is reliable for features such as lines detected with range lasers, where clutter is low and sensor precision is high, as long as the motion uncertainty is moderate [27]. However, the reliability quickly plummets as the uncertainty of features relative to the robot pose increases, as happends when revisiting previously mapped regions after a long loop. Reliability also plummets when less precise sensors are employed, e.g. sonar or edge-based monocular vision [100].

8.2.1 Mathematical Derivation

We will now derive the mathematical foundations of the Gated Nearest Neighbor algorithm. The reader not interested in the mathematical details is invited to skip this section at first reading. Nevertheless, it is recommended to read it for a better understanding of the similarities and ramifications of the Maximum Likelihood technique later discussed.

In stochastic mapping [132] the state of the robot r and a set of n landmarks $\{l_1, \ldots, l_n\}$ of the environment is represented by a vector $\mathbf{x}$. Let $\hat{\mathbf{x}}$ be the estimation of the robot and feature locations, and $\mathbf{P}$ the covariance of the estimation error

$$\hat{\mathbf{x}} = \begin{pmatrix} \hat{\mathbf{x}}_r \\ \hat{\mathbf{x}}_{l_1} \\ \vdots \\ \hat{\mathbf{x}}_{l_n} \end{pmatrix} \tag{8.1}$$

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_r & \mathbf{P}_{rl_1} & \cdots & \mathbf{P}_{rl_n} \\ \mathbf{P}_{rl_1}^T & \mathbf{P}_{l_1} & \cdots & \mathbf{P}_{l_1 l_n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{rl_n}^T & \mathbf{P}_{l_1 l_n}^T & \cdots & \mathbf{P}_{l_n} \end{pmatrix} \tag{8.2}$$

Similarly, let $\hat{\mathbf{y}}$ represent a set of m measurements $\{f_1, \ldots, f_n\}$ of environment features observed by a sensor onboard affected by white Gaussian noise $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{S})$

$$\hat{\mathbf{y}} = \mathbf{y} + \mathbf{u} \tag{8.3}$$

where $\mathbf{y}$ is the theorical value of the observations. Thus, $\hat{\mathbf{y}}$ and the covariance matrix $\mathbf{S}$ are

$$\hat{\mathbf{y}} = \begin{pmatrix} \hat{\mathbf{y}}_r \\ \hat{\mathbf{y}}_{f_1} \\ \vdots \\ \hat{\mathbf{y}}_{f_m} \end{pmatrix} \tag{8.4}$$

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}_r & \mathbf{S}_{rf_1} & \cdots & \mathbf{S}_{rf_m} \\ \mathbf{S}_{rf_1}^T & \mathbf{S}_{f_1} & \cdots & \mathbf{S}_{f_1 f_m} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{S}_{rf_m}^T & \mathbf{S}_{f_1 f_m}^T & \cdots & \mathbf{S}_{f_m} \end{pmatrix} \tag{8.5}$$

A measurement f_i and its corresponding landmark l_{j_i} are related by an *implicit measurement function* of the form

IMPLICIT MEASUREMENT FUNCTION

$$\mathbf{f}_{ij_i}(\mathbf{x}, \mathbf{y}) = \mathbf{0} \tag{8.6}$$

which states that the relative location between the measurement and the corresponding landmark must be null.

The purpose of the data association process is to generate a hypothesis $\mathcal{H}_m = \{j_1, \ldots, j_m\}$ that pairs each measurement f_i with a landmark l_{j_i} of the map. Data association algorithms must select in some way one of all hypotheses carrying out validations to determine the compatibility between measurements and landmarks.

INTERPRETATION TREE

The solution space is exponential an can be represented as an *Interpretation Tree* of m levels [63], to which can be applied searching techniques from the Artificial Intelligence field, as it is done by the JCBB algorithm described in Section 8.4.2.

INDIVIDUAL COMPATIBILITY NEAREST NEIGHBOR

If a single measurement is considered, GNN is called *Individual Compatibility Nearest Neighbor* (ICNN). It simply pairs each measurement with the landmark considered most compatible according to (8.6). Since the implicit measurement function is non-linear usually, linearization around the current estimation is necessary

$$\mathbf{f}_{ij_i}(\mathbf{x}, \mathbf{y}) \simeq \mathbf{h}_{ij_i} + \mathbf{H}_{ij_i}(\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{G}_{ij_i}(\mathbf{y} - \hat{\mathbf{y}}) \tag{8.7}$$

where

$$\mathbf{h}_{ij_i} = \mathbf{f}_{ij_i}(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \tag{8.8}$$

$$\mathbf{H}_{ij_i} = \left.\frac{\partial \mathbf{f}_{ij_i}}{\partial \mathbf{x}}\right|_{(\hat{\mathbf{x}}, \hat{\mathbf{y}})} \tag{8.9}$$

$$\mathbf{G}_{ij_i} = \left.\frac{\partial \mathbf{f}_{ij_i}}{\partial \mathbf{y}}\right|_{(\hat{\mathbf{x}}, \hat{\mathbf{y}})} \tag{8.10}$$

The vector $\mathbf{h}_{ij_i}$ represents the innovation of the paring between f_i and l_{j_i}. Its covariance can be obtained from (8.6) and (8.7) as

$$\begin{aligned} \mathbf{C}_{ij_i} &= \mathbf{H}_{ij_i} \mathrm{cov}\left(\mathbf{x} - \hat{\mathbf{x}}\right) \mathbf{H}_{ij_i}^T + \mathbf{G}_{ij_i} \mathrm{cov}\left(\mathbf{y} - \hat{\mathbf{y}}\right) \mathbf{G}_{ij_i}^T \\ &= \mathbf{H}_{ij_i} \mathbf{P} \mathbf{H}_{ij_i}^T + \mathbf{G}_{ij_i} \mathbf{S} \mathbf{G}_{ij_i}^T \end{aligned} \tag{8.11}$$

INDIVIDUAL COMPATIBILITY

The *individual compatibility* (IC) between f_i and l_{j_i} can be determined using an innovation test that measures the Mahalanobis distance

$$D_{ij_i}^2 = \mathbf{h}_{ij_i}^T \mathbf{C}_{ij_i}^{-1} \mathbf{h}_{ij_i} < \chi_{d,\alpha}^2 \tag{8.12}$$

where $d = \dim\left(\mathbf{f}_{ij_i}\right)$ and α is the desired confidence level. This χ^2 test, applied to the predicted state, determines the subset of landmarks that are compatible with a measurement f_i.

Finally, the *nearest neighbor* selection criterion for a given measurement consists in choosing among the landmarks that satisfy (8.12), the one with the smallest Mahalanobis distance.

8.3 Maximum Likelihood

MAXIMUM LIKELIHOOD

Montemerlo and Thrun propose the *Maximum Likelihood* approach for the data association problem in SLAM [92, 93, 94, 96, 149]. In particular, they propose to choose the data association n_t that maximizes the likelihood of the sensor measurement z_t given all available data

$$\hat{n}_t = \arg\max_{n_t} p(z_t \mid s^t, z^{t-1}, u^t, \hat{n}^{t-1}) \tag{8.13}$$

The term $p(z_t \mid s^t, z^{t-1}, u^t, \hat{n}^{t-1})$ is referred to as a *likelihood*, and this approach is an example of a *Maximum Likelihood* (ML) estimator. Such data association is also known as *Nearest Neighbor* data association, interpreting the negative log likelihood as a distance function. For Gaussians, the negative log likelihood is the Mahalanobis distance and the ML estimator selects data associations by minimizing it. The reader might recall these concepts from the GNN data association method described in the previous section, which are detailed in the mathematical derivation.

The most common approach to data association in SLAM is to assign each observation using a Maximum Likelihood rule [93], i.e. each observation is assigned to the landmark most likely to have generated it. If the maximum probability is below some fixed threshold p_0, the observation is considered as a new landmark that must be incorporated to the map.

ML data association works well when the correct data association is significantly more probable than the incorrect associations. However, if the measurement uncertainty is high, more than one data association will receive high probability. If a wrong data association is picked, this decision can have a catastrophic result on the accuracy of the resulting map [93, 103]. Such data association ambiguity can be induced easily if the sensors are very noisy. One approach to this problem is to only incorporate observations that lead to unambiguous data associations, but this is unfeasible in noisy environments because many observations will go unprocessed.

Algorithm 18 Maximum Likelihood

Require: Measurement z_t, control u_t, pose x_t and the N_{t-1} features known (map) with their mean $\mu_{j,t-1}$ and covariance $\Sigma_{j,t-1}$, for $j = 1, \ldots, N_{t-1}$.
A measurement prediction function $h(\mu_{t-1}, x_t)$, its Jacobian $h'(\mu_{t-1}, x_t)$ and the measurement model noise Q_t.
The importance factor p_0 used for new features.
Ensure: Weight w and index $\hat{c}$ of maximum likelihood correspondence or feature.
Algorithm: MaximumLikelihood$(z_t,\ u_t,\ x_t,\ N_{t-1})$ **return** $\langle w, \hat{c} \rangle$

1: **for** $j = 1$ **to** N_{t-1} **do** ▷ measurement likelihoods
2: $\hat{z}_j = h(\mu_{j,t-1}, x_t)$ ▷ measurement prediction
3: $H_j = h'(\mu_{j,t-1}, x_t)$ ▷ calculate Jacobian
4: $Q_j = H_j \Sigma_{j,t-1} H_j^T + Q_t$ ▷ measurement covariance
5: $w_j = \left|2\pi Q_j\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - \hat{z}_j)^T Q_j^{-1}(z_t - \hat{z}_j)\right\}$ ▷ likelihood correspondence
6: **end for**

7: $w_{N_{t-1}+1} = p_0$ ▷ importance factor, new feature
8: $w = \max_{j=1,\ldots,N_{t-1}+1} w_j$ ▷ max likelihood correspondence
9: $\hat{c} = \arg\max_{j=1,\ldots,N_{t-1}+1} w_j$ ▷ index of ML feature
10: **return** $\langle w, \hat{c} \rangle$

The FastSLAM algorithm discussed in this thesis uses an EKF to estimate the location of each landmark present in the environment. The probability of the Maximum Likelihood estimation for a particular data association is computed with the EKF equations commented in Chapter 3. Therefore, the data association is ob-

tained with the Algorithm 18. It applies the EKF to the mean $\mu_{j,t}$ and covariance $\Sigma_{j,t}$ of each landmark $j = 1, \ldots, n$ of the map. The probability of each possible data association is saved in line 5 to later retrieve the data association with higher probability in lines 7-9. If the maximum probability is below the threshold p_0 given in line 7, the observation is incorporated into the map as a new landmark. On the other hand, if a particular observation is assigned to different landmarks, none of the data associations are considered [93].

The measurement model is assumed to be affected by white Guassian noise with mean 0 and covariance matrix Q_t, as shown in line 4. The actual noise depends on the sensor and it yields a covariance matrix of the form

$$Q_t = \begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix} \tag{8.14}$$

that represents the noise over the range ρ and bearing θ of the observations. Juan Nieto and Tim Bailey recomend the initial values below

$$\sigma_\rho^2 = 0.1^2 = 0.01 \tag{8.15}$$

$$\sigma_\theta^2 = \left(\frac{\pi}{180}\right)^2 \approx 0.0003 \tag{8.16}$$

in the SLAM Package of Tim Bailey [136].

8.3.1 Mathematical Derivation

We will now derive the mathematical expression to compute the data associations n_t using the ML approach. As usual, the reader not interested in the mathematical details is invited to skip this section. Anyway, it is encouraged to read the GNN mathematical derivation given in Section 8.2.1 before this one.

In the case of the EKF, the probability of the observation can be written as a function of the difference between the observation z_t and the expected observation
INNOVATION $\hat{z}_{n_t}$ for the landmark n_t [93]. This difference is known as *innovation* and it also appears in the GNN derivation. Let n_t be the associations vector that pairs each measurement with a landmark in the map at time t, the path s^t and the controls u^t accomplished by the robot, and $\hat{n}^{t-1}$ the associations vector up to $t-1$, the data association heuristic of (8.13) yields

$$\hat{n}_t = \arg\max_{n_t} p(z_t \mid s^t, z^{t-1}, u^t, \hat{n}^{t-1}) \tag{8.17}$$

$$= \arg\max_{n_t} \frac{1}{\sqrt{\left|2\pi Z_t\right|}} \exp\left\{-\frac{1}{2}(z_t - \hat{z}_{n_t})^T Z_t^{-1}(z_t - \hat{z}_{n_t})\right\} \tag{8.18}$$

$$= \arg\max_{n_t} \left|2\pi Z_t\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - \hat{z}_{n_t})^T Z_t^{-1}(z_t - \hat{z}_{n_t})\right\} \tag{8.19}$$

which gives the data association $\hat{n}_t$ that maximizes the likelihood of observe the measurement z_t given all avialable data. The covariance matrix Z_t is obtained computing the line 4 of Algorithm 18, where it is denoted by Q_j for the landmark j. The expression (8.19) is often reformulated in terms of the negative log likelihood, as follows

$$\hat{n}_t = \arg\min_{n_t} \ln\left|Z_t\right| + (z_t - \hat{z}_{n_t})^T Z_t^{-1}(z_t - \hat{z}_{n_t}) \tag{8.20}$$

where the second term is known as Mahalanobis distance, a distance metric normalized by the covariances of the observation and the landmark estimate, i.e. the *innovation*. For this reason, data association using this metric is often referred to as *Nearest Neighbor* data association [9].

8.4 Other Data Association Methods

There exist a number of robust data association methods in the literature that might produce better results that the techniques discussed thus far. Here we describe briefly some of the most common methods as they appear in [93].

8.4.1 Local Map Sequencing

LOCAL MAP SEQUENCING

The *Local Map Sequencing* (LMS) technique is aimed to build indoor maps using sonar data [139]. The algorithm collects sonar readings as the robot moves over a short distance and they are processed by two Hough Transforms that detect corners and line segments in the robot's vicinity given the entire set of observations. These features are used to build a local map. Multiple local maps are pieced together to build a global map of the environment.

Data association is robust because multiple sensor readings, taken from different robot poses, vote to determine the correct interpretation of the data. Alternatively, RANSAC is suggested by the authors as another voting algorithm [139].

8.4.2 Joint Compatibility Branch and Bound

If multiple observations are gathered per control, the Maximum Likelihood approach treats each data association decision as an independent problem. However, the data associations of simultaneous observations are correlated. Moreover, considering the data association of each of the observations separately ignores the issue of mutual exclusion. These problems can be remedied by considering the data association of all of the observations simultaneously.

JOINT COMPATIBILITY BRANCH AND BOUND

That is what the *Joint Compatibility Branch and Bound* (JCBB) algorithm does [9, 100, 105]. Joint data association hypotheses are compared using joint compatibility, a measure of the probability of the set of observations ocurring together. The algorithm traverses the *Interpretation Tree* [62] that contains all possible joint correspondences. Many hypotheses can be excluded without traversing the entire tree, reducing the computational cost.

INTERPRETATION TREE

8.4.3 Combined Constraint Data Association

COMBINED CONSTRAINT DATA ASSOCIATION

A similar algorithm to JCBB called *Combined Constraint Data Association* (CCDA) constructs an undirected graph of data association constraints coined *Correspondence Graph* [7]. Each node represents a candidate pairing of observed features and landmarks, possibly determined using a Nearest Neighbor test. Edges between nodes represent joint compatibility between pairs of data associations. The set of joint data associations that correspond to the largest clique is retrieved. Although the result of JCBB and CCDA are similar, the later is able to determine data associations even when the robot pose is completely unknown.

8.4.4 Iterative Closest Point

SCAN MATCHING
ITERATIVE CLOSEST POINT

The *Scan Matching* method is a data association technique based on a modified version of the *Iterative Closest Point* (ICP) algorithm [12, 65, 84]. This algorithm alternates a step in which new correspondences between data are identified and a step in which a new robot path is recovered. Such iterative optimization is similar to Expectation Maximization and RANSAC [93].

8.4.5 Multiple Hypothesis Tracking

MULTIPLE HYPOTHESIS TRACKING

All data association methods presented thus far choose a single data association hypothesis. Contrary, the *Multiple Hypothesis Tracking* (MHT) algorithm maintains a set of hypothesized tracks of multiple targets [117]. If an observation has multiple data associations, new hypotheses are created for each correspondence. Some authors *pause* map-building when data association becomes ambiguous to reduce the computational cost [99].

8.5 Discussion

The data association methods discussed above are presented in the mindmap of Figure 8.3. In this thesis we employ the Maximum Likelihood technique because it is proposed by the authors of the FastSLAM algorithm [93] used in the present work. In the literature, it is common to consider a single observation per control, but we will manage multiple observations in the straightforward way, i.e. considering observation independence. Such assumption is actually wrong, as mentioned in Section 8.4.2 where the JCBB algorithm is presented as a possible solution to the problem.

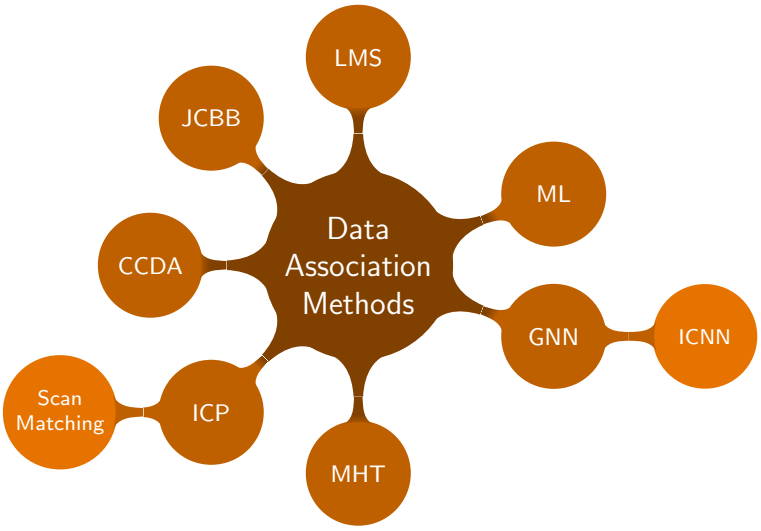


Figure 8.3: Data Association Methods

Most robust data association methods exploit geometric and probabilistic constraints among the possible data associations, when multiple observations are con-

sidered per control. Other methods consider multiple hypotheses, which is the case of MHT. Nonetheless, the SLAM algorithm might also favor the data association problem. In fact, FastSLAM takes a multi-hypothesis approach where each particle represents a different hypothesized path of the robot, so data association decisions can be made on a per-particle basis [93]. Particles that pick the correct data association will receive high probabilities because they explain the observations well. Particles that pick wrong associations will receive low probabilities and be removed.

While per-particle Maximum Likelihood data association addresses motion ambiguity, it does not address measurement ambiguity. Each observation is paired with the landmark most likely to have generated it, but if the measurement error is high there might be several plausible data associations per observation. In that case, an alternative approach assigns the correspondences probabilistically in accordance to their likelihoods. Such approach can be described as *Monte Carlo Data Association*. An equivalent scheme is proposed by Nieto et al. [103], where all K plausible data associations are enumerated for a given observation and particle. Each particle is then cloned K times and the observation is incorporated with the corresponding data association. Later, the particle set is reduced back to M particles by the resampling process of the FastSLAM algorithm, described thoroughly in Chapter 10.

MONTE CARLO DATA ASSOCIATION

Maximum Likelihood and Gated Nearest Neighbor are the most common data association techniques employed in SLAM and they suffices to solve the SLAM problem in not so large indoor environments. On the other hand, outdoor and dynamic environments demand most robust data association algorithms. For instance, the Iterative Closest Point has shown significant promise for data association in environments with large loops [93]. The application of such algorithms is outside the scope of this work, but they might improve the accuracy of the resulting map and allow the applicability in more complex environments.

8.6 Bibliographical and Historical Remarks

The data association problem is widely discussed in the literature, but we recomend the dissertations [93, 96] of Montermerlo since they are concerned with the SLAM problem. Furthermore, it proposes the Maximum Likelihood estimation that is part of the FastSLAM algorithm examined in the present work. In fact, a work of Nieto [103] describes ML as a real time data association method for FastSLAM. A concise mathematical derivation might be consulted in [93], but there is more information in the literature of the sibling Gated Nearest Neighbor method [9].

GNN was originally applied to tracking problems [9], but is has been studied in much more depth in contemporay texts due to Montemerlo and Thrun [92, 93, 94, 96, 149], among others. The stochastic mapping derivation given in this chapter is attributed to Smith [132].

Montemerlo has an updated list of robust data association methods [93]. Tardós et al. treats the data association problem with sonars in [139], that proposes the Local Map Sequencing method. A method aimed to deal with multiple observations per control coined Joint Compatibility Branch and Bound was developed by Neira in [100]. Some texts addressing this approach are due to Bar-Shalom [9] and Oussalah [105]. This method traverse the Interpretation Tree of all possible joint correspondences, which is studied by Grimson in [62, 63]. Later, Bailey presented a similar data association algorithm called Combined Constraint Data Association in [7].

The Iterative Closest Point algorithm is promising for outdoor environments and closing large loops. There exist a number of texts [12, 65, 84] that consider the Scan Matching

algorithm based on it. A multiple-hypothesis alternative is due to Reid [117]. It is known as Multiple Hypothesis Tracking in the target tracking literature. In the case of ambiguous data association Nebot propose to pause map-building temporary [99]. To some extend, the per-particle data association of the FastSLAM and Maximum Likelihood combination is similar in spirit to MHT.

Chapter 9

SLAM

9.1 Introduction

The ability of *mapping* is one of the core competencies of truly autonomous robots. For this reason is a very active research topic [93, 136, 147, 148, 149]. Nowadays, there are many robust methods for static, structured environments of limited dimensionality. Mapping dynamic, unstructured and large environments is still an open problem. MAPPING

In some measure, current mapping algorithms based on probabilistic techniques [134]. Some of them operate incrementally as filters and can be used in real time applications, while others are not because they require to process to whole historic of data, even several times. To build accurate maps some algorithms need the exact pose of the robot, which simplifies enormously the problem. In the present work we focus in the more complex problem of mapping without knowing the actual pose of the robot. In this case, it is possible to use the robot odometry, but it is very noise and error prone. Additionally, all mapping algorithm have to deal with the hard problem of data association. Some algorithms require special landmarks that can be identified uniquely. This impose modifications in the environment, that are not generally desirable. In this thesis we will focus in methods that can map the environment with unknown data association though.

