UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXI Ciclo

COMPUTATION AND TIME CONSTRAINTS IN LOCALIZATION AND MAPPING PROBLEMS

Coordinatore:

Chiar.mo Prof. Carlo Morandi

Tutor:

Chiar.mo Prof. Stefano Caselli

Dottorando: *Dario Lodi Rizzini*

Gennaio 2009

Alla mia famiglia

Contents

Chapter 1

Introduction

1.1 Localization and Mapping Problems

The key issue of mobile robotics is the interaction between the robot and the environment. The robot acquires information about the environment through sensors and changes the configuration of environment through actuators. A robotic task may be generally seen as a controlled exchange of sensor observations and motion commands conceived by a designer to achieve useful goals. The autonomy and intelligence of robots depend on their ability to answer to the inputs from the environment, which is complex and unpredictable. Moreover, even if the environment could be modelled completely, robot perception and motion are noisy and limited.

Several approaches have been proposed to design autonomous and intelligent robots. The *reactive* approach claims that, since a complete representation of environmental settings is not possible, a robotic task should be designed as a set of concurrent behaviors. A behavior is a function that maps sensor observations to actions without the capability of reasoning on a model of the world. While the imitation of "instinctive" behaviors is effective and computationally efficient for low level activities, more complex tasks, like planning the path to reach a goal position, require a representation of the environment.

Thus, a robot needs a geometric description of the world and an assessment of its space relationship inside this setting. *Localization* is the problem of estimating the robot pose, which is the union of position and orientation, with respect to an external reference frame. *Mapping* is the problem of integrating the information gathered with the robot sensors into a given representation, the map. When a map of the environment or an initial estimation of robot position are given, global localization or pose tracking can be addressed as independent issues. Otherwise the two complementary and strictly dependent problems constitute a unique *Simultaneous Localization and Mapping* (SLAM) problem. Localization is achieved by comparing observations with known references of the environment stored in a map. On the other hand, a set of observations can be arranged in a consistent map only if the locations, in which these observation have been acquired, is known. Therefore robot pose and environment map variables are estimated concurrently.

To perform the estimation the robot is given observations acquired by on board sensors and motion commands. These data have two major limitations. First, they provide *local* information since sensors have a limited visibility range. It is usually assumed that the inference on environment does not rely on a global off-board sensor infrastructure that provides additional information and the robot is independent from the context. An important consequence is that robot pose or observation arrangement in a map cannot be achieved by data integration: a single error in measurements produces a drift from the correct value. Correction of a wrong estimation depends on the comparison of the observation with the known map in localization and with a previously explored region of the map in SLAM.

The second limitation comes from the uncertainty on the knowledge about the world. Perception produces noisy information and the outcome of motion execution always differs from kinematic models description. Since uncertainty cannot be neglected in localization and mapping problems, the new paradigm of *probabilistic robotics* has been proposed [74]. In this paradigm, problem data are modelled as random variables and their relationships are expressed by Bayesian theory. According to the probabilistic paradigm localization and mapping problems have been formulated as stochastic state estimation problems. Robotics researchers have studied and proposed algorithms that can be classified in three main categories briefly listed in the following.

Extended Kalman Filters (EKFs) [45] are probably the earliest stochastic estimation techniques exploited for SLAM. Due to its strong mathematical foundation, the EKF is still the most used technique, even though its computational cost grows quadratically with the size of the state and it suffers from linearization error. Several variants and extensions derived from Kalman filters, like *Extended Information Filter* (EIF) and *Unscented Kalman Filters* (UKF), have been proposed to overcome these drawbacks [38, 37].

Particles filters [26] have been initially proposed in order to overcome the limitations of EKF in global localization: particle filters are suitable to represent any distribution of localization hypotheses with a discrete set of samples and not only the uni-modal one. Flexibility granted by *importance sampling* is balanced by the computational complexity due to the management of samples. Particle filters cannot be directly applied to solve SLAM problems due to the curse of dimensionality of state space. Particle filters have been subsequently adapted to SLAM by applying marginalization: the distribution of state is decomposed into robot pose distribution and map distribution that is then conditioned to robot pose. Thus, in *Rao-Blackwellized Particle Filters* (RBPF) importance sampling is used only to estimate robot pose and there is a map associated to each sample. Efficient implementations of RBPF allow sharing of map elements among several samples [62, 33].