We will introduce mapping first, to later describe the problem of mapping with unknown robot poses. Such problem is commonly called *Simultaneous Localization And Mapping* (SLAM), for reasons discussed then. A brief descriptions of state-of-the-art SLAM algorithms is given to show the range of solutions and locate the FastSLAM family of algorithms, that is the point at issue of this document.

9.2 Mapping

The field of *cartography* or mapping is focused on the study and development of graphical representations of the environment, typically on a bidimensional surface, but also in three dimensions. The goal of mapping is to obtain an spatial model of the environment using measurements retrieved by sensors onboard. Sensors of any kind are inevitably corrupted by measurement noise, that induce some level of uncertainty in the knowledge of the landmarks forming the environment. Furthermore, all sensors have a limited range of view, so the robot must navigate throughout the CARTOGRAPHY

environment to construct the map. Robot motion is also corrupted by noise and thus the controls commanded does not suffice to obtain the robot pose. These limitations make the mapping problem extremely challenging, since it must estimate both the map and robot poses simultaneously.

Although the earliest maps data back thousands of years BC, it is only in the last decades when automatic mapping has evolved spectacularly, due to important advances in Robotics. The main application of a map is navigation in the environment. Therefore, it is important to manage adequately the measurement and motion uncertainty in order to construct accurate maps, suitable for navigation. The uncertainty is usually managed with probabilistic techniques, that complicate the mathematical foundations and implementation of mapping algorithms to some extent. Moreover, the data association problem of pairing measurements with known landmarks is also highly influenced by the uncertainty induced measurement and motion noise [93, 147]. Thus, sophisticated methods must be considered, such as those discussed in Chapter 8.

Acquiring maps with mobile robots is a challenging problem for a number of reasons. First and foremost, the hypothesis space of all possible maps is huge, since maps are defined over a continuous space. This high-dimensional space makes it challenging to calculate the full posteriors over maps. Learning maps is a *chicken-and-egg* problem, for which reason it is often referred to as *Simultaneous Localization And Mapping* (SLAM). There is a localization problem due to the accumulation of errors in odometry when the robots moves through the environment, which makes it gradually less certain as to where it is. There is also a mapping problem, that it is affected by measurement noise and the error in the estimation of the robot pose, leading to erros in the location of landmarks detected in the environment. In the absence of both an initial map and exact pose information, the robot has to do both simultaneously.

SIMULTANEOUS LOCALIZATION AND MAPPING

The hardness of the mapping problem is the result of a collection of factors, that according with [149] the most important are:

Size The larger the environment relative to the robot's perceptual range, the more difficult it is to acquire a map.

Noise in the perception and actuation The larger the measurement and motion noise, the more difficult the problem.

Perceptual ambiguity The more frequently different places look alike, the more difficult it is to solve the data association problem.

Cycles Cycles or *loops* in the environment are particularly difficult to map. When closing a loop the accumulated error can be reduced, but before it is closed it might be huge. Thus, the higher the cycles, the higher the uncertainty of the map.

The mapping problem under the restrictive assumption that the robot poses are known is usually called [mapping!with known poses]mapping with known poses. The graphical model of this problem can be depicted with the dynamic Bayes network in Figure 9.1. The poses x^t and measurements z^t are known, and the goal of mapping is to recover the map m. Under this assumption the problem is quite straightforward and it usually solved with a family of algorithms callectively called *occupancy grid mapping*. They address the problem of generating consistent maps

OCCUPANCY GRID MAPPING

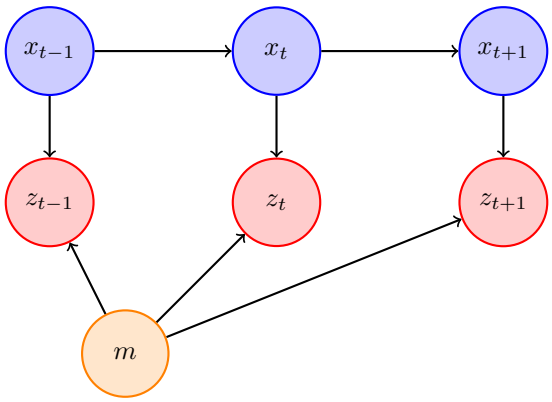


Figure 9.1: Bayes network of Mapping with known poses

from noisy and uncertain measurement data, given the robot pose. The main utility of the occupancy grid technique is in post-processing, since many of the SLAM algorithms do not generate maps fit for path planning and navigation. Occupancy grid maps are often used after solving the SLAM problem by some other means, and taking the resulting path estimates for granted.

Finally, note that mapping usually takes place in conjunction with other tasks that might complicate the process even more, such as exploration. Exploration is a planning problem that establishes where the robot must go to maximize its knowledge about the external world, and possibly other aspects —e.g. time, energy consumption, etc. It is common to construct the map while the robot is exploring. In such a case, it is mandatory to perform mapping in real time, so not all mapping algorithms are applicable.

9.2.1 Map Representation

Since the 1980s, mapping research has been divided in two approaches. Those who construct *metric maps* that represent the geometric features of the environment, and the supporters of *topologic maps*, which describe the connectivity between different places in the environment. One of the first metric approaches was the occupancy grid algorithms of Elfes and Moravec [47, 48, 97], which are explained in the next section. Topologic maps represent the environment by means of list of representative places interconnected through arcs in a graph. Such arcs are typically annotated with information that specify how to navigate from one place to another. Examples of topologic map are due to Mataríc [89], Kuipers [79] and most contemporary works [13, 14, 49].

METRIC MAPS

TOPOLOGIC MAPS

Nevertheless, the difference between topologic and metric maps is somewhat diffuse, since topologic maps actually rely on geometric information. In practice, metric maps provides fine-grain information if compared with topologic maps. Although this comes at some computational cost, it helps solve the complex problem of data association discussed in Chapter 8. In the literature, we find works that combine both approaches [74].

Maps might also be classify according with the frame of reference. As shown in Figure 9.2, we distinguish between maps centered in the world and maps centered in the robot. The former use a global frame and therefore there is no information

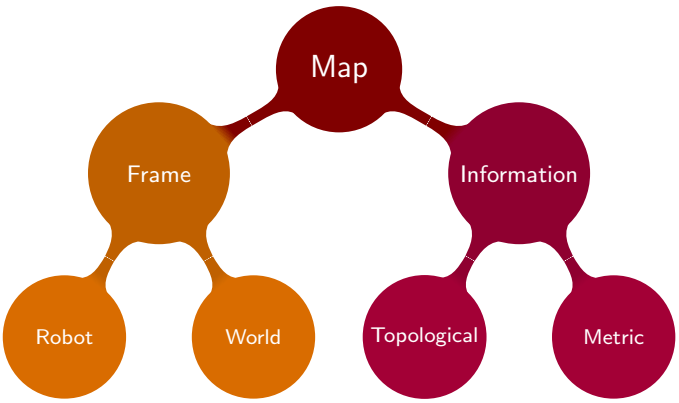


Figure 9.2: Map Classification according with the frame and information represented in the map

about the original measurementes, contrary to the later approach. In the present, most authors use maps are centered in the world, that is, they use a global frame.

Most of the mapping algorithms assume the environment is static, so they are not capable of manage dynamic phenomena. There exist some modifications that allow some basic types of changes in the environment, such as the state of the door —i.e. close or open. However, this is a problem that has been explored poorly and there is no formal approach to deal with it.

9.2.2 Occupancy Grid Maps

OCCUPANCY GRID MAP

The *occupancy grid map* is credited to Elfes and Moravec [47, 48, 97]. It calculates the posterior over maps given the measurements z^t and poses x^t up to time t

$$p(m \mid z^t, x^t) \tag{9.1}$$

The control u^t play no role, since the path is already known.

The types of maps considered by occupancy grid maps are fine-grained grids defined over the continuous space of locations. The most common domain of occupancy grid maps are 2-D floor plan maps, as the one shown in Figure 9.3, which are the representation of choice when a robot navigates on a flat surface.

Let m_i denote the grid cell with index i. An occupancy grid map partitions the space into finitely many grid cells

$$m = \{m_i\} \tag{9.2}$$

where each m_i has attached to it a binary occupancy value that specifies whether a cell is occupied or free.

The standard occupancy grid approach breaks down the problem of estimating the map into a collection of separate problems, namely that of estimating

$$p(m_i \mid z^t, x^t) \tag{9.3}$$

for all grid cell m_i. Each of these estimation problems is a binary problem with static state. The posterior over maps is approximated as the product of its marginal

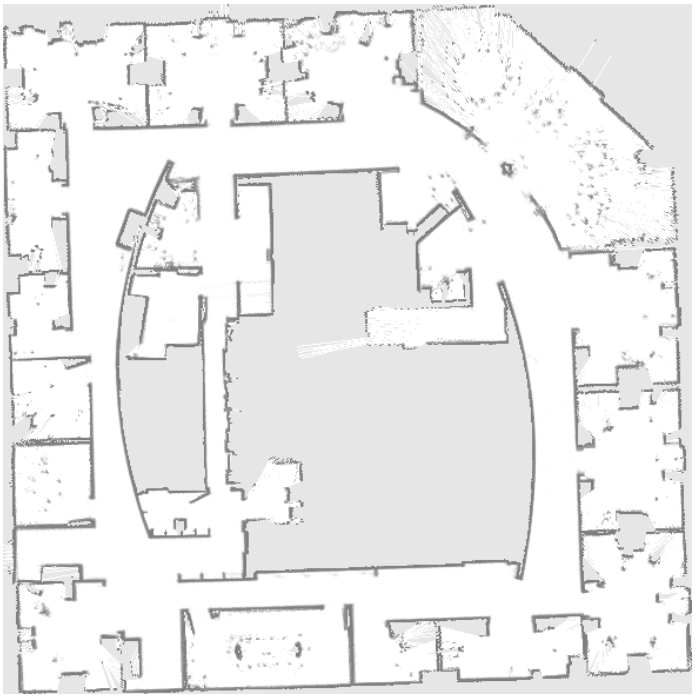


Figure 9.3: Occupancy grid map. Interior of the Intel Research Lab in Seattle

$$p(m \mid z^t, x^t) = \prod_i p(m_i \mid z^t, x^t) \tag{9.4}$$

This problem can be solved with the binary Bayes filter. As the original filter, the occupancy grid mapping algorithm uses the *log odds* representation of occupancy. LOG ODDS The advantage of the log odds over the probability representation is that we can avoid numerical instabilities for probabilities near to zero or one. Anyway, the probabilities are easily recovered from the log odds ratio. Thus, occupancy grid maps are probabilistic maps, which generate consistent metric maps from nosiy measurement data z^t. They are often used to represent the resulting map obtained with SLAM algorithms, since they provide a metric representation that can be of direct use for navigation and planning. The reader interested in the mathematical and implementation details might is encourage to consult [149].

9.2.3 Feature-based Maps

There exist a number of mapping algorithms that build maps composed of geometric elements or objects, such as points, lines, doors, walls, etc. The first approximation is attributed to Chatila and Laumond [28]. They propose to represent maps as a collection of lines instead of an occupancy grid, as in Figure 9.4, where the environment is represented with lines. Since then, some approaches have considered maps composed of objects present in the environment. *Feature-based maps* have FEATURE-BASED MAPS some important advantages over occupancy grid maps:

1. The maps is more compact, particularly in structured environments.

2. Depending on the environments they are more accurate.
3. It is possible to represent dynamic environments
4. Feature-based maps are conceptually easier to understand, since they represent elements of high semantic level. For this reason, they facilitates human-robot interaction.

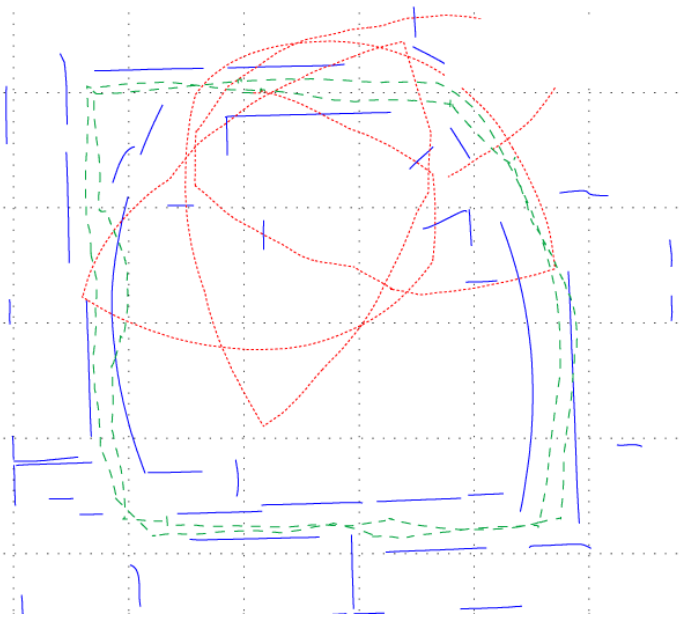


Figure 9.4: Feature-based map. Map that represent the environment with line features in blue (___). Interior of the Intel Research Lab in Seattle

The major drawback of feature-based maps is that of the assumption that the environment can be represented with a particular set of features. For this reason, the kind of features used depends significantly of the environment that the robot is mapping. In fact, the environment must be such that it can be represented by simple geometric shapes or objects.

9.3 The SLAM problem

It was in the 1990s when mapping embrace probabilistic techniques, following a probabilistic framework introduced by Smith, Self and Cheeseman [132, 133, 134] in a series of papers that develop an approach to solve the mapping and localization problem simultaneously. They identified the relationship between localization and mapping and since then the problem is known as *Simultaneous Localization And Mapping* (SLAM).

If the path of the robot were known, then mapping would be a straightforward problem. The positions of landmarks in the environment could be estimated using independent filters. However, when the path of the robot is unknown, errors in the robot path correlates error in the map. As a result, the state of the robot pose and

the locations of the landmarks in the map must be estimated *simultaneously*, hence the SLAM problem.

The most popular online solutions to the SLAM problem attempt to estimate the posterior probability distribution over all possible maps Θ and robot poses x_t conditioned on the full set of control u^t and measurements z^t at time t. Using this notation, the joint posterior distribution over maps and robot poses can be written as

$$p(x_t, \Theta \mid z^t, u^t) \tag{9.5}$$

This distribution is referred to as the SLAM posterior and it allows to solve the online SLAM problem, since it can operate as a filter, in real time.

In the probabilistic formulation of SLAM, the map Θ consists on a collection of landmarks θ_i for $i = 1, \ldots, N$, where N is the number of landmarks in the map. The pose x_t consists on the robot location $(x\ y)$ and its angular orientation θ. The complete trajectory of the robot, consisting of the robot pose at every time step up to t is denoted x^t. If we compute the posterior over maps and robot *paths*

$$p(x^t, \Theta \mid z^t, u^t) \tag{9.6}$$

we are solving the full SLAM problem, that is the problem of obtaining the map and the whole set of robot poses, not only the last one.

The robot perceives the environment with measurement sensors. A measurement at time t is denoted z_t. Without loss of generality, we assume the observation of a single landmark θ_t in the map per measurement z_t, whose association is identified by n_t. The process that models the acquisition of a measurement is the measurement model

$$p(z_t \mid x_t, \theta_{n_t}, n_t) = g(\theta_{n_t}, x_t) + \varepsilon_t \tag{9.7}$$

The generative measurement model is conditioned by the robot pose x_t, the identity n_t of the landmarks θ_{n_t} detected and the landmark itself. The model is governed by a deterministic function g with added Gaussian noise ε_t with mean zero and covariance R_t, which is a common and appropiate assumption [92, 147, 149].

A second source of information to solve the SLAM problem are the controls u_t executed by the robot throughout the time interval $[t-1, t)$. The evolution of the robot pose is governed by the motion model

$$p(x_t \mid x_{t-1}, u_t) = h(x_{t-1}, u_t) + \delta_t \tag{9.8}$$

The pose x_t at time t is given by a deterministic function h, that depends on the previous pose x_{t-1} and the controls commanded u_t, with added Gaussian noise δ_t with mean zero and covariance P_t. Habitually, the functions g and h are highly nonlinear.

Most contemporary SLAM algorithms based on Bayes filters that compute recursively the probability distribution of the map Θ and robot pose x_t from the prior distribution

$$p(x_t, \Theta \mid z^t, u^t, n^t) = \eta p(z_t \mid x_t, \Theta, n_t) \int p(x_t \mid x_{t-1}, u_t)\ p(x_{t-1}, \Theta \mid z^{t-1}, u^{t-1}, n^{t-1})\ dx_{t-1} \tag{9.9}$$

Here η is a normalizer factor that does not depends on the distributions —in fact, $\eta = p(z_t \mid z^{t-1}, u_t, n_t)$. If the functions g and h are linear, this approach is

equivalent to the Kalman filter. Otherwise, the Extended Kalman filter approximate g and h applying Taylor expansion. The FastSLAM algorithm factors the filter to compute it as a product of smaller and simpler terms. These and other SLAM algorithms are described in brief in the following section.

9.4 SLAM algorithms

The following sections enumerate a number of classical and different approaches to solve the SLAM problem. We describe solutions that based on techniques radically different to illustrate the great effort of research that this topic has attracted and the variety of perspectives to attack it [93, 149].

9.4.1 EKF SLAM

Many of the original SLAM algorithms originate from a seminal paper by Smith and Cheeseman [132, 133, 134], which proposed the use of the Extended Kalman Filter (EKF) to estimate the SLAM posterior. The EKF represents the SLAM posterior as a high-dimensional, multivariate Gaussian parametrized by a mean μ_t and a covariance matrix Σ_t. The mean describes the most likely stat of the robot and landmarks, while the covariance matrix encodes the pairwise correlations between all pairs of state variables.

EKF SLAM

The number of parameters to describe the EKF posterior is quadratic in the number of landmarks N in the map. In the *EKF SLAM* the motion and measurements are linearized around the most-likely state of the system, which is generally a good approximation. EKF-based SLAM has two substantial drawbacks: the quadratic complexity in the number of landmarks N and the sensitivity to failures in data association. There exist a number of powerful variants of the EKF SLAM algorithm that mitigate such limitations, such as Divide & Conquer (D&C) SLAM [110].

9.4.2 Expectation Maximization

EXPECTATION MAXIMIZATION

The *Expectation Maximization* (EM) algorithm is a stochastic approach developed by Dempster, Laird and Rubin [39]. Currently, it is one of the best solutions to the data association problem. EM algorithms are capable of mapping large and cyclic environments consistently, even if landmarks are similar and cannot be distinguished with the measurement data only.

However, EM does not retain a complete notion of the uncertainty, since it applies an heuristic search of the kind of *hill climbing* or *gradient descent* in the space of all maps to find the most likely one. It does not solve the online SLAM problem because it has to process the available data the several times.

9.4.3 Submap Methods

A great deal of SLAM research has concentrated on developing SLAM algorithms that approximate the performance of the EKF, but scale to much larger environments. The computational complexity of the EKF is due to the fact that the covariance matrix Σ_t represents every pairwise correlation between the state variables. Nonetheless, the observation of a single landmark will have a weak effect

on the positions of distant landmarks. For this reason, many researchers have developed EKF-based SLAM algorithms that decompose the global map into smaller *submaps* [93]. Postponement [36, 64] and the Compressed Extended Kalman Filter (CEKF) [64] are both techniques that delay the incorporation of local information into the global map while the robot stays inside a single submap. The D&C SLAM algorithm commented in Section 9.4.1 is actually a submap SLAM method too.

SUBMAPS

9.4.4 SEIF SLAM

Another filter approach to decomposing the SLAM problem is to represent maps using potential functions between nearby landmarks. One such approach is the *Sparse Extended Information Filter* (SEIF) proposed in [150], which uses the Information Filter instead of the Kalman Filter parametrization. Thus, SEIF update the precision matrix Σ^{-1}. This parametrization is useful because the precision matrix is sparse if correlations are maintained only between nearby landmarks. Under appropiate approximations, this technique has been shown to provide efficient updates with a linear memory requirement [93].

SPARSE EXTENDED INFORMATION FILTER

9.4.5 GraphSLAM

The posterior of the full SLAM problem naturally forms a *sparse graph*. This graph leads to a sum of nonlinear quadratic constraints. Optimizing these constraints yields a Maximum Likelihood map and a corresponding set of robot poses. The *GraphSLAM* algorithm compute the map posterior linearizing the set of constraints to produce an sparse information matrix and later apply standard inference techniques [149].

GRAPHSLAM

GraphSLAM represents the information as a graph of soft constraints. It can acquire maps that are many orders of magnitude larger than EKF SLAM can handle, and they are superior in accuracy. GraphSLAM has also some limitations: the size of the graph grows linearly over time., it calculates posteriors over robot paths, hence is not an incremental algorithm, and it requires inference to compute data association probabilities. Thus, to some extent, GraphSLAM compared to EKF SLAM are extreme ends of the spectrum of SLAM [149].

9.4.6 Thin Junction Tree Filter

The *Thin Junction Tree Filter* (TJTF) of Paskin [108] is a SLAM algorithm based on the same principle as the SEIF. It maintains a sparse network of probabilistic constraints between state variables, which enables efficient inference. The SLAM posterior is represented with a graphical model called Junction Tree. The size of this tree grows as new landmarks are incorporated to the map, but it can be *thinned* using an operations called variable contraction. TJTF has the advantage over SEIF that global maps can be extracted without any matrix inversion. This algorithm requires linear computation, which can be reduced to constant time with further approximation [93].

THIN JUNCTION TREE FILTER

9.4.7 Covariance Intersection

Julier and Uhlmann present an alternative to maintaining the complete joint covariance matrix called *Covariance Intersection* [72]. It updates the landmark position

COVARIANCE INTERSECTION

variances conservatively, in such a way that allows for all possible correlations between the observation and the landmark. The resulting algorithms requires linear time and memory. The drawback of this approach is that it tends to be extremely conservative, leading to notably slow convergence and highly ambiguous data association.

9.4.8 Graphical Optimization Methods

Commonly, offline SLAM algorithms treats the SLAM as an optimization problem. These methos exploits the fact that the set of constraints between different variables in the SLAM posterior can be represented by a set of sparse links, given all past poses. Optimization techniques leads to a most likely map and robot path. The earliest work on this paradigm is credited to Lu and Milios [83], but were Folkesson and Christensen who introduced the term *Graphical SLAM* into the literature [52], for a related graphical relaxation method. Graphical optimization methods are also discussed in [149], which are related with the Structure From Motion literature [153], since they are usually offline.