Maximum Likelihood (ML) mapping is a quite different approach from recursive Bayesian filters. Mapping is formulated as a *full SLAM problem*, that is the evaluation of the map and all poses belonging to robot trajectory. Maximum Likelihood mapping is naturally formulated in the form of graphical model: robot poses and map features correspond to the nodes of the graph, while the constraints between pairs of variables derived from odometry information and observations are associated to the edges. The full SLAM posterior distribution is usually decomposed into the product of all constraints. The best map estimation corresponds to the graph configuration that maximizes the likelihood of the posterior and can be found solving a least square estimation problem.

1.2 Motivation

The brief overview in the previous section gives an idea of the extensive research of simultaneous localization and mapping problems carried out by robotics community in the last decade. Several subproblems –like data association, map representation, dynamic environment or semantic mapping– have been more or less intensively investigated. One of the most important questions is the *computational complexity* of the algorithms exploited to solve the problem.

Complexity strongly depends on the scale of the problem, or equivalently on the size of the environment. In global localization the number of localization hypotheses depends on the dimension of the map and on the presence of symmetric or indistinguishable regions of the map, albeit it is difficult to evaluate formally this dependence. Scale affects dramatically the mapping problem: since SLAM is a state estimation problem and map size increases with the exploration of new locations, map construction implies state augmentation and an increase of complexity. Thus, the evolution of SLAM algorithms has been devoted to address scale. *Extended Sparse Information Filters* [64, 74] have been introduced to better handle the quadratic increase of complexity of EKF. Marginalization is the trick exploited by RBPFs to cope with the curse of dimensionality of importance sampling methods. Recent popularity of maximum likelihood techniques with hybrid metric-topological maps is mainly due to their ability of better addressing large scale environments without numerical divergences.

Nonetheless, less effort has been devoted by the robotics community to another important implication of map scale: the increase of the *time* required to update state estimation with incoming data from the environment. Even if time for algorithm execution is related to computational complexity, the ability to respect time constraints in adapting robot belief about the world to world stimuli is a distinct issue. Robot motion information and sensor observations are usually acquired periodically. In several robotics applications the localization and mapping evaluation is useful only if it is provided concurrently with the robot activity. Therefore, online methods for localization and mapping are subjected to time constraints due to odometry and perception.

Such constraints are considered feasible when the system is able to process data according to given deadlines, even if the results of processing are delayed in time. In other words *real-time feasibility* depends on the design of the localization and mapping algorithms and may be granted by a proper distribution of computation in time intervals in spite of the computational complexity.

In literature there are examples of algorithmic adaptations that match or approach such feasibility. All the methods discussed in this thesis are derived from existing state-of-the-art algorithms. For example, the *Real-Time Particle Filter* [44] is an evolution of standard Monte-Carlo localization that meets time constraints. Furthermore, maximum likelihood methods for SLAM were originally applied offline and have been adapted to incremental construction of the map. The structure of the problem does not lend itself to easy adaptation to real-time operation, at least at the moment, but several solutions have been devised to reduce update time.

Beyond these partial contributions, no comprehensive discussion of real-time feasibility in localization and mapping problems has been proposed. Despite differences in the nature of each specific problem and in the solution methods, general design criteria for real-time or update time bounded localization and mapping can be found. In this thesis, we claim that *Locality* of the effects due to the newly added information is the fundamental criterion to address online execution of localization and mapping algorithms. Locality may be applied to perform temporal or spatial decomposition of the global estimation. In localization the complexity lies in the number of localization hypotheses tracked by the localizer before the convergence to the correct robot location. In this case the decomposition concerns the space of the hypotheses. Complexity of mapping algorithms depends on the relationships among the basic elements of the map. Locality of map estimation is then possible when these features are loosely correlated and the problem structure is sparse. Algorithms designed according to the locality principle may not always grant real-time feasibility, but they help online execution. Several existing localization and mapping solutions already exploit locality to reduce their complexity, but to our knowledge locality has never been explicitly related to time constrained execution.