GRAPHICAL SLAM

9.4.9 Hybrid Methods

There exist a number of hybrid algorithms that try to solve the SLAM problem. They are hybrid in the sense that they combine different approaches. Most of them integrate probabilistic computations with Maximum Likelihood estimators, which are much more efficient computationally. One of this approaches is the *FastSLAM* algorithm [93, 94] discussed in this work, since it uses Extended Kalman filters to estimate landmarks locations in the map, a particle filter to estimate the robot path and Maximum Likelihood estimation to solve the data association problem. Doing so, it is capable to manage large environment in real time.

FASTSLAM

FastSLAM Algorithms

The FastSLAM family of algorithms uses particle filters to solve the SLAM problem, in such a way that motion uncertainty is approximated with several hypotheses, one for each possible robot path. Each path generates its own map, where each landmark is estimated using Extended Kalman filters. This approach is computationally efficient and it allows to obtain maps with unprecedented large dimensionality and accuracy. In fact, FastSLAM has a $\mathcal{O}(\log N)$ computational cost, where n is the number of landmarks. Therefore, the algorithm scales efficiently to large maps and is robust to significant ambiguity in data association, since it manages multiple hypotheses to estimate the robot pose and data associations.

One of the major drawbacks of FastSLAM is the underestimation of the uncertainty, that might lead to incorrect data association, specially when closing loops. The original version FastSLAM 1.0 suffers this and other problems, that a later version coined FastSLAM 2.0 addresses. There exist also a version that does not rely on features. Called *Grid-based FastSLAM* [93, 149], it estimates an occupancy grid map instead of a set of landmarks. In Chapter 10, the FastSLAM algorithm is described with further detail, since it is the SLAM algorithm used in the present work.

GRID-BASED FASTSLAM

9.5 Discussion

The SLAM algorithms described above are shown in Figure 9.5, where the FastSLAM algorithm appears classified as an hybrid methods because it is approach we have selected to solve the SLAM problem in this thesis. All SLAM algorithms have their *pros and cons*. In the present work we intend to solve the online SLAM problem, that is, we need an approach feasible to operate in real time.

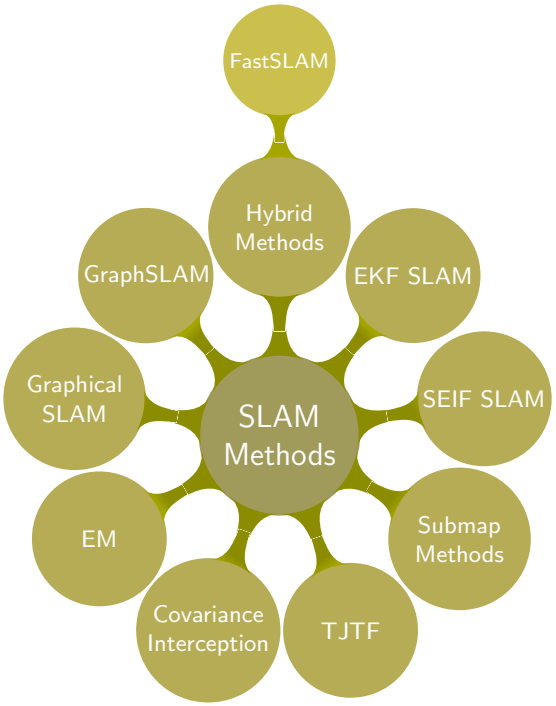


Figure 9.5: SLAM Methods

We have introduced some SLAM methods suitable for real time, that generate maps with sufficient accuracy. Although any option might be as valid as the others, we have adopted FastSLAM because it is very robust and efficient, produces accurate maps, and it is relatively easy to implement if compared with other methods.

Chapter 10

FastSLAM

10.1 Introduction

In this chapter we describe the basic FastSLAM algorithm, an alternative approach to SLAM that is based on particle filtering [93]. The algorithm is based on an important characteristic of the SLAM problem. The full SLAM problem with known correspondences c^t possesses a contional independence between any two disjoint sets of landmarks given the robot pose. Therefore, we can estimate the location of all landmarks independently of each other given the true robot path. This structural observation makes it possible to apply a verion of particle filters to SLAM known as *Rao-Blackwellized Particle filter*, which uses particles to represent the posterior over some variables, along with Gaussians to represent all other variables [149].

RAO-BLACKWELLIZED PARTICLE FILTER

Particularly, FastSLAM uses particle filters for estimating the robot path. For each of these particles the individual map landmarks are conditionally independent. Hence the mapping problem can be factored into many separate problems, one for each landmark, whose location is estimated by low-dimensional EKFs in FastSLAM. The basic algorithm can be implemented in time logarithmic $\mathcal{O}(logN)$ in the number of landmarks N. There exist two versions of FastSLAM. FastSLAM 1.0 being the first version is improved by FastSLAM 2.0, which attempts to solve the limitations of the former. In brief, it incorporates the current observation z_t into the proposal distribution in order to better match the posterior.

The use of particle filters creates the unusual situation that FastSLAM solves both the *full SLAM problem* and the *online SLAM problem*. As we shall see, FastSLAM is formulated to calculate the full path posterior, since only the full path renders feature locations conditionally independent. However, because particle filters estimate one pose at-a-time, FastSLAM is indeed an online algorithm. Hence it also solve the online SLAM problem. Among all SLAM algorithms discussed thus far, FastSLAM is the only algorithm that fits both categories [149]. Furthermore, particle filters can cope with non-linear robot motion models, whereas other SLAM techniques approximate such models via linear functions. This is important when the kinematics are highly non-linear or when the pose uncertainty is relatively high.

This chapter describes several instantiations of the FastSLAM algorithm as they are described in [93, 149]. A version for known data association is presented first and it is later extended to the most general case of unknown data association. The FastSLAM 1.0 algorithm is described, along with some extensions and implementa-

tion concerns. The modified version FastSLAM 2.0 is later discussed.

10.2 The Basic Algorithm

In the basic FastSLAM algorithm each particle contains an estimated robot pose $x_t^{[k]}$, and a set of Extended Kalman filters with mean $\mu_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$, one for each landmark θ_j in the map Θ. Here $[k]$ denotes the index of the particle. As usual, the total number of particles is M. The update step consists on repeating M times the following steps:

Retrieval Retrieve a pose $x_{t-1}^{[k]}$ from the particle set $\mathcal{Y}_{t-1}$.

Prediction Sample a new pose $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$.

Measurement update For each observed feature z_t^i identify the correspondence c_j with a landmark θ_j in the map, and incorporate z_t^i into the the EKF by updating the mean $\mu_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$.

Importance weight Calculate the importance weight $w^{[k]}$ for the new particle.

The temporary particle set obtained is then resampled. The final resampling step consists on sample with replacement M particles, where each particle is sampled with a probability proportional to $w^{[k]}$. Barring the many details of the update step, the algorithm is in large parts identical to the particle filter, as discussed in Section 4.2. The initial step retrieves a particle representing the posterior at time $t-1$, and samples a robot pose using the probabilistic motion model. Next, the EKFs for the observed features are updated. The final steps are concerned with the calculation of an importance weight, which are then used ro resample the particles.

We will now discuss these steps in more details, while the derivation of them is up to the reader decision to be consulted in [93, 94, 149]. Our discussion presupposes that FastSLAM solves the full SLAM problem, not the online problem. However, FastSLAM is a solution to both of these problems, since each particle can be thought of as a sample in path space as required for the full SLAM problem, but the update only requires the most recent pose. For this reason, FastSLAM can be run just like a filter, suitable for real time.

10.3 Factoring the SLAM Posterior

The majority of SLAM approaches are based on estimating the posterior over maps Θ and robot *pose* x_t

$$p(x_t, \Theta \mid z^t, u^t, c^t) \tag{10.1}$$

Contrary, FastSLAM computes the posterior over maps Θ and robot *paths* x^t

$$p(x^t, \Theta \mid z^t, u^t, c^t) \tag{10.2}$$

This subtle difference allows to factor the SLAM posterior into a product of simpler terms

$$p(x^t, \Theta \mid z^t, u^t, c^t) = \underbrace{p(x^t \mid z^t, u^t, c^t)}_{\text{path posterior}} \underbrace{\prod_{n=1}^{N} p(\theta_n \mid x^t, z^t, u^t, c^t)}_{\text{landmark estimators}} \tag{10.3}$$

This factorization, first developed by Murphy [98], states that the SLAM posterior can be separated into a product of a robot path posterior $p(x^t \mid z^t, u^t, c^t)$ and N landmark posteriors $p(\theta_n \mid x^t, z^t, u^t, c^t)$ conditioned on the robot's path. It is important to note that this factorization is exact, since if follows directly from the structure of the SLAM problem [93]. There exist filtering techniques that can computed the path posterior as efficiently as the pose posterior, due to the path posterior factorization obtained [94].

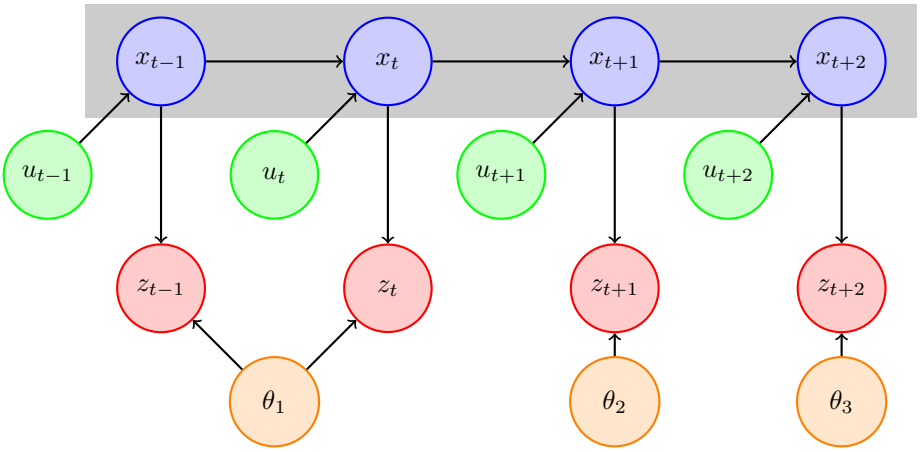


Figure 10.1: SLAM Bayes Network that characterizes the evolution of states x_t, controls u_t, measurements z_t and associated landmarks θ_i

To illustrate the correctness of this factorization, Figure 10.1 shows the data acquisition process graphically, in the form of a Dynamic Bayes Network (DBN). As the graph suggests, each measurement $z_1, \ldots, z_n$ is a function of the position of the corresponding landmark θ_i, along with the robot pose x_t at the time t the measurement z_t was taken. Knowledge of the robot path separates the individual landmark estimation problems and renders them independen of one another, in the sense that no direct path exists in this graphical depiction from one landmark to another that would *not* involve variables on the robot's path. Therefore, knowledge of the exact location of one landmark θ_i will tell us nothing about the locations of other features $\theta_{j \neq i}$, if the robot path is known. This implies that landmarks are *conditionally independent* given the robot path as stated in (10.3). The reader might consult the mathematical proof of the factored SLAM posterior in [93, 149].

10.4 FastSLAM with Known Data Association

We begin considering that the correspondences c^t are known and so is the number of landmarks N observed thus far. This assumption states that the unique correspondence between a measurement z_t and a landmark θ_t is available. The data association problem is therefore assumed already solved. Such assumption is rarely the case in practice, but it let us introduce the FastSLAM algorithm clearly.

FastSLAM estimates the path posterior $p(x^t \mid z^t, u^t, c^t)$ using a modified particle filter. The landmark locations are estimated using EKFs. Therefore, FastSLAM exploits the factored representation of (10.3) by maintaining $MN + 1$ filters. By doing so, all $MN + 1$ filters are low-dimensional and each update of the filter demands a constant computational cost, regardless the path length. It also allows to manage nonlinear motion models. Each individual EKF is conditioned on a robot path with each particle possessing its own set of N EKFs, and hence each EKF is low-dimensional. In total there are NM EKFs, one for each of the N landmarks in the map and one for each of the M particles of the particle filter. Particles in FastSLAM will be denoted

$$\mathcal{Y}_t^{[k]} = \left\langle x_t^{[k]}, \langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} \rangle, \ldots, \langle \mu_{N_t^{[k]},t}^{[k]}, \Sigma_{N_t^{[k]},t}^{[k]} \rangle \right\rangle \tag{10.4}$$

The bracketed notation $[k]$ indicates the index of the particle, $x_t^{[k]}$ is the path estimate of the robot, and $\mu_{j,t}^{[k]}, \Sigma_{j,t}^{[k]}$ are the mean and covariance of the Gaussian representing the j-th landmark location conditioned on the path $x^{t,[k]}$. Together all of these quantities form the k-th particle $\mathcal{Y}_t^{[k]}$, of which there are a total of M in the FastSLAM posterior.

Filtering, i.e. calculating th posterior at time t from the one at time $t-1$, involves generating a new particle $\mathcal{Y}_t$ from $\mathcal{Y}_{t-1}$. The new particle set incorporates the latest control u_t and measurement z_t with its corresponding data association c_t. This update is performed in the following steps:

1. **Extending the path posterior by sampling new poses.** FastSLAM samples the pose x_t in accordance with each k-th particle by drawing a sample according to the motion posterior

   $$x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t) \tag{10.5}$$

   Here $x_t^{[k]}$ is the posterior estimate for the robot pose at time $t-1$, residing in the k-th particle. This process is applied to all particles to obtain a temporary set of particles.

2. **Updating the observed landmark estimate.** Next, FastSLAM updates the posterior over the landmark estimates, represented by the mean $\mu_{j,t-1}^{[k]}$ and the covariance $\Sigma_{j,t-1}^{[k]}$ for each of the $j = 1, \ldots, N_{t-1}^{[k]}$ landmarks of the k-th particle. The updated values are then added to the temporary particle set, along with the new pose.

   The exact update equation depends on whether or not a landmark θ_j was observed at time t. For $j \neq c_t$ we did *not* observe landmark θ_j and it remains unchanged. For the observed landmark $j = c_t$, the update is specified through the expansion of the posterior using Bayes Rule

   $$p(\theta_{c_t} \mid x^t, z^t, c^t) = \eta p(z_t \mid x^t, \theta_{c_t}, c_t) p(\theta_{c_t} \mid x^{t-1}, z^{t-1}, c^{t-1}) \tag{10.6}$$

   If the observation does *not* correspond to any of the landmarks in the map, a new one is incorporated to the map.

   FastSLAM implements the update equation (10.6) using an EKF. As in EKF solutions to SLAM, this filter uses a linear Gaussian approximation for the

measurement model. As usual, we approximate the measurement function h by Taylor expansion. Under this approximation, the posterior for the location of landmark θ_{c_t} is indeed Gaussian. The new mean and covariance are obteined using the standard EKF measurement update

$$Q_{c_t,t} = H_t^{[k]} \Sigma_{c_t,t-1}^{[k]} H_t^{[k]T} + Q_t \tag{10.7}$$

$$K_t^{[k]} = \Sigma_{c_t,t-1}^{[k]} H_t^{[k]T} Q_{c_t,t}^{-1} \tag{10.8}$$

$$\mu_{c_t,t}^{[k]} = \mu_{c_t,t-1}^{[k]} + K_t^{[k]}(z_t - \hat{z}_t^{[k]}) \tag{10.9}$$

$$\Sigma_{c_t,t}^{[k]} = (I - K_t^{[k]} H_t^{[k]}) \Sigma_{c_t,t-1}^{[k]} \tag{10.10}$$

Steps 1 and 2 are repeated M times, resulting in a temporary set of M particles.

3. **Resampling.** In a final step, FastSLAM resamples this set of particles. It draws from its temporary set M particles with replacement according to an importance weight. The resulting set of M particles then forms the new and final particle set $\mathcal{Y}_t$. The necessity to resample arises from the fact that the particles in the temporary set are not distributed according to the desired posterior, since step 1 generates poses x_t only in accordance with the most recent control u_t, paying no attention to the measurement z_t. Resampling is the common technique in particle filtering to correct for such mismatches and avoid the filter degeneration [5, 41, 42, 43].

   By weighting particles and resampling according to those weights, the resulting particle set approximates the target distribution. To determine the *importance factor*, it will prove usefult to calculate the actual proposal distribution of the path particles in the temporary set. Path particles in the temporary set are distributed according to

   IMPORTANCE FACTOR

$$p(x^{t,[k]} \mid z^{t-1}, u^t, c^{t-1}) = p(x_t^{[k]} \mid x_{t-1}^{[k]}, u_t) p(x^{t-1,[k]} \mid z^{t-1}, u^{t-1}, c^{t-1}) \tag{10.11}$$

   where the factor $p(x_t^{[k]} \mid x_{t-1}^{[k]}, u_t)$ is the sampling distribution used in (10.5). The target distribution takes into account the measurement z_t, along with the correspondence c_t

$$p(x^{t,[k]} \mid z^t, u^t, c^t) \tag{10.12}$$

   The resampling process accounts for the difference of the target and the proposal distribution. The importance factor for resampling is given by the quotient of the target and the proposal distribution

$$\begin{aligned} \omega_t^{[k]} &= \frac{\text{target distribution}}{\text{proposal distribution}} \\ &= \frac{p(x^{t,[k]} \mid z^t, u^t, c^t)}{p(x^{t,[k]} \mid z^{t-1}, u^t, c^{t-1})} \\ &= \eta p(z_t \mid x_t^{[k]}, c_t) \end{aligned} \tag{10.13}$$

   The last transformation is a direct consequence of the following transformation

of the numerator in (10.13), i.e. the target distribution

$$\begin{aligned} p(x^{t,[k]} \mid z^t, u^t, c^t) &\overset{\text{Bayes}}{\propto} \eta p(z_t \mid x^{t,[k]}, z^{t-1}, u^t, c^t) p(x^{t,[k]} \mid z^{t-1}, u^t, c^t) \\ &\overset{\text{Markov}}{=} \eta p(z_t \mid x_t^{[k]}, c_t) p(x^{t,[k]} \mid z^{t-1}, u^t, c^{t-1}) \end{aligned} \tag{10.14}$$

The landmark estimator is an EKF, so this observation likelihood can be computed in closed form. The probability of the observation z_t is equal to the probability of the innovation $z_t - \hat{z}_t$ being generated by a Gaussian with zero mean and covariance $Q_{c_t,t}$, that is the innovation covariance matrix defined in (10.7), which can be written as

$$\omega_t^{[k]} \approx \eta \left|2\pi Q_{c_t,t}\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - \hat{z}_t^{[k]})^T Q_{c_t,t}^{-1}(z_t - \hat{z}_t^{[k]})\right\} \tag{10.15}$$

These three steps together constitute the update rule of the FastSLAM 1.0 algorithm for SLAM problems with known data association. We note that the execution time of the update does not depend on the total path length t. In fact, only the most recent pose $x_{t-1}^{[k]}$ is used in the process of generating a new particle at time t. Consequently, past poses can safely be discarded. Therefore, neither the time requirements nor the memory requirements of FastSLAM depend on t.

A summary of the FastSLAM 1.0 algorithm with known data association is provided in Algorithm 19. For simplicity, this implementation assumes that only a single feature is measured at each point in time. The algorithm implements the various update steps in a straighforward manner.

10.5 Unknown Data Association

This sections extendes FastSLAM algorithms to cases where the correspondences c^t are unknown. A key advantage of using particle filters for SLAM is that each particle can rely on its own, local data association decisions [149]. The reader might recall that the data association problem consists on determining the correspondences c^t at time t based on the available data, which is thoroughly discussed in Chapter 8. So far, we described a number of data association techniques using arguments such as maximum likelihood. Most of those techniques provided a single data association per measurement, for the entire filter. FastSLAM, by virtue of using multiple particles, can determine the correspondence on a per-particle basis, already introduced in Section 8.5. Thus, the filter not only samples over robot paths, but also over possible data association decisions along the way.

FastSLAM uses the Maximum Likelihood data association technique discussed in Section 8.3. With ML each data association $\hat{c}_t^{[k]}$ is determined by maximizing the likelihood of the measurement z_t

$$\hat{c}_t^{[k]} = \arg\max_{c_t} p(z_t \mid c_t, \hat{c}^{t-1,[k]}, x^{t,[k]}, z^{t-1}, u^t) \tag{10.16}$$

ML makes it possible to estimate the number of landmarks in the map. It creates new landmarks if the likelihood falls below a threshold p_0 for all known landmarks in the map.

Algorithm 19 FastSLAM1.0 with known correspondence

Require: Observation or measurement z_t, correspondence c_t, control u_t and the set of M particles $\mathcal{Y}_{t-1}$ obtained in the previous temporal step.
Motion model to sample new pose $x_t \sim p(x_t \mid x_{t-1}, u_t)$.
A measurement prediction function $h(\mu_{t-1}, x_t)$, its Jacobian $h'(\mu_{t-1}, x_t)$, inverse function $h^{-1}(z_t, x_t)$ and the model noise Q_t.
The importance factor p_0 used for new landmarks.
A resampling method to draw M particles with probability $\propto_{k=1,\ldots,M} w^{[k]}$.

Ensure: Set of M particles $\mathcal{Y}_t$ obtained applying FastSLAM 1.0 algorithm.

Algorithm: FastSLAM1.0KC(z_t, c_t, u_t, $\mathcal{Y}_{t-1}$) **return** $\mathcal{Y}_t$

1: **for** $k = 1$ **to** M **do** ▷ loop over all particles
2: retrieve $\left\langle x_{t-1}^{[k]}, N_{t-1}^{[k]}, \langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]} \rangle, \ldots, \langle \mu_{N_{t-1}^{[k]},t-1}^{[k]}, \Sigma_{N_{t-1}^{[k]},t-1}^{[k]} \rangle \right\rangle$ from $\mathcal{Y}_{t-1}$
3: $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$ ▷ sample new pose
4: $N_t^{[k]} = \max \left\{ N_{t-1}^{[k]}, c_t \right\}$ ▷ new number of landmarks in map
5: **if** landmark c_t never seen before **then**
6: $\mu_{c_t,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ ▷ initialize mean
7: $H = h'(\mu_{c_t,t}^{[k]}, x_t^{[k]})$ ▷ calculate Jacobian
8: $\Sigma_{c_t,t}^{[k]} = (H^{-1})^T Q_t H^{-1}$ ▷ initialize covariance
9: $w^{[k]} = p_0$ ▷ default importance weight
10: **else**
11: $\hat{z} = h(\mu_{c_t,t-1}^{[k]}, x_t^{[k]})$ ▷ measurement prediction
12: $H = h'(\mu_{c_t,t-1}^{[k]}, x_t^{[k]})$ ▷ calculate Jacobian
13: $Q = H \Sigma_{c_t,t-1}^{[k]} H^T + Q_t$ ▷ measurement covariance
14: $K = \Sigma_{c_t,t-1}^{[k]} H^T Q^{-1}$ ▷ calculate Kalman gain
15: $\mu_{c_t,t}^{[k]} = \mu_{c_t,t-1}^{[k]} + K(z_t - \hat{z})$ ▷ update mean
16: $\Sigma_{c_t,t}^{[k]} = (I - KH)\Sigma_{c_t,t-1}^{[k]}$ ▷ update covariance
17: $w^{[k]} = \left|2\pi Q\right|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2}(z_t - \hat{z})^T Q^{-1} (z_t - \hat{z}) \right\}$ ▷ importance wt.
18: **end if**
19: $\underset{j \neq c_t}{\forall} \langle \mu_{j,t}^{[k]}, \Sigma_{j,t}^{[k]} \rangle = \langle \mu_{j,t-1}^{[k]}, \Sigma_{j,t-1}^{[k]} \rangle$ ▷ unobserved landmarks left unchanged
20: **end for**

21: $\mathcal{Y}_t \leftarrow \varnothing$ ▷ initialize new particle set
22: **for** M times **do** ▷ resample M particles
23: draw random k from $\hat{\mathcal{Y}}_t$ with probability $\propto w^{[k]}$ ▷ resample
24: add $\left\langle x_t^{[k]}, N_t^{[k]}, \langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]} \rangle, \ldots, \langle \mu_{N_t^{[k]},t}^{[k]}, \Sigma_{N_t^{[k]},t}^{[k]} \rangle \right\rangle$ to $\mathcal{Y}_t$
25: **end for**
26: **return** $\mathcal{Y}_t$

The likelihood is calculated as follows

$$\begin{aligned} & p(z_t \mid c_t, \hat{c}^{t-1,[k]}, x^{t,[k]}, z^{t-1}, u^t) \\ & = \int p(z_t \mid \theta_{c_t}, c_t, \hat{c}^{t-1,[k]}, x^{t,[k]}, z^{t-1}, u^t) p(\theta_{c_t} \mid c_t, \hat{c}^{t-1,[k]}, x^{t,[k]}, z^{t-1}, u^t) d\theta_{c_t} \\ & = \int \underbrace{p(z_t \mid \theta_{c_t}, c_t, x_t^{[k]})}_{\sim \mathcal{N}(z_t;\, h(\theta_{c_t}, Q_t))} \underbrace{p(\theta_{c_t} \mid \hat{c}^{t-1,[k]}, x^{t-1,[k]}, z^{t-1})}_{\sim \mathcal{N}(\mu^{[k]}_{c_t,t-1}, \Sigma^{[k]}_{c_t,t-1})} d\theta_{c_t} \end{aligned} \tag{10.17}$$

The linearization of the measurement model function h enables us to obtain this in closed form

$$\begin{aligned} & p(z_t \mid c_t, \hat{c}^{t-1,[k]}, x^{t,[k]}, z^{t-1}, u^t) \\ & \quad = \left| 2\pi Q^{[k]}_{c_t,t} \right|^{-\frac{1}{2}} \exp\left\{ -\frac{1}{2} (z_t - \hat{z}_j)^T Q^{[k]-1}_{c_t,t} (z_t - \hat{z}_j) \right\} \end{aligned} \tag{10.18}$$

where $\hat{z}_j = h(\mu^{[k]}_{c_t,t-1}, x^{[k]}_t)$ is the estimated measurement and $Q^{[k]}_{c_t,t}$ was defined in (10.7) as a function of the data association c_t.

The steps of the FastSLAM algorithm with unknown data association are fairly the same with the new data association step discussed above:

1. **Extending the path posterior by sampling new poses.**
2. **Obtain data association.**
3. **Updating the observed landmark estimate.**
4. **Resampling.**

10.6 The FastSLAM Algorithms

The Algorithm 20 and 21 summurizes the FastSLAM 1.0 algorithm with unknown data association. Particles are of the form

$$\mathcal{Y}^{[k]}_t = \left\langle x^{[k]}_t, N^{[k]}_t, \langle \mu^{[k]}_{1,t}, \Sigma^{[k]}_{1,t}, i^{[k]}_{1,t} \rangle, \ldots, \langle \mu^{[k]}_{N^{[k]}_t,t}, \Sigma^{[k]}_{N^{[k]}_t,t}, i^{[k]}_{N^{[k]}_t,t} \rangle \right\rangle \tag{10.19}$$

In addition to the pose $x^{[k]}_t$ and the landmark estimates $\mu^{[k]}_{j,t}$ and $\Sigma^{[k]}_{j,t}$ for $j = 1, \ldots, N^{[k]}_t$, each particle k contains the number of landmarks $N^{[k]}_t$ in its local map, and each landmark carries a probabilistic estimate of its existence $i^{[k]}_{j,t}$, which is actually a counter.

Iterating the filter requires time linear $\mathcal{O}(N)$ in the maximum number of landmarks $\max_{k=1,\ldots,M} N^{[k]}_t$ in each map, and it is also linear $\mathcal{O}(M)$ in the number of particles M. In sum, FastSLAM performs with $\mathcal{O}(MN)$ if implemented *naively*, but it is possible to achieve $\mathcal{O}(M \log N)$ is implemented efficiently. For the reader interested, [93] is concerned with several aspects that must be considered in order to implement FastSLAM efficiently.

Algorithm 20 FastSLAM1.0 (Part I)

Require: Observation or measurement z_t, control u_t and the set of M particles $\mathcal{Y}_{t-1}$ obtained in the previous temporal step.
Motion model to sample new pose $x_t \sim p(x_t \mid x_{t-1}, u_t)$.
A measurement prediction function $h(\mu_{t-1}, x_t)$, its Jacobian $h'(\mu_{t-1}, x_t)$, inverse function $h^{-1}(z_t, x_t)$ and the model noise Q_t.
The importance factor p_0 used for new landmarks.
A resampling method to draw M particles with probability $\propto_{k=1,\ldots,M} w^{[k]}$.

Ensure: Set of M particles $\mathcal{Y}_t$ obtained applying FastSLAM 1.0 algorithm.

Algorithm: FastSLAM1.0(z_t, u_t, $\mathcal{Y}_{t-1}$) **return** $\mathcal{Y}_t$

1: **for** $k = 1$ **to** M **do** ▷ loop over all particles
2: retrieve $\left\langle x_{t-1}^{[k]}, N_{t-1}^{[k]}, \left\langle \mu_{1,t-1}^{[k]}, \Sigma_{1,t-1}^{[k]}, i_{1,t-1}^{[k]} \right\rangle, \ldots, \left\langle \mu_{N_{t-1}^{[k]},t-1}^{[k]}, \Sigma_{N_{t-1}^{[k]},t-1}^{[k]}, i_{N_{t-1}^{[k]},t-1}^{[k]} \right\rangle \right\rangle$ from $\mathcal{Y}_{t-1}$
3: $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$ ▷ sample new pose
4: **for** $j = 1$ **to** $N_{t-1}^{[k]}$ **do** ▷ measurement likelihoods
5: $\hat{z}_j = h(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ ▷ measurement prediction
6: $H_j = h'(\mu_{j,t-1}^{[k]}, x_t^{[k]})$ ▷ calculate Jacobian
7: $Q_j = H_j \Sigma_{j,t-1}^{[k]} H_j^T + Q_t$ ▷ measurement covariance
8: $w_j = \left|2\pi Q_j\right|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - \hat{z}_j)^T Q_j^{-1}(z_t - \hat{z}_j)\right\}$ ▷ likelihood correspondence
9: **end for**
10: $w_{N_{t-1}^{[k]}+1} = p_0$ ▷ importance factor, new landmark
11: $w^{[k]} = \max_{j=1,\ldots,N_{t-1}+1} w_j$ ▷ max likelihood correspondence
12: $\hat{c} = \arg\max_{j=1,\ldots,N_{t-1}+1} w_j$ ▷ index of ML landmark
13: $N_t^{[k]} = \max\left\{N_{t-1}^{[k]}, \hat{c}\right\}$ ▷ new number of landmarks in map
14: **for** $j = 1$ **to** $N_t^{[k]}$ **do** ▷ update Kalman filters
15: **if** $j = \hat{c} = N_{t-1}^{[k]} + 1$ **then** ▷ is new landmark?
16: $\mu_{j,t}^{[k]} = h^{-1}(z_t, x_t^{[k]})$ ▷ initialize mean
17: $H_j = h'(\mu_{j,t}^{[k]}, x_t^{[k]})$ ▷ calculate Jacobian
18: $\Sigma_{j,t}^{[k]} = (H_j^{-1})^T Q_t H_j^{-1}$ ▷ initialize covariance
19: $i_{j,t}^{[k]} = 1$ ▷ initialize counter
▷ see next page for continuation

The algorithm described here considers a single measurement at a time. This choice is made for notational convenience only. Multiple readings can be incorporated per time step by processing each observation sequentially. The weight for each particle is equal to the product of the weights due to each observation considered

Algorithm 21 FastSLAM1.0 (Part II)

▷ continued from the previous page

else if $j = \hat{c} \leq N_{t-1}^{[k]}$ **then** ▷ is observed landmark?
  $K = \Sigma_{j,t-1}^{[k]} H_j^T Q_{\hat{c}}^{-1}$ ▷ calculate Kalman gain
  $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]} + K(z_t - \hat{z}_{\hat{c}})$ ▷ update mean
  $\Sigma_{j,t}^{[k]} = (I - KH_j)\Sigma_{j,t-1}^{[k]}$ ▷ update covariance
  $i_{j,t}^{[k]} = i_{j,t-1}^{[k]} + 1$ ▷ increment counter
else ▷ all other landmarks
  $\mu_{j,t}^{[k]} = \mu_{j,t-1}^{[k]}$ ▷ copy old mean
  $\Sigma_{j,t}^{[k]} = \Sigma_{j,t-1}^{[k]}$ ▷ copy old covariance
  if $\mu_{j,t-1}^{[k]}$ outside perceptual range of $x_t^{[k]}$ **then** ▷ should landmark have been seen?
    $i_{j,t}^{[k]} = i_{j,t-1}^{[k]}$ ▷ no, do not change
  else if $i_{j,t-1}^{[k]} = 0$ **then** ▷ yes, and null counter?
    discard landmark j ▷ yes, discard dubious landmarks
  else
    $i_{j,t}^{[k]} = i_{j,t-1}^{[k]} - 1$ ▷ no, decrement counter
  end if
end if
end for
add $\left\langle x_t^{[k]}, N_t^{[k]}, \langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, i_{1,t}^{[k]} \rangle, \ldots, \langle \mu_{N_t^{[k]},t}^{[k]}, \Sigma_{N_t^{[k]},t}^{[k]}, i_{N_t^{[k]},t}^{[k]} \rangle \right\rangle$ to $\hat{\mathcal{Y}}_t$
end for

$\mathcal{Y}_t \leftarrow \varnothing$ ▷ construct new particle set
for M times **do** ▷ resample M particles
  draw random k from $\hat{\mathcal{Y}}_t$ with probability $\propto w^{[k]}$ ▷ resample
  add $\left\langle x_t^{[k]}, N_t^{[k]}, \langle \mu_{1,t}^{[k]}, \Sigma_{1,t}^{[k]}, i_{1,t}^{[k]} \rangle, \ldots, \langle \mu_{N_t^{[k]},t}^{[k]}, \Sigma_{N_t^{[k]},t}^{[k]}, i_{N_t^{[k]},t}^{[k]} \rangle \right\rangle$ to $\mathcal{Y}_t$
end for
return $\mathcal{Y}_t$

alone. Incorporating multiple observations per time step increase both the accuracy of data association and the accuracy of the resulting map. Therefore, the actual implementation takes multiple observation into account per time step. However, the reader might recall that in this case the data associations of each observation are clearly correlated and better data association methods must be applied, as discussed in Chapter 8.

Chapter 11

Results

11.1 Introduction

In this chapter we will test and evaluate the FastSLAM algorithm. Since it is a highly parametrizable method, we might analyze the influence of the parameters. However, we consider the results of the study in [38]. Here we will evaluate the algorithm in similar environments to demonstrate its correctness. Moreover, we have migrate it to real time scenarios using a mobile robot operating in *real* environments.

The SLAM problem can be studied in a myriad of environments and under different conditions. In the present work we concentrated exclusively in static, structured, indoor environments. We will call them *office-like* environments. They are easily described with a set of simple geometric features such as lines and corner points. In the following section we describe some environments of this kind, that have been used in the experiments. The metrics shown later allow us to evaluate the performance of our FastSLAM algorithm implementation quantitatively.

11.2 Experiments

We have used a common robotic differential drive platform called Pioneer P3-DX [2], developed by MobileRobots —formerly ActiveMedia. The motion models discussed in Chapter 5 are directly applicable to this kind of robots moving in a plane.

The range laser sensor employed is the SICK LMS-200 [128]. It is very precise, with a low sistematic error of $\pm 15\mathrm{mm}$ and statistical error of $5\mathrm{mm}$ [70]. The feature-based measurement model described in Section 6.3 will be applied to the points and lines extracted, using the methods discussed in Chapter 7, from the range laser scans provided by this device. It will be possible to employ other measurement models discussed in Chapter 6, it is outside the scope of this work though.

The evaluation of the SLAM methods has take place in different types of environments, which are described below.

Synthetic Maps design with CAD tools. The path and measurements are simulated. Synthetic maps let us control the whole configuration of the environment. The simulation tool is responsible of simulate the workings of the range laser and the odometry. We have used Player/Stage with this purpose [58].

Controlled Real Both the path followed by the robot and the *real* environment are developed by us. We can demonstrate that the SLAM algorithm works correctly with real environment and devices, i.e. robot and laser sensor.

Benchmark Real There exist data logs of third-party experiments. In this case we have no control at all over the robot path and the environment. Most known repositories are Radish and Rawseeds, but there are many others.

Additionally, the experiments might be real or simulated. Depending on the kind of environment, we might be constraint to a single option. That is the case of synthetic maps, for which the experiments has to be simulated. We can also run the experiments offline, which is the common case if using data logs, or online. In the present work we consider both offline and online experiments, to demonstrate that the FastSLAM algorithm can perform in real time with equivalent results.

In the following sections we describe the environments used, specifying their experimental purpose and the source. In the case of synthetic environments, we show the map and the parameters of its features. Contrary, for real maps we show the occupancy grid maps provided by the authors of the experiments or the one corresponding for the best particle obtained with our implementation of FastSLAM.

11.2.1 Basic map

Description Rectangular room with no elements inside, as shown in Figure 11.1. It is simplest map possible. It contains large lines forming 90° angles among them, which are easy to detect. Lines and corner parameters are given in Table 11.1.

Purpose Reference map to compared the results obtain with more complex maps. Although it is the simplest synthetic map considered, it does not mean it is more favorable for mapping, since the scarcity of landmarks might lead to incorrect results.

Source Synthetic map developed with CAD tools. Robot path and measurements are simulated.

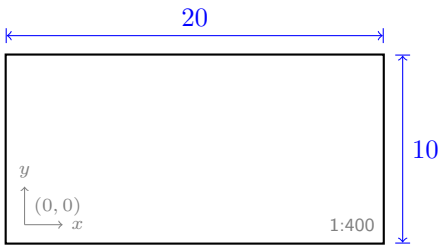


Figure 11.1: Basic Map

11.2.2 Midline map

Description Rectangular room with a single wall inside, as shown in Figure 11.2. It contains large lines forming 90° angles among them, which are easy to detect. Lines and corner parameters are given in Table 11.2.

Lines		Corners	
ρ (m)	θ (°)	x (m)	y (m)
1	−90	−1	−1
19	0	19	−1
9	90	19	9
1	180	−1	9

Table 11.1: Basic Map features

Purpose Analyze the behavior with large, simple loops. This maps simply adds one line to the *basic* map. However, it creates a particular structure common in indoor environments, known as *loop*.

Source Synthetic map developed with CAD tools. Robot path and measurements are simulated.

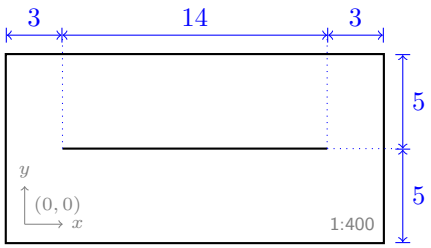


Figure 11.2: Midline Map

Lines		Corners	
ρ (m)	θ (°)	x (m)	y (m)
1	−90	−1	−1
19	0	19	−1
9	90	19	9
1	180	−1	9
4	90		

Table 11.2: Midline Map features

11.2.3 Bigloop map

Description Rectangular room with a big loop block inside. As shown in Figure 11.3 both the room and the loop block are rectangular, made of large lines forming $90°$ angles among them, which are easy to detect. Lines and corner parameters are given in Table 11.3.

Purpose Analyze the behavior with large loops, with a higher level of complexity than the *midline* map.

Source Synthetic map developed with CAD tools. Robot path and measurements are simulated.

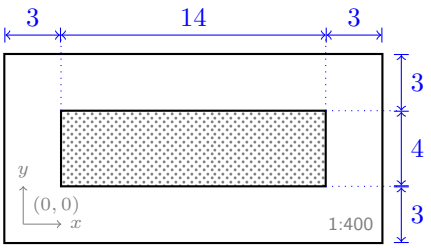


Figure 11.3: Bigloop Map

Lines		Corners	
ρ (m)	θ (°)	x (m)	y (m)
1	−90	−1	−1
19	0	19	−1
9	90	19	9
1	180	−1	9
2	90	2	2
16	0	16	2
6	90	16	6
2	0	2	6

Table 11.3: Bigloop Map features

11.2.4 Bigloop2 map

Description Rectangular room with two medium loop blocks inside. As shown in Figure 11.4 both the room and the loop blocks are rectangular, made of large lines forming $90°$ angles among them, which are easy to detect. Lines and corner parameters are given in Table 11.4. Note that some of the lines of the loop blocks are shared, since we detect infinite lines, not segments.

Purpose Analyze the behavior with multiple loops slightly smaller than those of the *midline* and *bigloop* maps.

Source Synthetic map developed with CAD tools. Robot path and measurements are simulated.

11.2.5 Complex map

Description Non-rectangular room with two big, rectangular loop blocks inside, as shown in Figure 11.5. Both the room and the loop blocks are made of large lines forming $90°$ angles among them, which are easy to detect. Lines and corner parameters are given in Table 11.5. Note that some of the lines of

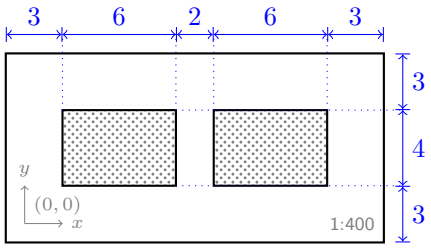


Figure 11.4: Bigloop2 Map

Lines		Corners	
ρ (m)	θ (°)	x (m)	y (m)
1	−90	−1	−1
9	90	−1	9
1	180	19	−1
19	0	19	9
2	90	2	2
8	0	8	2
6	90	8	6
2	0	2	6
16	0	10	2
10	0	16	2
		16	6
		10	6

Table 11.4: Bigloop2 Map features

the room and the loop blocks are shared, since we detect infinite lines, not segments.

Purpose Analyze the behavior with environments with moderate complexity, with several loops of different size and orientation.

Source Synthetic map developed with CAD tools. Robot path and measurements are simulated.

11.3 Metrics

To assess the accuracy of the resulting maps we consider both the feature-based map generated by the best particle of FastSLAM and the occupancy grid map constructed with the whole set of laser scans and the robot path estimated with FastSLAM. If we consider the feature-based map, we can compute the difference between the resulting map and the *real* or *reference* map. We denote this difference the map error ε_{map}. Unfortunately, this metric is only applicable to synthetic maps, since we now their real features. On the other hand, we compare the resulting occupancy grid map and the *real* map visually. The main advantage of this approach is that it can be applied when the *real* map is not available. Furthermore, it is useful to

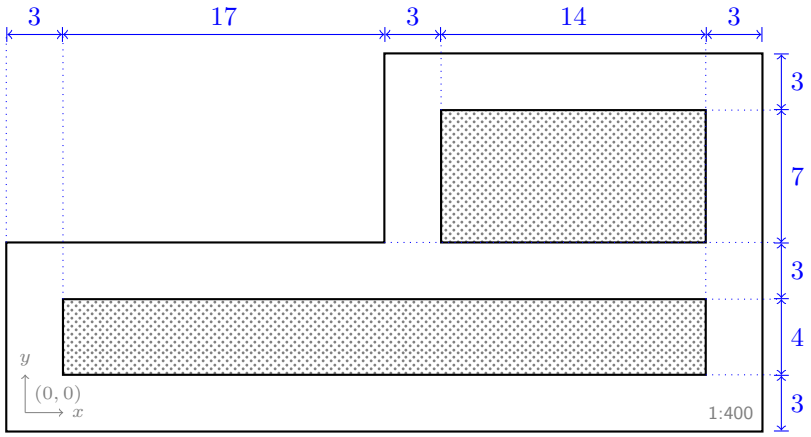


Figure 11.5: Complex Map

Lines		Corners	
ρ (m)	θ (°)	x (m)	y (m)
1	−90	−1	−1
39	0	39	−1
19	90	39	19
19	0	19	19
9	90	19	9
1	180	−1	9
2	90	2	2
36	0	36	2
6	90	36	6
2	0	2	6
16	90	22	9
22	0	36	9
		36	16
		22	16

Table 11.5: Complex Map features

analyze the performance of the SLAM algorithm when it close loops, among other aspects.

The error of an estimated map of N landmarks is given by

$$\varepsilon_{\text{map}} = \frac{1}{N} \sum_{i=1}^{N} \varepsilon_{\text{landmark}_i} \tag{11.1}$$

where $\varepsilon_{\text{landmark}_i}$ is the error associated with the i-th estimated landmark, with respecto to the *real* landmark most likely to have generated it. This rule also applies in the case that the resulting map has more landmarks than the *real* map.

If we consider point landmarks, the error is

$$\varepsilon_{\text{point}} = \sqrt{(x - m_{j,x})^2 + (y - m_{j,y})^2} \tag{11.2}$$

that compute the euclidean distance with real landmark j.

In the case of line landmarks, they are converted into points using

$$x = \rho \cos\theta \tag{11.3}$$

$$y = \rho \sin\theta \tag{11.4}$$

thus, we can also apply (11.2) to compute line errors after the conversion above.

Being FastSLAM a probabilistic algorithm, the evaluation must be performed applying statistical analysis. Therefor, we consider the error for M runs is the mean of the errors of the map obtained for each simulation

$$\varepsilon = \frac{1}{M} \sum_{i=1}^{N} \varepsilon_{\mathsf{map}_i} \tag{11.5}$$

This metric is thought to highlight the performance of the FastSLAM algorithm implemented. Another interesting metric is the number of detected landmarks, which can be compared with the true number in the *real* map. For M runs we take the mean of detected landmarks of each single run. Combining these metrics and by analyzing the occupancy grid map and the detected landmarks, we have complete information regarding the performance of the SLAM algorithm under different scenarios and configurations.

11.3.1 Error Assessment

When it comes to analyze the error of the resulting map, one might wonder whether a particular error in aceptable or not. The answer is not trivial and it depends on the final application of the resulting map, and the conditions under which it was obtained. Note that the error is not only attributable to the accuracy of the detected landmarks, but also to contributions of the following issues:

1. Discretization of the space due to the simulation tool. The map is passed to the simulator as an image that the simulator represents internally with a resolution of $2\mathrm{cm/pixel}$. The error induced by such discretization is not quantifiable because we ignore the internal details of the simulator.

2. The location of landmarks in the maps does not take into account the width of the walls, which is depicted in Figure 11.6. The synthetic maps described thus far have been designed with the thinner width possible, around 2 or 3 pixels. Depending on the scale of the map, it might induce $\pm 13\mathrm{cm}$ of error.

   Let a map of $1024 \times 768\mathrm{p}$ represent one environment of $40 \times 20\mathrm{cm}$ and another one of $20 \times 10\mathrm{cm}$. The resolution per pixel is $3.2\mathrm{cm/p}$ and $1.8\mathrm{cm/p}$, respectively.

   If the walls has $3\mathrm{p}$ width, the maximum error induced equals the length of the hypothenuse in the corners, as shown in Figure 11.6. Therefore, the error is $\sqrt{(3 \cdot 3.2)^2 + (3 \cdot 3.2)^2} \approx 13.58\mathrm{cm}$ for an environment of $40 \times 20\mathrm{cm}$ and $\sqrt{(3 \cdot 1.8)^2 + (3 \cdot 1.8)^2} \approx 7.64\mathrm{cm}$ for an environment of $20 \times 10\mathrm{cm}$.

3. With FastSLAM, the resulting map is not the best map, but the most likely one. The best particle at time t is the one that better *explains* the last measurement z_t, but not the one that minimize the error. Recall that the true map is represented by the whole set of particles, which represent the probabilistic distribution of all possible maps and robot poses.

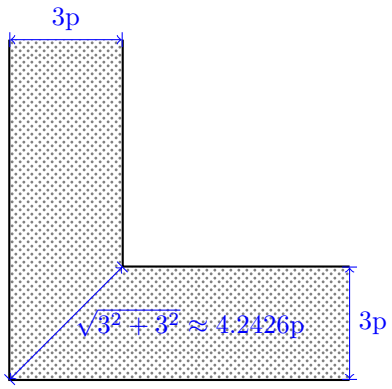


Figure 11.6: Detail of the wall width, in pixels

Within the scope of Robotics, the resulting map is usually considered appropiate if it is suitable for navigation. In general, the experiments discussed thus far are carried out in large environments where the robot moves at a maximum speed of $0.5\mathrm{m/s} = 1.8\mathrm{km/h}$, with trajectories of $\approx 200\mathrm{m}$. Therefore, errors in the location of landmarks around $10\mathrm{cm}$ are fairly acceptable.

11.4 Parametrization

The parameters that affect the FastSLAM algorithm implemented can be divided in three major groups. First, we have the parameters of the feature extraction methods. In particular, since we use an Split & Merge algorithm known as Iterative End-Point Fit to extract lines, we have:

1. The minimum number of points $p_{\min}$ of a valid line.
2. The minimum length $l_{\min}$ of a valid line.
3. The minimum distance $d_{\min}$ from a point to a line to split it.
4. The upper thresholds of collinearity $\rho_{\max}$ and $\theta_{\max}$.

and, for the corner extraction algorithm,

1. The maximum distance $d_{\max}$ from the interception point of a pair of lines to the actual segments.

Secondly, the motion and measurement models are parametrized by means of the:

1. Covariance matrix of the noise of the measurement model $Q_t = \begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}$.
2. Covariance matrix of the noise of the motion model $R_t = \begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix}$.
   The α_i parameters of the odometry and velocity motion models discussed in Chapter 5 are related with R_t in some measure.

Finally, we have the parameters of the particle filer:

1. The number of particles M.

2. The threshold of the number of effective particles N_{eff}. If the number of effective particles is lower than N_{eff} the resampling step is applied, otherwise it is omitted.

3. The probability p_0 of observing a new landmark.

Parameter	Description	Type	Constraint
m	num particles	$\mathbb{N}$	$m > 0$
p_0	new feature probability	$\mathbb{R}$	$p_0 \geq 0$
N_{eff}	num effective particles	$\mathbb{N}$	$0 \leq N_{\text{eff}} \leq m + 1$
$d_{\min}$	min point-line distance split	$\mathbb{R}$	$d_{\min} \geq 0$
$l_{\min}$	min line length	$\mathbb{R}$	$l_{\min} \geq 0$
$p_{\min}$	min num points in line	$\mathbb{N}$	$p_{\min} \geq 2$
$\rho_{\max}$	ρ colineality threshold	$\mathbb{R}$	$\rho_{\max} \geq 0$
$\theta_{\max}$	θ colineality threshold	$\mathbb{R}$	$0 \leq \theta_{\max} \leq \pi$
$d_{\max}$	max corner-lines distance	$\mathbb{R}$	$d_{\max} \geq 0$
$\begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\theta^2 \end{pmatrix}$	meas model noise covariance	$\mathbb{R}$	$\sigma_\cdot^2 \geq 0$
$\begin{pmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{pmatrix}$	motion model noise covariance	$\mathbb{R}$	$\sigma_\cdot^2 \geq 0$
f_e	feature extractor	$\mathcal{F}_e$	
l_e	line extractor	$\mathcal{L}_e$	
l_f	line fitting	$\mathcal{L}_f$	
m_m	motion model	$\mathcal{M}_m$	
r	resample method	$\mathcal{R}$	

Table 11.6: FastSLAM parameters

Set	Elements
$\mathcal{F}_e$	{ LINE, POINT, LINEPOINT }
$\mathcal{L}_e$	{ LT, SEF, IEPF, SM }
$\mathcal{L}_f$	{ LS, TLS }
$\mathcal{M}_m$	{ VELOCITY, ODOMETRY }
$\mathcal{R}$	{ SECUENTIAL, STRATIFIED }

Table 11.7: Sets of FastSLAM algorithms

All the parameters are summarized in Table 11.6, with their domain and the range of values they might take in general. These are not the optimal or recommended values which are shown in the sequel. The table also shows some alternative algorithms that might take charge of particular steps of the SLAM algorithm. In Table 11.7 are shown the sets of possible algorithms for those steps. Such sets are limited to the algorithms available in the current FastSLAM implementation, but there exist many other methods as it has been discussed throughout this document.

The recommended parametrization of Table 11.8 for the FastSLAM algorithm is taken from [38]. This configuration must be thought as an initial parametrization

before further tunning. The reader might consult [38], where the FastSLAM algorithm is evaluted for different parameter configurations to obtain such an appropiate parametrization.

Parameter	Value	Description
Iterative End-Point Fit		
$d_{\min}$	$[2,3]\mathrm{cm}$	Max dist. point-line
$l_{\min}$	$[0.5,2]\mathrm{m}$	Min line length
$p_{\min}$	15 points	Min # points
$\rho_{\max}$	$0.1\mathrm{m}$	ρ collinearity th.
$\theta_{\max}$	$0.07\mathrm{rad}$	θ collinearity th.
Corner extractor		
$d_{\max}$	$0.1\mathrm{m}$	
Motion model		
R_t	$\begin{pmatrix} [0.003,0.008] & 0 & 0 \\ 0 & [0.003,0.008] & 0 \\ 0 & 0 & 0.008 \end{pmatrix}$	Error covariance
Measurement model		
Q_t	$\begin{pmatrix} [0.01,0.08] & 0 \\ 0 & [0.0008,0.003] \end{pmatrix}$	Error covariance
Particle filter		
M	$[100,200]$ particles	# particles
p_0	0.005	New landmark prob.
N_{eff}	∞ (resample always)	Effective particles th.

Table 11.8: Recommended Parametrization

Additionally, one might select between the different methods that are part of FastSLAM. Below we discussed some recommendation regarding each:

Features Better results are obtained with both line and corner point features. Nevertheless, lines suffices to obtain accurate maps.

Line extraction Split & Merge is the line extraction method with better performance and lower computational cost. Its variant Iterative End-Point Fit is generally recommended, as discussed in Section 7.2. The parametrization given in Table 11.8 is advisable.

Line fitting As discussed in Section 7.3, it is important to use the Hessian model and the Total Least Squares (TLS) method, or one more robust, to manage *vertical* lines correctly. For instance, the Weighted TLS and RANSAC are recommended.

Motion model The odometry motion model is preffered over the velocity motion model, although not significantly.

Resampling Both sequential and stratified resampling give similar results.

Normal sampling There exist a number of normal sampling methods, which are discussed in Appendix A. The Ziggurat method is meant to be the most efficient one. However, the Box-Muller or the Marsaglia methods usually suffice. In fact, in the present work we have used Marsaglia mostly.

RNG The Random Number Generator (RNG) recommended in the literature for Monte Carlo simulation is the Mersenne Twister. Other RNGs does not work properly and the resampling step of the particle filter could degenerate faster.

11.5 Evaluation in Synthetic Environments

We have run FastSLAM with simulated data for all the synthetic maps shown in Section 11.2. The parametrization discussed thus far has been taken, but we have only use $M = 10$ particles. The robot path have been control by hand at relatively high speeds, at least for the linear velocity v, which is less error prone than the angular velocity ω. All simulations have been done in real time. The results are shown in Figure 11.7, 11.8, 11.9, 11.10 and 11.11, for the *basic*, *midline*, *bigloop*, *bigloop2* and *complex*, respectively.

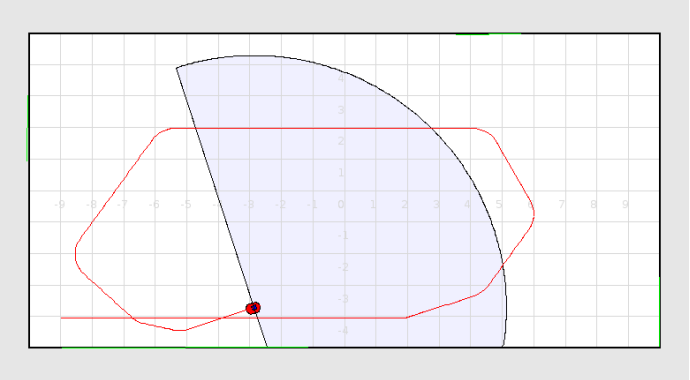


Figure 11.7: Resulting feature-based map after running FastSLAM with the *basic* map in the simulator

The resulting feature-based map and the estimated robot path are shown in the simulator. Therefore, we superposed the resulting map over the real synthetic map for comparison. In each step, the robot path and map of the best particle are shown. The robot path is shown with a red line (—), while the landmarks in the map are shown with green lines (—) —we are not using the corner point features. The simulated robot, the range laser scan and the real map are shown by the simulator itself.

We observe that the *basic* and *midline* maps are very accurate. More complex maps, like the *bigloop* and *bigloop2* maps are slightly less accurate. In general, all the resulting maps are clearly suitable for navigation. The algorithm is capable to map large maps of $20 \times 10\text{m}$, even with multiple loops.

The *complex* map is even larger with $40 \times 20\text{m}$, and it has two loops of different dimensions and orientation. The resulting map is less accurate but it still suffices for navigation. We might increase the number of particles M to obtain better results. In fact, this is necessary when maps are larger, specially if they have large loops. This is the case of the *complex* map experiment, where the path is $\approx 160\text{m}$ length.

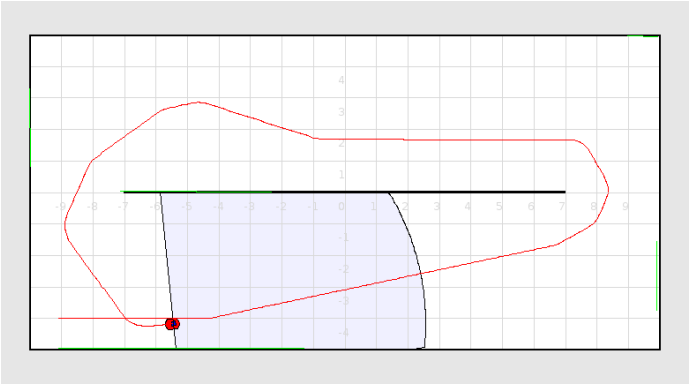


Figure 11.8: Resulting feature-based map after running FastSLAM with the *midline* map in the simulator

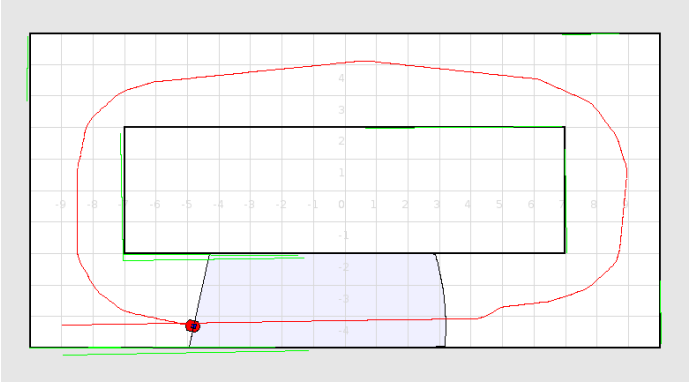


Figure 11.9: Resulting feature-based map after running FastSLAM with the *bigloop* map in the simulator

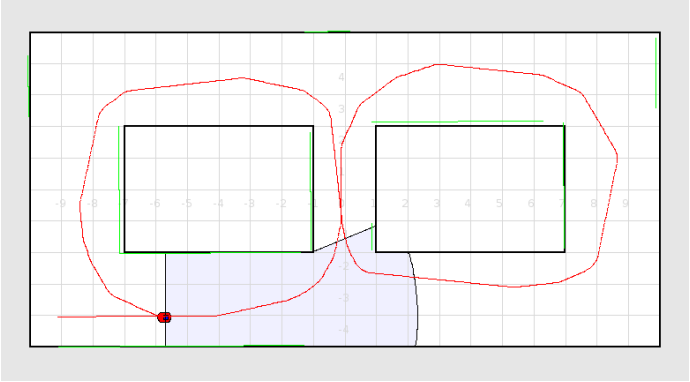


Figure 11.10: Resulting feature-based map after running FastSLAM with the *bigloop2* map in the simulator

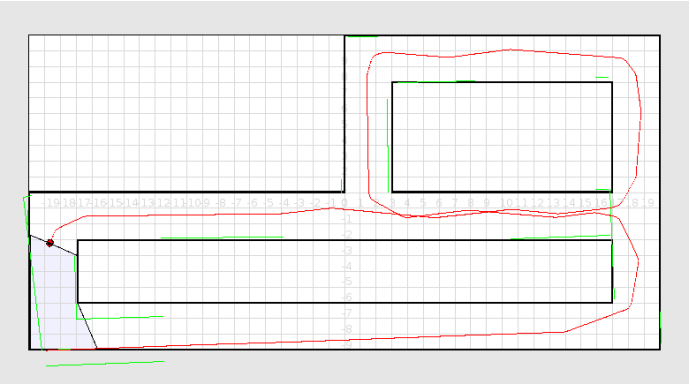


Figure 11.11: Resulting feature-based map after running FastSLAM with the *complex* map in the simulator

11.6 Evaluation in Real Environments

Now we turn into experiments in real environments using a mobile robot and a range laser. We use the same configuration for the FastSLAM algorithm, with $M = 10$ particles. The environment we are mapping is similar to the *basic* map. In Figure 11.12 we see the path follow by the robot represented using pure odometry. The acutal path consisted in two nearly perfect loops with square shape. The error induced by the odometry produce an important divergence of the orientation component of the robot pose.

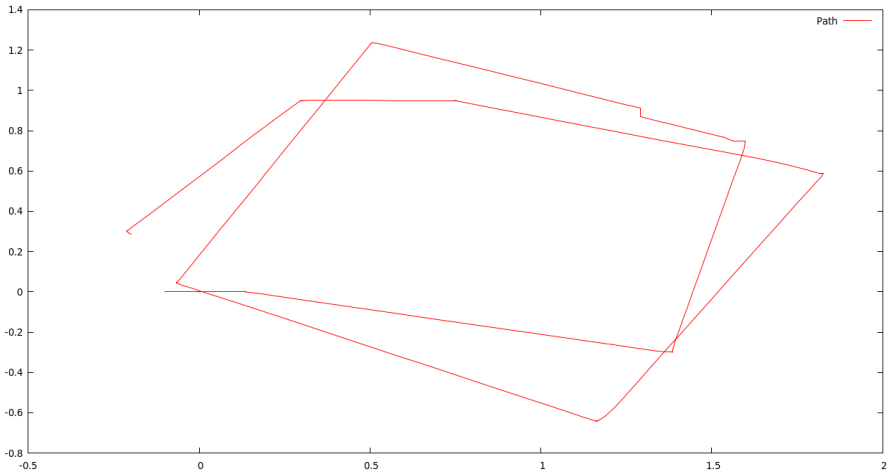


Figure 11.12: Path with pure odometry of real experiment

If we apply FastSLAM we can obtain both the map and the actual robot path estimates. In Figure 11.13 we have the resulting map and the robot path estimates obtained by the best particle with FastSLAM. The image shows the feature-based map, which containes the line features detected. Note that with real environ-

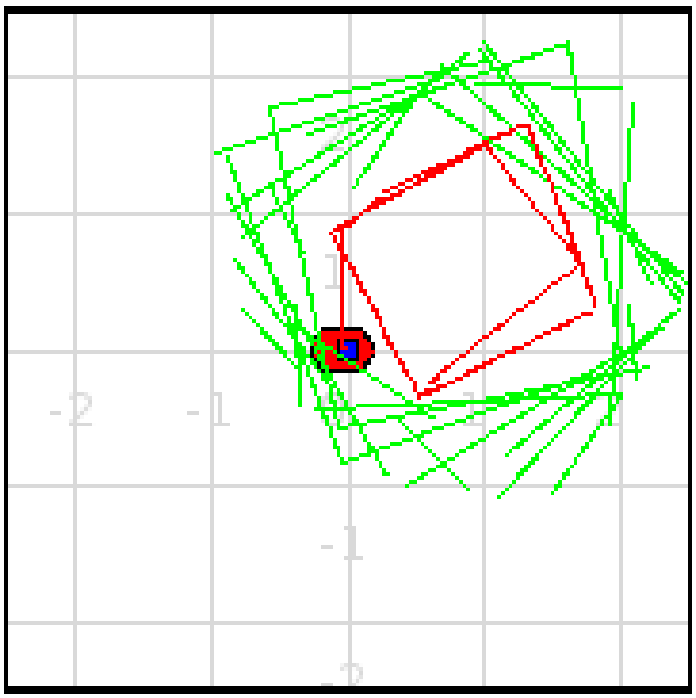


Figure 11.13: Map and robot path estimates of the best particle obtained with FastSLAM

ments and sensors, the noise is relatively higher. Thus, the same configuration that works in simulated environments performs worse. In fact, the resulting map might contain more lines than the *real* map has. For this reason, in Figure 11.13 the number of landmarks detected is excessive. Considering a higher motion and measurement noise will produce better results. Furthermore, with more particles we could maintain more hypotheses at some computational cost. However, if the motion uncertainty increase, we must increase the number of particles M similarly, to represent adequately the space of possible robot poses.

Chapter 12

Conclusions

12.1 Introduction

This thesis evaluates FastSLAM, a scalable approach to the Simultaneous Localization And Mapping (SLAM) problem with unknown data association proposed by Montemerlo in [93], in a real mobile robot. FastSLAM exploits sparsity in the dependencies between data and the state variable *over time* to factor the SLAM problem into a set of low-dimensional problems. It samples over the robot's path and data associations, and computes independent landmark estimates conditioned on each particle. It is an efficient solution suitable for real time application and deployment in mobile robots.

Recall that mapping is a field of active research and it is considered by many to be a key prerequisite to truly autonomous robots [145]. There exist a number of SLAM algorithms, but they are constraint to static, structured and low-dimensional environments to some extent. This is also the case of FastSLAM, for which reason the experiments have been performed in such type of scenario.

In the following sections we take some of the results obtained in [38], since they are generally equivalent to the ones obtain here. The present work confirms the applicability of FastSLAM in mobile robots operating in real time, with similar efficiency and accuracy. In particular, we have used the robot model Pioneer P3-DX in *office-like* environments, and the resulting maps are sufficiently accurate for navigation.

12.2 FastSLAM Algorithm

We have implemented the FastSLAM 1.0 algorithm discussed in Chapter 10. It has been evaluated under different conditions and environments, with both real and simulated data. The integration in a mobile robot, being one of the major goals of this thesis, have been fully achieved with promising results.

The FastSLAM algorithm must be thoroughly parametrized to work properly. In this thesis we embrace the configuration proposed in [38] as an initial setting. Thus, the algorithm gives accurate maps even for cluttered environments and noisy odometry information. In all cases, the resulting map generated by the best particle suits for navigation.

12.3 Feature-based Maps. Landmarks

The original FastSLAM algorithm generates feature-based maps, that is, maps composed of a set of landmarks. The election of the type of landmarks is crucial, because it constraints the application of the algorithm to environments that can be represented as a combination of such landmarks. Feature-based maps are low-dimensional, which is an important advantage over other approaches. For this reason, they are preferable for real time applications.

We considered lines and corner points, which suffices to map *office-like* environments. Since corner points are computed over the detected lines, the later are always necessary. However, it is possible to obtain accurate maps using line features only. Corners give additional information that reduce the convergence time of the method. Although not compulsory, corners are less ambiguous than line because it is not common to find close corners, that might produce wrong data associations. In this sense, corners are very information rich. Unfortunately, being points they are observed sporadically, while line are observed more frequently because of their length.

12.4 Feature Extraction

The robot observed the environment using a range laser sensor. There exist a number of feature extraction methods for range lasers, as discussed in Chapter 7. Since we consider line and corner points as the basic features present in the environment, the algorithms we have used are focused on the extraction of such features. Actually, we only extract lines, because corners are easily detected with an intersection test between all possible pairs of lines.

The line extraction method used is called Iterative End-Point Fit, which is a variant of the Split & Merge. It is an efficient, reliable, robust and simple algorithm that produce excellent results in *office-like* environments. However, it is highly parametrizable and the correct parametrization depends tightly of the environment. Here, we considered the configuration summarized in Section 11.4 and further discussed in [38].

12.5 Motion Model

The motion noise modelled has a high impact in the SLAM process. Indeed, the noise considered in the motion model must be at least equal to the higher real noise in the odometry. This is a mandatory condition to let the particle population cover the whole space of possible robot poses. Otherwise, the SLAM method will diverge. Adjusting the motion noise might be thought as an odometry calibration process.

However, one cannot use a very high value for the noise. If we increment the noise of the motion model, we must also increment the number of particles in order to cover the larger space of possible robot poses. This inevitably slows the algorithm, since the computational cost of FastSLAM depends linearly on the number M of particles. Also note that the noise in the orientation of the robot has a greater impact if compared with that of the location [93, 147].

We have considered the velocity and odoemtry motion models proposed in [149]. In general, the odometry motion models produces slightly better results. Both

models have several parameters α_i that establish the Gaussian noise covariance as a function of the linear and angular velocities v and ω. Therefore, it is possible to express some kind of conditional dependence between both velocity components. Such motion models let a fine-grain parametrization, that if adjusting adequately it produces good results.

12.6 Measurement Model

Range lasers are very precise sensors, so it is recommended to consult the data sheet to model their note accurately. Contrary to the motion model, measurement noise cannot be compensate adjusting other parameters in the FastSLAM algorithm. Thus, the noise must be adjusted thoroughly.

According with [38], once the range component ρ is fixed, the bearing component θ affects notably in the SLAM process because it is common to find nearby lines whose angles differ slightly. Depending on the noise considered, it might produce incorrect data associations leading to the method divergence. Contrary, given a fixed noise for θ, the influence of the noise for ρ is irrelevant because it is not common to find parallel lines close, and then the possibility of incorrect data association is low.

12.7 Data Association

The FastSLAM algorithm uses the Maximum Likelihood data association, which generally works well when the correct data association is significantly more probable than the incorrect associations. Since FastSLAM uses a particle filter to estimate the robot path, it maintains multiple hypothesis that track different data associations for each robot path estimate. This multi-hypothesis data association factors robot pose uncertainty out of the data association problem. The resampling step of the particle filter discard those particles that represent hypotheses with low likelihood.

However, it is still possible that a measurement corresponds to several landmarks in the map. In the present work we simply discard such measurements, but there exist a number of robust data association methods that consider this situation, as discussed in Section 8.4.

12.8 Particle Filter

The resampling step of the particle filter is considered very important, since it is responsible of select the particles most likely to represent the real robot path and map, and discard the rest. The resampling method applied, called *sequential resampling* or *low variance resampling*, has the property of maintain the variety in the particle set [149]. Alternatively, the *stratified resampling* method has lower sampling variance and tends to perform well when a robot tracks multiple, distinct hypotheses with a single particle filter. In practice, both methods produce similar results.

If the number of effective particles is greater than a threshold, the resampling step can be omitted. The effect of such threshold is actually negligible in practice, because the number of effective particles is kept high —close to the total number

of particles M. Thus, only high values of the threshold, close to M might make a difference, as shown in Section 4.2.

The higher the number of particles M, the better the accuracy of the resulting map. Nevertheless, it is possible to obtain very accurate maps with less than 200 particles [38]. Finally, the probability p_0 to consider a measurement as a new landmark has not effect if it is kept within a rational interval. Such interval goes from values greater than 1 for landmarks previously observed to values close to 0 for new landmarks. Probably, with a large number of particles M and a large number of landmarks N, it is possible that some landmarks produce data associations with low probabilities, so in such cases p_0 must be particularly small.

Chapter 13

Future Work

In the following sections we discuss a number of open research issues organized in topics.

13.1 Map Representation

Feature-based maps considered in this work are formed by lines and corner points. Such features are appropiate to describe *office-like* environment, but they are not suitable for other environment, such as lands or forests. Furthermore, lines are also a poor representation for structured environments because they have infinite length. A better feature will be the segment, that is, a line delimited by a starting and end point. This will also reduce the ambiguity of data association to some extent.

It will be even better to apply SLAM algorithms that does not rely on features. In the case of FastSLAM, the Grid-based FastSLAM [149] variant is quite promising. It is also interesting the combination of metric and topologic information in hybrid maps [13, 14, 49].

The information registered of the environment might also be extended, specially if multiple sensors are used. Landmarks can be annotated with properties such as color, texture and so on. This additional information will be extremely useful to reduce the ambiguity in the data association problem.

13.1.1 Dynamic Environments

In general, the environments where mobile robots operate are dynamic, e.g. doors that get close or open, people walking, other robots, structural elements that are moved, etc. It will be useful to develop a system with the ability to classify the landmarks between static or dynamic [93]. The integration with the mapping algorithm will allow to remove or modify landmarks in the map, considering dynamic objects. The data association problem could also take this information into account.

13.1.2 Exploration

In order to construct the map of an unknown environment, the robot must explore it. There exist a number of exploration algorithms [149], some of them with a probalistic background. These algorithms control the robot so as to maximize its knowledge

about the externel world. One approach is known as FastSLAM exploration. It integrates a FastSLAM algorithm in the exploration method. It uses the grid-based version of FastSLAM and it outputs an exploration path expressed in relative motion commands. The action sequence is selected such that it minimizes the expected entropy of the resulting map after executing the controls.

13.2 Robust Data Association

In the present work, under ambiguous data association the measurement is discarded. This naive approach suffices in most cases, but there exist a number of robust data association techniques that propose more appropiated solutions, as discussed in Section 8.4. In general, these methods consider multiple hypotheses when there are many plausible data associations. For instance, Nieto propose the creation of new particles with each data association hypothesis [103]. Only the correct hypotheses will survive after the resampling step, since they will be asigned a high weights according with the maximum likelihood rule.

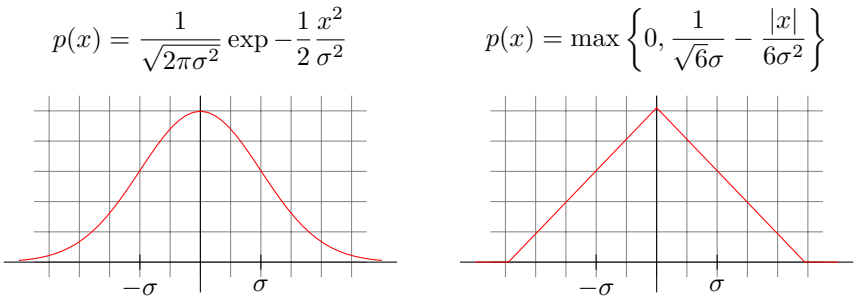
Similarly, the *Multiple Hypothesis Tracking* data association algorithm tracks several hypotheses, one for each plausible data association [99, 117]. Other robust data associations algorithms are the *Joint Compatibility Branch and Bound* [9, 100, 105] and *Combined Constraint Data Association* [7], which based on the comparison of joint data association hypotheses using joint compatibility, a measure of the probability of the set of observations ocurring together.

13.3 FastSLAM 2.0

Taking the implementation of FastSLAM 1.0, it is relatively straightforward to implement the improved version FastSLAM 2.0 [93]. It solve the problem of the particle filter degeneration and it produces equivalent results with fewer particles. FastSLAM 2.0 incorporates the current observation into the proposal distribution over robot paths, not just the importance weights, in order to better match the posterior.

Appendix A

Sampling



(a) Normal Distribution (b) Triangular Distribution

Figure A.1: Probability density functions with variance σ^2

If $X \sim \mathcal{N}(\mu, \sigma^2)$ and a and b are real numbers, then $aX+b \sim \mathcal{N}(a\mu+b, (a\sigma)^2)$. Therefore, it is possible to relate all normal random variables X to the standard normal. If $X \sim \mathcal{N}(\mu, \sigma^2)$, then

$$Z = \frac{X - \mu}{\sigma} \tag{A.1}$$

Conversely, if Z is a standard normal random variable, sampled from $Z \sim \mathcal{N}(0, 1)$, then

$$X = \mu + \sigma Z \tag{A.2}$$

is a normal random variable with mean μ and variance σ^2.

Algorithm 22 Central Limit Theorem

Require: An uniform random number generator **rand**(a, b) that generates a random number $r \in [a, b]$.

Ensure: Random sample from a zero-mean normal distribution with variance $\sigma^2 = 1$.

Algorithm: NormalCentralLimit() **return** u

1: **return** $u = \frac{1}{2} \sum_{i=1}^{12} \textbf{rand}(-1, 1)$

Algorithm 23 Box-Muller

Require: An uniform random number generator **rand**(a, b) that generates a random number $r \in [a, b]$.

Ensure: A pair of random samples $\mathcal{U} = \langle u_1, u_2 \rangle$ from a zero-mean normal distribution with variance $\sigma^2 = 1$, obtained from a pair of uniform random samples applying Box-Muller theorem.

Algorithm: BoxMuller() **return** $\mathcal{U}$

1: $r_1 = \textbf{rand}(0, 1)$
2: $r_2 = \textbf{rand}(0, 2\pi)$
3: $s = \sqrt{-2 \log (1 - r_1)}$
4: **return** $\mathcal{U} = \langle s \cos r_2, s \sin r_2 \rangle$

Algorithm 24 Marsaglia

Require: An uniform random number generator **rand**(a, b) that generates a random number $r \in [a, b]$.

Ensure: A pair of random samples $\mathcal{U} = \langle u_1, u_2 \rangle$ from a zero-mean normal distribution with variance $\sigma^2 = 1$, obtained from a pair of uniform random samples applying the polar form of Box-Muller theorem, also known as Marsaglia.

Algorithm: Marsaglia() **return** $\mathcal{U}$

1: **repeat**
2: $\quad r_1 = \textbf{rand}(-1, 1)$
3: $\quad r_2 = \textbf{rand}(-1, 1)$
4: $\quad r = r_1^2 + r_2^2$
5: **until** $0 < r \leq 1$
6: $s = \sqrt{-2 \frac{\log r}{r}}$
7: **return** $\mathcal{U} = \langle s r_1, s r_2 \rangle$

Algorithm 25 Triangular Central Limit Theorem

Require: An uniform random number generator **rand**(a, b) that generates a random number $r \in [a, b]$.

Ensure: Random sample from a zero-mean triangular distribution with variance $\sigma^2 = 1$.

Algorithm: TriangularCentralLimit() **return** u

1: **return** $u = \frac{\sqrt{6}}{2} \left(\textbf{rand}(-1, 1) + \textbf{rand}(-1, 1) \right)$

Algorithm 26 Triangular Geometric Expression

Require: An uniform random number generator **rand**(a, b) that generates a random number $r \in [a, b]$.

Ensure: Random sample from a zero-mean triangular distribution with variance $\sigma^2 = 1$.

Algorithm: TriangularGeometric() **return** u

$a = 0$ ▷ lower limit
$b = 0.5$ ▷ mode
$c = 1$ ▷ upper limit
▷ internal precomputed values
$d_1 = b - a$
$d_2 = c - a$
$d_3 = c - b$
$p_{12} = d_1 d_2$
$p_{13} = d_1 d_3$
$p_{23} = d_2 d_3$
$q_1 = \dfrac{d_1}{d_2}$
▷ compute triangular random variable
$r =$ **rand**$(0, 1)$
if $r > q_1$ **then**
    return $u = c - \sqrt{p_{23} r - p_{13}}$
else
    return $u = a + \sqrt{p_{12} r}$
end if

Appendix B

Point Geometry

B.1 Two-dimensional Point representation. Coordinate Systems

B.1.1 Cartesian Coordinate System

A cartesian (or rectangular) coordinate system in two dimensions is commonly defined by two axes, at right angles to each other, forming a plane (an xy-plane).

A point p is defined by orthogonal coordinates x and y. It's usually represented as $p = (x, y)$.

Distances between points are defined by the length of the line vector between the coordinates of each point $p_0 = (x_0, y_0)$ and $p_1 = (x_1, y_1)$:

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \tag{B.1}$$

Polar to Cartesian coordinate system transformation:

$$x = \rho \cos \theta \tag{B.2}$$
$$y = \rho \sin \theta \tag{B.3}$$

B.1.2 Polar Coordinate System

A polar coordinate system is a two-dimensional coordinate system in which each point on a plane is determined by an angle and a distance.

Each point in the polar coordinate system can be described with the two polar coordinates, which are usually called r (the radial coordinate, sometimes represented as ρ) and θ (the angular coordinate, polar angle, or azimuth angle, sometimes represented as φ or t). The r coordinate represents the radial distance from the pole, and the θ coordinate represents the anticlockwise (counterclockwise) angle from the $0°$ ray (sometimes called the polar axis), known as the positive x-axis on the Cartesian coordinate plane.

Distances between two points $p_0 = (\rho_0, \theta_0)$ and $p_1 = (\rho_1, \theta_1)$ is defined by:

$$d = \sqrt{\rho_0^2 + \rho_1^2 - 2\rho_0\rho_1 \cos(\theta_0 - \theta_1)} \tag{B.4}$$

Distance between two points in polar coordinates. Distances between two points $p_0 = (\rho_0, \theta_0)$ and $p_1 = (\rho_1, \theta_1)$ in polar coordinates can be obtained applying the law of cosines:

$$d^2 = a^2 + b^2 - 2ab\cos C \tag{B.5}$$

where $C = \theta_1 - \theta_0$, $a = \rho_1$, $b = \rho_0$ and $c = d$ so

$$d = \sqrt{\rho_1^2 + \rho_0^2 - 2\rho_1\rho_0\cos(\theta_1 - \theta_0)} \tag{B.6}$$

which yields (B.4) after applying cosine reflecting property $\cos(-\theta) = \cos\theta$ —i.e. the difference between point angles can be shifted. □

Cartesian to Polar coordinate system transformation:

$$\rho = \sqrt{x^2 + y^2} \tag{B.7}$$

$$\theta = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \end{cases} \tag{B.8}$$

Appendix C

Line Geometry

C.1 Line Representation

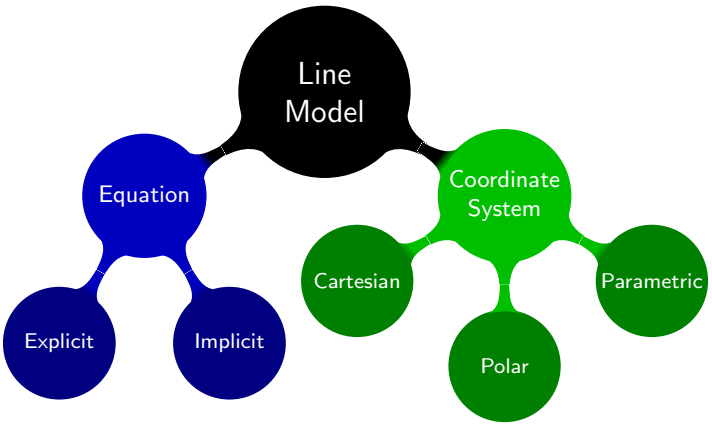


Figure C.1: Line Model Representation

The cartesian or rectangular representation in explicit form for y is denoted

$$y = b + mx \tag{C.1}$$

while the implicit form is

$$ax + by + c = 0 \tag{C.2}$$

Note that the implicit representation can be normalize into $a_n^2 + b_n^2 = 1$ dividing all parameters by $n = \sqrt{a^2 + b^2}$, that is

$$a_n x + b_n y + c_n = \tag{C.3}$$

$$\frac{a}{n}x + \frac{b}{n}y + \frac{c}{n} = 0 \tag{C.4}$$

Normalization of line Cartesian Implicit representation. Let a line l defined by the cartesian implicit representation equation $ax + by + c = 0$, if $a^2 + b^2 = 1$ it is meant

to be normalize. Dividing all line parameters by a factor n we can normalize a line l,

$$\frac{a}{n}x + \frac{b}{n}y + \frac{c}{n} = 0 \tag{C.5}$$

$$\left(\frac{a}{n}\right)^2 + \left(\frac{b}{n}\right)^2 = 1 \tag{C.6}$$

$$\frac{a^2}{n^2} + \frac{b^2}{n^2} = 1 \tag{C.7}$$

$$\frac{a^2 + b^2}{n^2} = 1 \tag{C.8}$$

$$a^2 + b^2 = n^2 \tag{C.9}$$

$$n = \sqrt{a^2 + b^2} \tag{C.10}$$

In order to keep a, b and c original sign we take the positive solution

$$n = \left|\sqrt{a^2 + b^2}\right| \tag{C.11}$$

□

The polar form is denoted

$$\rho = x\cos\theta + y\sin\theta \tag{C.12}$$

The parametric form for two points p_1 and p_2 in cartesian coordinates $p_i = (x_i, y_i)$ is denoted

$$x = x_1(1-t) + x_2 t \tag{C.13}$$

$$y = y_1(1-t) + y_2 t \tag{C.14}$$

which also represents a segment $\overline{p_1 p_2}$.

According with Figure C.2 (d) the rectangular to polar transformation is done with

$$\theta = \arctan\left(-\frac{1}{m}\right) \tag{C.15}$$

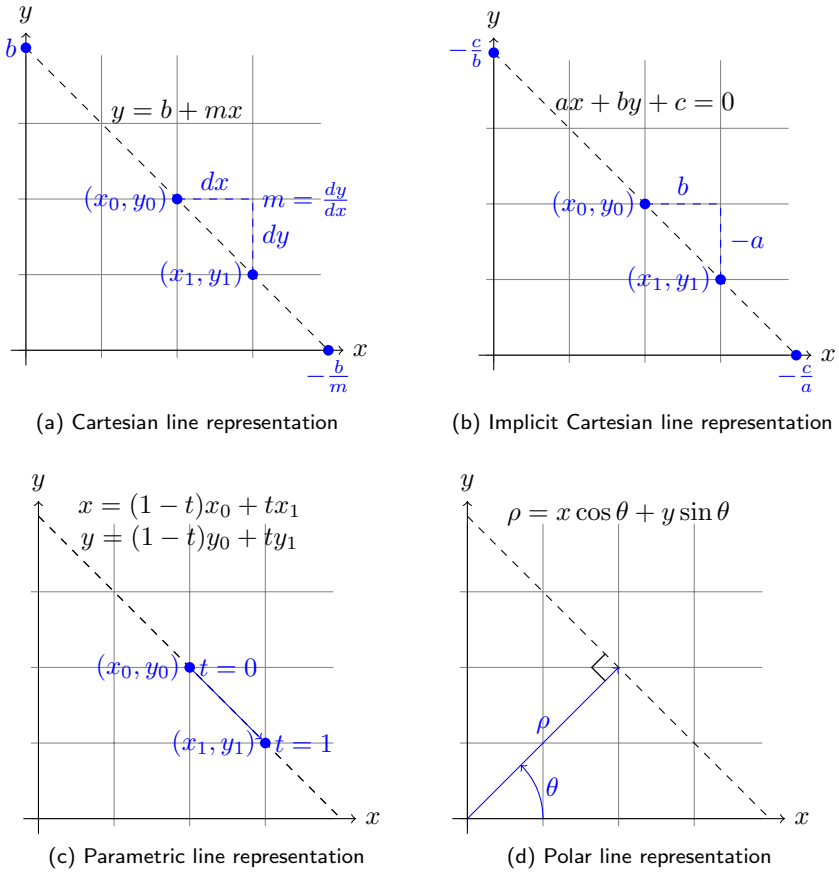
$$\rho = b\sin\theta \tag{C.16}$$

If $\arctan$ is computing with **atan2** we obtain the right angle, but we need two values dx and dy used to compute slope $m = \frac{dy}{dx}$. We only have m so we will usualy have to use **atan** and apply a transformation, which can be speed up using correction shown in Table C.1, depending on b value.

Fortunately, we can apply the following trick because we actually have two values b and m.

$$\theta = \arctan\left(\operatorname{sign}(b), \operatorname{sign}(-b)m\right) \tag{C.17}$$

Take into account that $-\operatorname{sign} a = \operatorname{sign}(-a)$ and the two parameters version of $\arctan$ is usually known as `atan2` in many programming languages, which gives an angle in the range $[-\pi, \pi]$.



(a) Cartesian line representation

(b) Implicit Cartesian line representation

(c) Parametric line representation

(d) Polar line representation

Figure C.2: Different line representations

Arctangent computation. Let a line defined in cartesian coordinates with the implicit equation $y = b + mx$, with y-intercept b and slope m, then a parallel line with y-intercept ± 1 can be constructed substracting $b - \operatorname{sign} b$.

$$\begin{aligned} y &= b + mx - (b - \operatorname{sign} b) && \text{(C.18)} \\ &= b + mx - b + \operatorname{sign} b && \text{(C.19)} \\ &= \operatorname{sign} b + mx && \text{(C.20)} \end{aligned}$$

The original line $y = b + mx$ has the following intersection points with x and y axis,

$$x\text{-intercept:} \quad y = 0 \rightarrow 0 = b + mx \Rightarrow x = -\frac{b}{m} \qquad \text{(C.21)}$$

$$y\text{-intercept:} \quad x = 0 \rightarrow y = b \qquad \text{(C.22)}$$

Analogously, the shifted line $y = \operatorname{sign} b + mx$ has the following,

$$x\text{-intercept:} \quad y = 0 \rightarrow 0 = \operatorname{sign} b + mx \Rightarrow x = -\frac{\operatorname{sign} b}{m} \qquad \text{(C.23)}$$

$$y\text{-intercept:} \quad x = 0 \rightarrow y = \operatorname{sign} b \qquad \text{(C.24)}$$

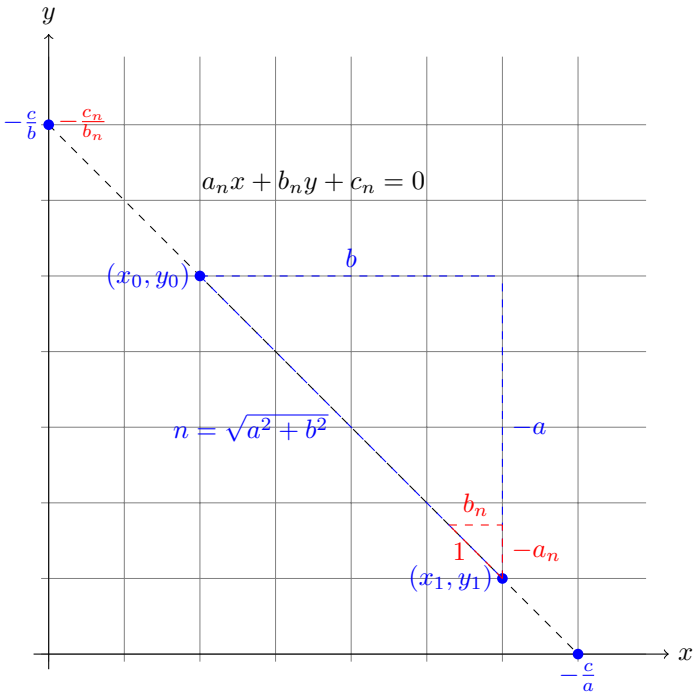


Figure C.3: Normalize Implicit Cartesian line representation

$\arctan$	b	Cuadrant	Correction
$+$	$+$	I	—
$+$	-	III	$+\pi$
-	$+$	II	$+\pi$
-	-	IV	$+2\pi$

Table C.1: Angle transformation

In both (C.21) and (C.23), the minus sign actually belongs to m. This will be important since $\arctan$ strongly depends on the signs.

The slope m can be computed with the incremental coordinate differences dx and dy between two points that belong to the line. Choosing the origin $(0, 0)$ and the intersection points $p_x = (x_c, 0)$ with x axis and $p_y = (0, y_c)$ with y axis, yields

$$dx = x_c \tag{C.25}$$
$$dy = y_c \tag{C.26}$$

For the orignal line $y = b + mx$ we have

$$dx = -\frac{b}{m} \tag{C.27}$$
$$dy = b \tag{C.28}$$

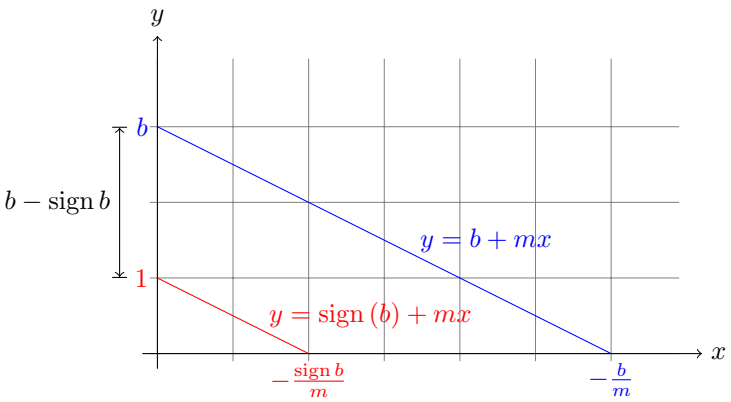


Figure C.4: Arctangent computation from line cartesian representation parameters b and m

and for the shifted line $y = \operatorname{sign} b + mx$

$$dx = -\frac{\operatorname{sign} b}{m} \tag{C.29}$$

$$dy = \operatorname{sign} b \tag{C.30}$$

The slope of a line is $m = \frac{dy}{dx}$, while the slope of the normal $\vec{N}$ is $m_{\vec{N}} = \frac{dx}{dy}$. Both lines can be used for this computation, but the shifted line gives a simple expression because in the original one we had to compute the quotient $-\frac{b}{m}$. Hence,

$$m_{\vec{N}} = \frac{dx}{dy} \tag{C.31}$$

$$= \frac{\frac{\operatorname{sign} b}{-m}}{\operatorname{sign} b} \tag{C.32}$$

$$= \frac{\operatorname{sign} b}{-\operatorname{sign}(b)m} \tag{C.33}$$

$$= \frac{\operatorname{sign} b}{\operatorname{sign}(-b)m} \tag{C.34}$$

$$= -\frac{1}{m} \tag{C.35}$$

Therefore, the argument of the normal vector to the line is the $\arctan$ of $m_{\vec{N}}$. To obtain an angle in the range $[-\pi, \pi]$ we use the not simplify quotient (C.34).

$$\theta = \arctan(\operatorname{sign}(b), \operatorname{sign}(-b)m) \tag{C.36}$$

□

With **atan2** the angle is in the range $[-\pi, \pi]$, so we must add 2π if the angle is negative to transform it into the range $[0, 2\pi]$.

C.2 Line-line intersection

The cartesian explicit representation of a pair of lines l_0 and l_1 is

$$l_0 : \quad y = b_0 + m_0 x \tag{C.37}$$
$$l_1 : \quad y = b_1 + m_1 x \tag{C.38}$$

Matching both equations yields

$$b_0 + m_0 x = b_1 + m_1 x \tag{C.39}$$

Solving for x and y,

$$x = \frac{b_1 - b_0}{m_1 - m_0} \tag{C.40}$$
$$y = b_0 + m_0 x \tag{C.41}$$

If $m_0 = m_1$ lines l_0 and l_1 are parallel and if $b_0 = b_1$ too, they are colinear —i.e. $l_0 = l_1$. That is, the intersection point p is

$$p = \begin{cases} \text{none} & \text{if } m_0 = m_1 \begin{cases} \text{colinear } (l_0 = l_1) & \text{if } b_0 = b_1 \\ \text{parallel } (l_0 \parallel l_1) & \text{others} \end{cases} \\ (x, y) & \text{others} \end{cases} \tag{C.42}$$

The intersection between two lines may also be computed using the implicit cartesian equation:

$$l_0 : \quad A_0 x + B_0 y + C_0 = 0 \tag{C.43}$$
$$l_1 : \quad A_1 x + B_1 y + C_1 = 0 \tag{C.44}$$

The corresponding transformation to explicit form yields

$$l_0 : \quad y = -\frac{C_0}{B_0} - \frac{A_0}{B_0} x \tag{C.45}$$
$$l_1 : \quad y = -\frac{C_1}{B_1} - \frac{A_1}{B_1} x \tag{C.46}$$

so we have the following relations with the explicit equation parameters

$$l_0 : \quad \begin{cases} b_0 & = -\frac{C_0}{B_0} \\ m_0 & = -\frac{A_0}{B_0} \end{cases} \tag{C.47}$$
$$l_1 : \quad \begin{cases} b_1 & = -\frac{C_1}{B_1} \\ m_1 & = -\frac{A_1}{B_1} \end{cases} \tag{C.48}$$

Therefore, by substitution in (C.40)

$$x = \frac{-\frac{C_1}{B_1} + -\frac{C_0}{B_0}}{-\frac{A_0}{B_0} + \frac{A_1}{B_1}} \tag{C.49}$$
$$= \frac{\frac{C_0 B_1 - C_1 B_0}{B_0 B_1}}{\frac{A_1 B_0 - A_0 B_1}{B_0 B_1}} \tag{C.50}$$
$$= \frac{C_0 B_1 - C_1 B_0}{A_1 B_0 - A_0 B_1} \tag{C.51}$$

As the polar equation is actually a normal implicit cartesian form, where $A = \cos\theta$, $B = \sin\theta$ and $C = -\rho$, by substitution we have

$$x = \frac{-\rho_0 \sin\theta_1 + \rho_1 \sin\theta_0}{\cos\theta_1 \sin\theta_0 - \cos\theta_0 \sin\theta_1} \tag{C.52}$$

$$= \frac{\rho_1 \sin\theta_0 - \rho_0 \sin\theta_1}{\sin\theta_0 \cos\theta_1 - \cos\theta_0 \sin\theta_1} \tag{C.53}$$

$$\tag{C.54}$$

Applying the trigonometric property $\sin(\alpha \pm \beta) = \sin\alpha\cos\beta - \cos\alpha\sin\beta$ yields

$$x = \frac{\rho_1 \sin\theta_0 - \rho_0 \sin\theta_1}{\sin(\theta_0 - \theta_1)} \tag{C.55}$$

However, as $\sin$ and $\cos$ of θ_0 and θ_1 may be precomputed for each line, (C.53) is preferable to (C.55) because it will be faster.

Finally, y is obtained by substitution in the polar form and solving for y

$$y = \frac{(\rho - x\cos\theta)}{\sin\theta} \tag{C.56}$$

$$= \frac{\rho}{\sin\theta} - \frac{x}{\tan\theta} \tag{C.57}$$

$$= \frac{\rho}{\sin\theta} + x\tan\left(\theta + \frac{\pi}{2}\right) \tag{C.58}$$

where ρ and θ are taken from the line where $\sin\theta$ is greater to avoid the loss of precission due to the quotient. When $\sin\theta$ and $\cos\theta$ are precomputed, (C.56) is the safer and faster choice.

Furthermore, when $\theta_0 = \theta_1$ lines l_0 and l_1 are parallel and if $\rho_0 = \rho_1$ too, they are colinear —i.e. $l_0 = l_1$. The intersection point p is

$$p = \begin{cases} \text{none} & \text{if } \theta_0 = \theta_1 \begin{cases} \text{colinear } (l_0 = l_1) & \text{if } \rho_0 = \rho_1 \\ \text{parallel } (l_0 \parallel l_1) & \text{others} \end{cases} \\ (x, y) & \text{others} \end{cases} \tag{C.59}$$

C.3 Point-line distance

Consider the cartesian implicit representation

$$ax + by + c = 0 \tag{C.60}$$

The point-line distance is given by projecting vector $\mathbf{r}$ (from the point to the

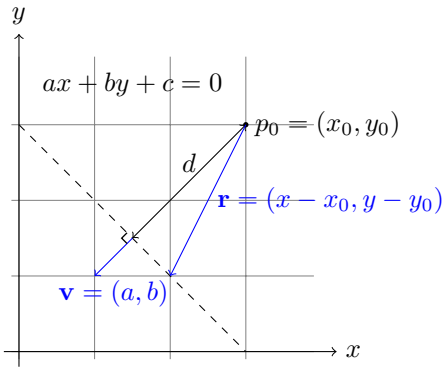


Figure C.5: Point-Line distance using projection

line) onto $\mathbf{v}$ (perpendicullar to the line)

$$d = |\text{proj}_{\mathbf{v}}\mathbf{r}| \tag{C.61}$$
$$= \frac{|\mathbf{v} \cdot \mathbf{r}|}{\mathbf{v}} \tag{C.62}$$
$$= |\hat{\mathbf{v}} \cdot \mathbf{r}| \tag{C.63}$$
$$= \left| \frac{a(x - x_0) + b(y - y_0)}{\sqrt{a^2 + b^2}} \right| \tag{C.64}$$
$$= \left| \frac{ax + by - ax_0 - by_0}{\sqrt{a^2 + b^2}} \right| \tag{C.65}$$
$$= \left| \frac{ax_0 + by_0 + c}{\sqrt{a^2 + b^2}} \right| \tag{C.66}$$

If the cartesian implicit representation is normalize, i.e. $a^2 + b^2 = 1$ the distance can be computed faster:

$$d = |ax_0 + by_0 + c| \tag{C.67}$$

If we used the explicit cartesian form $y = mx + b$, we take

$$A = m \tag{C.68}$$
$$B = -1 \tag{C.69}$$
$$C = b \tag{C.70}$$

hence

$$d = \left| \frac{mx_0 - y_0 + b}{\sqrt{m^2 + 1}} \right| \tag{C.71}$$

If we used the polar form $\rho = x \cos\theta + y \sin\theta$, we take

$$A = \cos\theta \tag{C.72}$$
$$B = \sin\theta \tag{C.73}$$
$$C = -\rho \tag{C.74}$$

hence

$$d = \left| \frac{x_0 \cos\theta + y_0 \sin\theta - \rho}{\sqrt{\cos^2\theta + \sin^2\theta}} \right| \tag{C.75}$$

$$= |x_0 \cos\theta + y_0 \sin\theta - \rho| \tag{C.76}$$

The polar form is actually a normal implicit cartesian form, with two parameters instead of three.

Proof. Both distances expressions, obtained from explicit cartesian and polar equation, are actually equivalent.

$$d = |x_0 \cos\theta + y_0 \sin\theta - \rho| \tag{C.77}$$

$$= |\sin\theta| \left| \frac{x_0}{\tan\theta} + y_0 - \frac{\rho}{\sin\theta} \right| \tag{C.78}$$

where $\frac{1}{\tan\theta} = -m$ and $\frac{\rho}{\sin\theta} = b$. Hence

$$d = |\sin\theta| \, |m x_0 - y_0 + b| \tag{C.79}$$

At this point we only have to proof that $|\sin\theta| = \frac{1}{|\sqrt{m^2+1}|}$. Substituting $m = -\frac{1}{\tan\theta}$ yields

$$\frac{1}{|\sqrt{m^2+1}|} = \frac{1}{\left|\sqrt{\left(-\frac{1}{\tan\theta}\right)^2 + 1}\right|} \tag{C.80}$$

$$\frac{1}{|\sqrt{m^2+1}|} = \frac{1}{\left|\sqrt{\frac{1}{\tan^2\theta} + 1}\right|} \tag{C.81}$$

$$\frac{1}{|\sqrt{m^2+1}|} = \frac{1}{\left|\sqrt{\frac{\cos^2\theta + \sin^2\theta}{\sin^2\theta}}\right|} \tag{C.82}$$

$$\frac{1}{|\sqrt{m^2+1}|} = \frac{1}{\left|\sqrt{\frac{1}{\sin^2\theta}}\right|} \tag{C.83}$$

$$\frac{1}{|\sqrt{m^2+1}|} = \frac{1}{\frac{1}{|\sin\theta|}} \tag{C.84}$$

$$\frac{1}{|\sqrt{m^2+1}|} = |\sin\theta| \tag{C.85}$$

$$\tag{C.86}$$

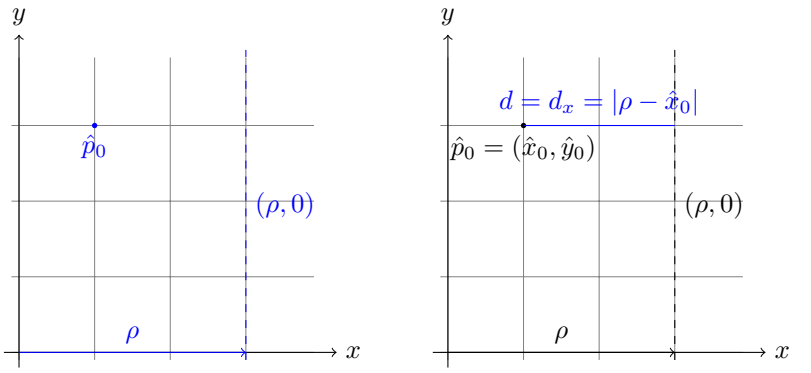
□

An alternative way to obtained the distance in terms of the polar equations is obtained applying the $-\theta$ rotation shown in Figure C.6 (a) to both line and point, in order to have a vertical line with $\theta = 0$. Therefore, the distance can be computed through the x-axis only, i.e. $d = d_x = |\rho - \hat{x}_0|$, as Figure C.6 (b) shows, and later rotated θ to undo the initial transformation.

$$d = |\rho - \hat{x}_0| \tag{C.87}$$

$$= |\rho - (x_0 \cos\theta + y_0 \sin\theta)| \tag{C.88}$$

$$= |x_0 \cos\theta + y_0 \sin\theta - \rho| \tag{C.89}$$



(a) Transformation applying $-\theta$ rotation

(b) Horizontal distance, after $-\theta$ rotation

Figure C.6: Point-Line distance using transformation

Bibliography

[1] Numerical Recipes. The Art of Scientific Computing. Source Code CD-ROM v3.02. CD-ROM, 2007.

[2] Pioneer P3-DX. Mobile Robots, Inc., 2008.

[3] Kai Oliver Arras. Introduction to Mobile Robotics — Line Extraction. First-Order Error Propagation. Presentation, 2006.

[4] Kai Oliver Arras and Roland Y. Siegwart. Feature Extraction and Scene Interpretation for Map-Based Navigation and Map Building. In *Proceedings of SPIE, Mobile Robotics XII*, volume 3210, Swiss Federal Institute of Technology Lausanne, Institute of Microengineering, 1997.

[5] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, February 2002.

[6] Josep Aulinas. MSc. Thesis dissertation: 3D Visual SLAM applied to large-scale underwater scenarios, 2008. A Thesis Submitted for the Degree of MSc Erasmus Mundus in Vision and Robotics (VIBOT).

[7] Tim Bailey. *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments*. PhD thesis, University of Sydney, Australian Centre for Field Robotics. Department of Aerospace, Mechanical and Mechatronic Engineering. The University of Sydney, August 2002.

[8] A. V. Balakrishnan. *Kalman Filtering Theory*. University Series in Modern Engineering. Optimization Software, Inc., 1984.

[9] Yaakov Bar-Shalom and Thomas E. Fortmann. *Tracking and Data Association*, volume 179 of *Mathematics in Science and Engineering*. Academic Press, Inc., Boston, 1st (3rd printing) edition, 1988.

[10] Yaakov Bar-Shalom, X.-Rong Li, and Thiagalingam Kirubarajan. *Estimation with Applications To Tracking and Navigation*. Wiley-Interscience, 2001.

[11] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359, 2008.

[12] P. J. Besl and H. D. Mckay. A method for registration of 3-D shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256, 1992.

[13] Jose-Luis Blanco, Juan-Antonio Fernández-Madrigal, and Javier Gonzalez. A New Approach for Large-Scale Localization and Mapping: Hybrid Metric-Topological SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'07)*, 2007.

[14] Jose-Luis Blanco, Juan-Antonio Fernández-Madrigal, and Javier Gonzalez. Towards a Unified Bayesian Approach to Hybrid Metric-Topological SLAM. *IEEE Transactions on Robotics*, 24(2):259–270, 2008.

[15] Miodrag Bolić, Petar M. Djurić, and Sangjin Hong. Resampling Algorithms for Particle Filters: A Computational Complexity Perspective. Technical report, Department of Electrical and Computer Engeering, Stony Brook University, Stony Brook, New York 11794, USA, January 2004.

[16] A. Bonarini, W. Burgard, G. Fontana, M. Matteucci, D. G. Sorrenti, and J. D. Tardos. RAWSEEDS: Robotics Advancement through Web-publishing of Sensorial and Elaborated Extensive Data Sets. *IROS'06 Workshop on Benchmarks in Robotics Research*, pages 1–5, 2006.

[17] Geovany Araujo Borges and Marie-José Aldon. A Split-and-Merge Segmentation Algorithm for Line Extraction in 2-D Range Images. In IEEE Computer Society, editor, *15th Proceedings of the International Conference on Pattern Recognition (ICPR'00)*, volume 1, page 1441, Robotics Department, LIRMM, UMR CNRS/Université Montpellier II, 2000.

[18] Geovany Araujo Borges and Marie-José Aldon. Line Extraction in 2D Range Images for Mobile Robotics. *Journal of Intelligent and Robotic Systems*, 40:267–297, 2004. Netherlands.

[19] Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomas Lozano-Perez, and Matthew T. Mason. *Robot Motion. Planning and Control*. The MIT Press series in Artifitial Intelligence. The MIT Press, Cambridge, Massachussets. London, England, 4th printing edition, 1984.

[20] Eli Brookner. *Tracking and Kalman Filtering Made Easy*. Wiley-Interscience, 1998.

[21] R. G. Brown and P. Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering*. John Wiley & Sons, Inc., 2 edition, 1992.

[22] Robert Grover Brown. *Introduction to Random Signal Analysis and Kalman Filtering*. John Wiley & Sons, 1st (3rd printing) edition, 1983.

[23] Wolfram Burgard, Dieter Fox, Daniel Hennig, and Timo Schmidt. Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 896–901, Menlo Park, 1996. AAAI, AAAI Press/MIT Press.

[24] James Carpenter, Peter Clifford, and Paul Fearnhead. Building robust simulation-based filters for evolving data sets. Technical report, Department of Statistics, University of Oxford, 1999.

[25] James Carpenter, Peter Clifford, and Paul Fearnhead. An Improved Particle Filter for Non-linear Problems. *Radar, Sonar and Navigation, IEE Proceedings*, 146(1):2–7, February 1999. Systematic Resampling, Variance Reduction.

[26] Jose A. Castellanos, J. M. Martínez, J. Neira, and J. D. Tardós. Simultaneous Map Building and Localization for Mobile Robots: A Multisensor Fusion Approach. In *IEEE International Conference on Robotics and Automation*, pages 1244–1249, Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, May 1998. IEEE.

[27] Jose A. Castellanos, J. M. M. Montiel, J. Neira, and J. D. Tardós. The SPmap: A Probabilistic Framework for Simultaneous Localization and Map Building. In *IEEE Transactions on Robotics and Automation*, volume 15, pages 948–952. IEEE, October 1999.

[28] Raja Chatila and Jean-Paul Laumond. Position Referencing and Consistent World Modeling for Mobile Robots. *IEEE International Conference on Robotics and Automation*, 1985.

[29] Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, and Lydia Kavraki. *Principles of Robot Motion. Theory, Algorithms, and Implementation*. The MIT Press, Cambridge, Massachusetts. London, England, 2005.

[30] Henrik I. Christensen. Feature Detection. Presentation, 2004. Centre for Autonomous Systems, Kungl Tekniska Högskolan.

[31] Jose Luis Blanco Claraco. *Development of Scientific Applications with the Mobile Robot Programming Toolkit. The MRPT reference book*. Machine Perception and Intelligent Robotics Laboratory, University of Málaga, Málaga, Spain, December 2008.

[32] James L. Crowley. World Modeling and Position Estimation for a Mobile Robot Using Ultrasonic Ranging. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, volume 3, pages 674–680, May 1989.

[33] Michael Csorba. *Simultaneous Localisation and Map Building*. PhD thesis, Balliol College, Dept. of Engineering Science, University of Oxford, Robotics Research Group. Department of Engineering Science. University of Oxford, 1997.

[34] Murray Cumming, Bernhard Rieder, Jonathon Jongsma, Jason M'Sadoques, Ole Laursen, Gene Ruebsamen, Cedric Gustin, Marko Anastasov, and Alan Ott. *Programming with Gtkmm*, 2006.

[35] Fred Daum. Nonlinear Filters: Beyond the Kalman Filter. *IEEE A&E Systems Magazine*, 20(8):57–69, August 2005. Part 2: Tutorials.

[36] Andrew John Davison. *Mobile Robot Navigation Using Active Vision*. PhD thesis, Keble College, Robotics Research Group. Department of Engineering Science. University of Oxford, June 1998.

[37] P. de Groen. Title: An Introduction to Total Least Squares. *Nieuw Archief voor Wiskunde*, 14:237–253, 1996.

[38] Julio Delgado Mangas. Evaluación y Prueba del Algoritmo FastSLAM de Construcción de Mapas para un Robot Móvil, November 2007.

[39] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[40] Jurgen A. Doornik. An Improved Ziggurat Method to Generate Normal Random Samples. University of Oxford, 2005.

[41] Arnaud Doucet. Lecture 3: Sequential Importance Sampling Resampling. Presentation, April 2009. Departments of Statistics and Computer Science. University of British Columbia.

[42] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[43] Arnaud Doucet and Adam M. Johansen. *A Tutorial on Particle Filtering and Smoothing: Fifteen years later*, December 2008.

[44] Tom Duckett and Ulrich Nehmzow. Mobile Robot Self-Localisation Using Occupancy Histograms and A Mixture of Gaussian Location Hypotheses. *Journal of Robotics and Autonomous Systems*, 34(2–3):119–130, January 2001.

[45] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. Wiley-Interscience, Standford Research Institute, Menlo Park, California, 1st (28th printing) edition, 1973.

[46] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd (7th printing) edition, 2001.

[47] Alberto Elfes. Sonar-based Real-World Mapping and Navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

[48] Alberto Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1989.

[49] F. Ferreira, I. Amorim, R. Rocha, and J. Dias. T-SLAM: Registering Topological and Geometric Maps for Robot Localization in Large Environments. In *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 392–398, Seoul, Korea, August 2008.

[50] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[51] George S. Fishman. *Monte Carlo. Concepts, Algorithms, and Applications*. Springer Series in Operations Research. Springer, United States of America, 1st (6th printing) edition, 1996.

[52] John Folkesson and Henrik Christensen. Graphical SLAM — A Self-Correcting Map. In *In IEEE International Conference on Robotics and Automation (ICRA)*, pages 383–390, New Orleans, USA, 2004.

[53] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, us ed edition, August 2002.

[54] Dieter Fox. KLD-Sampling: Adaptive Particle Filters. In *In Advances in Neural Information Processing Systems 14*, pages 713–720. MIT Press, 2001.

[55] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov Localization for Mobile Robots in Dynamic Environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[56] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Patrones de Diseño. Elementos de software orientado a objetos reutilizable*. Pearson Education, S.A., Núñez de Balboa, 120. 28006 Madrid, 1st edition, 2003. Spanish edition, translated from Design Patterns: Elements of Reusable Object-Oriented Software.

[57] Walter Gander, Gene H. Golub, and Rolf Strebel. Least-Squares Fitting of Circles and Ellipses. *BIT Numerical Mathematics*, 34(4):558–578, December 1994.

[58] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR'03)*, pages 317–323, Coimbra, June 2003.

[59] Jonathan Goodman. *Lecture Notes on Monte Carlo Methods*, chapter Chapter 2: Simple Sampling of Gaussians. Courant Institute of Mathematical Sciences, NYU, August 2005.

[60] Neil Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *Proceedings F of the IEE Radar and Signal Processing*, 140(2):107–113, April 1993. Multinomial Sampling.

[61] Mohinder S. Grewal and Angus P. Andrews. *Kalman Filtering: Theory and Practice. Using MATLAB*. Wiley-Interscience, 2nd edition, 2001.

[62] W. Grimson and T. Lozano-Perez. Localizing Overlapping Parts by Searching the Interpretation Tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(4):469–482, July 1987.

[63] W. E. L. Grimson. *Object Recognition by Computer: The Role of Geometric Constraints*. The MIT Press, Cambridge, Massachusetts, 1990.

[64] Jose Guivant and Eduardo Nebot. Optimization of the Simultaneous Localization and Map Building Algorithm for Real Time Implementation. *IEEE Transactions on Robotics and Automation*, 17:242–257, 2001.

[65] Dirk Haehnel, Dirk Schulz, and Wolfram Burgard. Map Building with Mobile Robots in Populated Environments. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[66] J. M. Hammersley and D. C. Handscomb. *Monte Carlo Methods*. Methuen's Monographs on Applied Probability and Statistics. Methuen & Co. Ltd., Fletcher & Son Ltd., Norwich, London, 1964. Catalogue No. (Methuen) 12/5234/64.

[67] Chris G. Harris and Mike J. Stephens. A Combined Corner and Edge Detector. In *Proceedings of Forth Alvey Vision Conference*, pages 147–151, 1988.

[68] Andrew Howard and Nicholas Roy. The Robotics Data Set Repository (Radish), 2003.

[69] Inseok Hwang, Hamsa Balakrishnan, Kaushik Roy, Jaewon Shin, Leonidas Guibas, and Claire Tomlin. Multiple-Target Tracking and Identity Management. In *Proceedings of IEEE Sensors*, volume 1, pages 36–41, October 2003.

[70] SICK Sensor Intelligence. Laser Measurement Technology. LMS2xx / LMS200 / Indoor / Short-Range. Online Data Sheet, 2009.

[71] Patric Jensfelt and Henrik I. Christensen. Laser Based Position Acquisition and Tracking in an Indoor Environment. In *Proceedings of the IEEE International Symposium Robotics and Automation*, volume 1, 1998.

[72] S. Julier and J. Uhlmann. Building a Million Beacon Map. In *SPIE Sensor Fusion*, 2001.

[73] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[74] Suhyeon Kim, Hyungrae Kim, and Tae-Kyu Yang. Increasing SLAM Performance by Integrating Grid and Topology Map. *Journal of Computers*, 4(7):601–609, July 2009.

[75] Genshiro Kitagawa. Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5(1):1–25, 1996. Deterministic and Stratified Resampling.

[76] Joss Knight, Andrew Davison, and Ian Reid. Towards Constant Time SLAM using Postponement. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2001.

[77] Kurt Konolige and Ken Chou. Markov Localization using Correlation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[78] Samuel Kotz and Johan René van Dorp. *Beyond Beta. Other Continuous Families of Distributions with Bounded Support and Applications*, chapter 1: The Triangular Distribution. World Scientific, December 2004.

[79] Benjamin Kuipers and Yung tai Byun. A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[80] John J. Leonard and Hugh F. Durrant-Whyte. Mobile Robot Localization by Tracking Geometric Beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, June 1991.

[81] Jun S. Liu and Rong Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93:1032–1044, 1998.

[82] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, January 2004.

[83] F. Lu and E. Milios. Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4:333–349, 1997.

[84] Feng Lu and Evangelos Milios. Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. *Journal of Intelligent and Robotic Systems*, 18:249–275, 1994.

[85] William G. Madow. On the Theory of Systematic Sampling, II. *The Annals of Mathematical Statistics*, 20(3):333–354, 1949.

[86] William G. Madow. On the Theory of Systematic Sampling, III. Comparison of Centered and Random Start Systematic Sampling. *The Annals of Mathematical Statistics*, 24(1):101–106, 1953.

[87] William G. Madow and Lilliam H. Madow. On the Theory of Systematic Sampling, I. *The Annals of Mathematical Statistics*, 15(1):1–24, 1944.

[88] George Marsaglia and Wai Wan Tsang. The Ziggurat Method for Generating Random Variables. *Journal of Statistical Software*, 5(8):1–7, October 2000.

[89] Maja J. Matarić. A Distributed Model for Mobile Robot Environment-Learning and Navigation. Technical report, MIT Artificial Intelligence Laboratory, Cambridge, MA, USA, 1990.

[90] Kurt Mehlhorn and Stefan Näher. Implementation of a Sweep Line Algorithm for the Straight Line Segment Intersection Problem. Technical report, TR MPI-I-94-105, Max-Planck-Institut für Informatik, Saarbrücken, 66123 Saarbrücken, Germany, 1994.

[91] Adam Milstein and Tao Wang. Dynamic motion models in Monte Carlo Localization. *Integrated Computer-Aided Engineering*, 14:243–262, 2007.

[92] Michael Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, June 2003.

[93] Michael Montemerlo and Sebastian Thrun. *FastSLAM. A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*, volume 27 of *Springer Tracts in Advanced Robotics (STAR)*. Springer, Germany, 1st edition, 2007.

[94] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598, Edmonton, Canada, 2002. AAAI.

[95] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *In Proceedings of the International Conference on Artificial Intelligence (IJCAI)*, pages 1151–1156, 2003.

[96] Michael Montermerlo and Sebastian Thrun. Simultaneous Localization and Mapping with Unknown Data Association using FastSLAM. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation*, 2003.

[97] Hans P. Moravec. Sensor Fusion in Certainty Grids for Mobile Robots. *AI Magazine*, 9(2):61–74, 1988.

[98] Kevin P. Murphy. Bayesian Map Learning in Dynamic Environments. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Neural Information Processing Systems (NIPS)*, pages 1015–1021. MIT Press, 2000.

[99] Eduardo Nebot, Favio Masson, Jose Guivant, and H. Durrant-Whyte. Robust Simultaneous Localization and Mapping for Very Large Outdoor Environments. In B. Siciliano and P. Dario, editors, *Proceedings of the 8th International Symposium on Experimental Robotics (ISER)*, volume 5, pages 200–209, 2003.

[100] José Neira and Juan D. Tardós. Data Association in Stochastic Mapping using the Joint Compatibility Test. *IEEE Transactions on Robotics and Automation*, 17(6):890–897, 2001.

[101] P. Newman, J. Leonard, J. D. Tardós, and J. Neira. Explore and Return: Experimental Validation of Real-Time Concurrent Mapping and Localization. In *IEEE International Conference on Robotics and Automation*, pages 1802–1809, Washington, DC, USA, May 2002. IEEE.

[102] Viet Nguyen, Agostino Martinelli, Nicola Tomatis, and Roland Siegwart. A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics. In *Conference on Intelligent Robots and Systems, IROS'2005*, Autonomous Systems Laboratory, Ecole Polytechnique Fédérale de Lausanne (EPFL), 2005. IEEE.

[103] Juan Nieto, Jose Guivant, Eduardo Nebot, and Sebastian Thrun. Real Time Data Association for FastSLAM. In *Proceedings of the IEEE Conference on Robotics and Autonomation, Taipeh*, pages 412–418, 2003.

[104] Andreas Nüchter. *3D Robotic Mapping. The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*, volume 52 of *Springer Tracts in Advanced Robotics (STAR)*. Springer, Germany, 1st edition, 2009.

[105] Mourad Oussalah and Joris De Schutter. Hybrid fuzzy probabilistic data association filter and joint probabilistic data association filter. *Information Sciences*, 142(1–4):195–226, May 2002.

[106] Jennifer Owen. *How to Use Player/Stage*, July 2009.

[107] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill series in Electrical Engineering. Communications and Signal Processing. McGraw-Hill, Inc., 3rd edition, 1991.

[108] Mark A. Paskin. Thin Junction Trees Filters for Simultaneous Localization and Mapping. Technical Report UCB/CSD-02-1198, Computer Science Division. University of California, Berkeley, 2003.

[109] Theodosios Pavlidis and Steven L. Horowitz. Segmentation of Plane Curves. In *IEEE Transactions on Computers*, volume c-23, pages 860–874, August 1974.

[110] L. M. Paz, P. Jensfelt, J. D. Tardós, and J. Neira. EKF SLAM updates in $\mathcal{O}(n)$ with Divide and Conquer SLAM. *2007 IEEE International Conference on Robotics and Automation*, pages 1657–1663, April 2007.

[111] Lina M. Paz, Pedro Piniés, Juan D. Tardós, and José Neira. Large Scale 6DOF SLAM with Stereo-in-Hand. *IEEE Transactions on Robotics*, 24(5):1–12, October 2008.

[112] S. T. Pfister, S. I. Roumeliotis, and J. W. Burdick. Weighted Line Fitting Algorithms for Mobile Robot Map Building and Efficient Data Representation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 1304–1311, September 2003.

[113] Cristiano Premebida and Urbano Nunes. Segmentation and Geometric Primitives Extraction from 2D Laser Range Data for Mobile Robot Applications. In *Robótica 2005: Scientific Meeting of the 5 th National Robotics Festival*, pages 17–25, Coimbra, Portugal, April 2005.

[114] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, The Pitt Building, Trumpington Street, Cambridge CB2 1RP. United States of America, 2nd edition, 1992.

[115] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes. The Art of Scientific Computing*. Cambridge University Press, 32 Avenue of the Americas, New York, NY 10013-2473, USA, 3 edition, 2007. Printing corrected to software version 3.02.

[116] William Gareth Rees. *Physical Principles of Remote Sensing*. Cambridge University Press, Cambridge, UK, 2nd edition, 2001.

[117] Donald B. Reid. An Algorithm for Tracking Multiple Targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979.

[118] Kristof Richmond. *Real-Time Visual Mosaicking and Navigation on the Seafloor*. PhD thesis, Standford University, Department of Mechanical Engineering. Standford University, March 2009.

[119] Søren Riisgaard and Morten Rufus Blas. *SLAM for Dummies. A Tutorial Approach to Simultaneous Localization and Mapping*, May 2004.

[120] Leonardo Romero and Carlos Lara. A Probabilistic Approach to Build 2D Line Based Maps From Laser Scans in Indoor Environments. In *Iberoamerican Congress in Pattern Recognition*, volume 4225 of *11th Iberoamerican congress in pattern recognition, CIARP 2006*, pages 733–742, Cancun, Mexico, November 2006. Springer.

[121] Leonardo Romero and Carlos Lara. A Robust Approach to Build 2D Line Maps From Laser Scans. *Lecture Notes in Computer Science*, pages 733–742, 2006. Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics.

[122] Daniel Sack and Wolfram Burgard. A Comparison of Methods for Line Extraction from Range Data. In *Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, University of Freiburg, Dept. of Computer Science, Germany, 2003.

[123] Bernt Schiele and James L. Crowley. A Comparison of Position Estimation Techniques Using Occupancy Grids. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1628–1634, 1994.

[124] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

[125] Joris De Schutter, Jan De Geeter, Tine Lefebvre, and Herman Bruyninckx. *Kalman Filters: A Tutorial*. Belgium, October 1999.

[126] Takashi Shinzato. Box Muller Method. January 2007.

[127] Ali Siadat, Axel Kaske, Siegfried Klausmann, Michel Dufaut, and René Husson. An Optimized Segmentation Method for a 2D Laser-Scanner Applied to Mobile Robot Navigation. In *Intelligent Components and Instruments for Control Applications 1997 (SICICA'97)*, pages 149–154, 1997.

[128] SICK. *LMS200/211/221/291 Laser Measurement Systems. Technical Description*. SICK Sensor Intelligence, SICK AG Waldkirch. Auto Ident, Reute Plant. Nimburger Strasse 11, 79276 Reute, Germany, 2006.

[129] Roland Siegwart and Illah Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, Cambridge, Massachusetts, 2004.

[130] Jeremy G. Siek and Andrew Lumsdaine. The Matrix Template Library: A Unifying Framework for Numerical Linear Algebra. In *Parallel Object Oriented Scientific Computing (ECOOP)*, pages 466–467, Laboratory for Scientific Computing. Department of Computer Science and Engineering. University of Notre Dame, 1998. Springer-Verlag.

[131] Eero Simoncelli. Least Squares Optimization. Technical report, Center for Neural Science and Courant Institute of Mathematical Sciences, July 2003.

[132] Randall Smith, Matthew Self, and Peter Cheeseman. A Stochastic Map for Uncertain Spatial Relationships. In *The Fourth International Symposium*, Cambridge, MA, USA, 1988. MIT Press.

[133] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating Uncertain Spatial Relationships in Robotics. *Autonomous Robot Vehicles*, pages 167–193, 1990.

[134] Randall C. Smith and Peter Cheeseman. On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5(4):56–68, 1987.

[135] Alan D. Sokal. Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms. Technical report, Department of Physics, New York University, 4 Washington Place, New York, NY 10003, USA, September 1996.

[136] Cyrill Stachniss, Udo Frese, and Giorgio Grisetti. OpenSLAM. Web page, 2009.

[137] Richard Stallman, Roland Pesch, and Stan Shebs. *Debugging with GDB*. Free Software Foundation, 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA, 9 edition, July 2009. GDB version 6.8.50.20090702.

[138] P. Swerling. A Proposed Stagewise Differential Correction Procedure for Satellite Tracking and Prediction. Technical Report P-1292, RAND Corporation, 1958.

[139] Juan D. Tardos, Jose Neira, Paul M. Newman, and John J. Leonard. Robust Mapping and Localization in Indoor Environments using Sonar Data. *International Journal of Robotics Research*, 21:311–330, 2002.

[140] R. M. Taylor and P. J. Probert. Range Finding and Feature Extraction by Segmentation of Images for Mobile Robot Navigation. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation (ICRA)*, pages 95–100, April 1996.

[141] David B. Thomas, Wayne Luk, Philip H.W. Leong, and John D. Villasenor. Gaussian Random Number Generators. *ACM Computing Surveys*, 39(4):38, October 2007. Article 11.

[142] Stephen J. Thomas. Real-time Stereo Visual SLAM, 2008. A Thesis Submitted for the Degree of MSc Erasmus Mundus in Vision and Robotics (VIBOT).

[143] Stephen J. Thomas. Real-time Stereo Visual SLAM for an AUV. Presentation, June 2008.

[144] Stephen J. Thomas, Joaquin Salvi, and Yvan Petillot. Real-time stereo visual SLAM for autonomous underwater vehicles. 2008.

[145] Chuck Thorpe and Hugh Durrant-whyte. Field Robots. In *International Journal of Pattern Recognition and Artificial Intelligence*, pages 39–7, 2001.

[146] Sebastian Thrun. A Probabilistic Online Mapping Algorithm for Teams of Mobile Robots. *International Journal of Robotics Research*, 20:335–363, 2001.

[147] Sebastian Thrun. Robotic Mapping: A Survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[148] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Intelligent robotics and autonomous agents series. The MIT Press, 55 Hayward Street, Cambridge, MA 02142, 1st (2nd printing) edition, 2005.

[149] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Intelligent robotics and autonomous agents series. The MIT Press, 55 Hayward Street, Cambridge, MA 02142, 1st (3rd printing) edition, 2006.

[150] Sebastian Thrun, Daphne Koller, Zoubin Ghahramani, Hugh Durrant-Whyte, and Andrew Y. Ng. Simultaneous Mapping and Localization with Sparse Extended Information Filters. In *Proceedings of WAFR*, 2002.

[151] Sebastian Thrun, Michael Montemerlo, Daphne Koller, Ben Wegbreit, Juan Nieto, and Eduardo Nebot. FastSLAM: An Efficient Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association. Journal of Machine Learning Research, 2004 (in press), 2004.

[152] Sebastian B. Thrun. Exploration and Model Building in Mobile Robot Domains. In E. Ruspini, editor, *Proceedings of the 1993 IEEE International Conference on Neural Networks (ICNN)*, pages 175–180, San Francisco, CA, 1993.

[153] Carlo Tomasi and Takeo Kanade. Shape and Motion from Image Streams: a Factorization Method. Technical Report CMU-CS-92-104, Carnegie Mellon University, 1992.

[154] P. H. S. Torr and A. Zisserman. MLESAC: A New Robust Estimator with Application to Estimating Image Geometry. *Computer Vision and Image Understanding*, 78:138–156, 2000.

[155] Phil Torr and Andrew Zisserman. Robust Computation and Parametrization of Multiple View Relations. *Sixth International Conference on Computer Vision*, pages 727–732, January 1998.

[156] J. Vandorpe, H. Van Brussel, and H. Xu. Exact Dynamic Map Building for a Mobile Robot using Geometrical Primitives Produced by a 2D Range Finder. In *Proceedings of the 1996 IEEE Intemational Conference on Robotics and Automation*, pages 901–908. IEEE, April 1996.

[157] Eric W. Weisstein. Least Squares Fitting. From MathWorld — A Wolfram Web Resource, 2009.

[158] Eric W. Weisstein. Least Squares Fitting — Perpendicular Offsets. From MathWorld — A Wolfram Web Resource, 2009.

[159] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical Report 95-041, Department of Computer Science. University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, July 2006.

[160] Brian Yamauchi and Pat Langley. Place Recognition in Dynamic Environments. *Journal of Robotic Systems*, 14:107–120, 1997.

[161] Marco Zuliani. *RANSAC for Dummies*, November 2008. With examples using the RANSAC toolbox for Matlab and more...

[162] Erol Şahin and Paolo Gaudiano. Visual Looming as a Range Sensor for Mobile Robots. In *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, pages 114–119, Zurich, Switzerland, 1998. Also appears as Boston University Technical Report CAS/CNS-TR-98-007.

[163] Erol Şahin, Paolo Gaudiano, and Robert Wagner. A Comparison of Visual Looming and Sonar as Mobile Robot Range Sensors. In *Proceedings of the Second International Conference on Cognitive And Neural Systems*, Boston, MA, May 1998.

List of Tables

List of Figures

List of Algorithms

Index