1.3 Contribution

The two main contributions of this thesis are the identification of real-time issue in localization and mapping problem and the improvement and adaptation of algorithms according to this perspective. More specifically, the most important contributions of the thesis include:

- a general perspective of real-time feasibility and the identification of proximity decomposition as a general design criterion for algorithms to meet time constraints;
- a new enhanced version of *Real-Time Particle Filter* that is less prone to bias problem and to numeric divergence;
- the proposal of an incremental version of a maximum likelihood map estimator based on stochastic gradient descent;
- a parallel mapping algorithm that decomposes a graphical model into independent subproblems that can be solved on different processors.

1.4 Organization of the Thesis

The thesis is organized as follows. Chapter 2 illustrates the common probabilistic formulation of localization and mapping problems and discusses real-time feasibility related to the models that represent the structure of the problem and, in particular, to the graphical model. Then, classical Monte Carlo Localization and Maximum Likelihood methods for mappings are presented. These methods will be referred and developed in the rest of the dissertation.

Chapter 3 presents the Real-Time Particle Filter, a solution proposed to meet time constraints feasible in localization. In particular, the limitations and drawbacks of the original technique are formally interpreted using the key concept of effective sample size of the mixture representation and an improved version is proposed and compared experimentally with the original algorithm.

Chapter 4 describes a recent maximum likelihood method for robot mapping based on stochastic gradient descent and exploiting an efficient tree parameterization. An incremental version of this algorithm, originally conceived for offline map learning, is the presented. The incremental algorithm is then integrated into a system that estimates a map from odometry data and laser scans.

Chapter 5 illustrates a parallel maximum likelihood algorithm based on Gauss-Seidel relaxation. This parallel algorithm is potentially suitable for multi-robot map building. The general network representing the map is partitioned in clusters which are solved independently.

A final chapter summarizes this dissertation.

Chapter 2

Problem Model

In this chapter, localization and mapping problems are described in probabilistic formulation according to recent mainstream literature in robotics. Specific formulation of localization or mapping problems and different methods for their solution can be derived from the general problem using the concepts of marginalization and conditioning. Such unified perspective suggests that feasibility of time constraints in localization and mapping algorithm may be addressed with a uniform criterion like locality principle. Reduction of original problem in local subproblem is the basic procedure that will be adopted in the following chapters. The remaining of the chapter is devoted to the introduction of the methods that will be used in the thesis. In particular, the latest sections present particle filters for robot localization and maximum likelihood approaches to robot mapping.

2.1 Probabilistic formulation of the problem

2.1.1 Definitions

Localization and mapping problems are usually formulated as a stochastic state estimation. Uncertainty on both state variables and acquired data can be conveniently represented by using random variables. Depending on the specific problem, state may consist of the robot *pose* with respect to an external fixed reference frame or the *map* of the environment. Robot pose is usually labeled as x_t [1], where pedix t refers to a discrete time index. When a whole sequence of variables is involved in the estimation, we use the shorthand notation $x_{0:t}$ that represents the vector of values $x_0, \cdots, x_t$ assumed by a variable in time. Vector m contains the variables of map m_j and the interpretation of each map elements depends on the adopted representation for the map. Possible models for maps, feature or occupancy grid maps, will be discussed later. An important assumption about the map is that it does not change in time: the environment is assumed to be *static*. Furthermore, we assume the environment state fully represented by the mentioned variables, x_t and m. This assumption is called *complete state hypothesis* and is better discussed in the remaining of the chapter.

In the estimation two kinds of information sources are used: the sequence of motion commands $u_{1:t}$ and the sequence of observations $z_{1:t}$. Motion information u_t may be retrieved from control commands sent to the robot or from the measurements of robot encoders. The motion command vector may contain the speed commands or the relative displacement between consecutive robot poses. The interpretation of observation vector z_t depends strictly on the nature of sensor data. Different sensor devices give a quite different range of data: range sensors like sonars or laser scanners return mostly accurate measurements of distance to nearest obstacle, cameras provide bitmap images that require further processing, inertial measurement units maintain orientation and position information. Furthermore, data can be used in raw or processed form. Several examples of observations will be illustrated in the rest of the thesis.

Location and mapping estimations can be described as the problems of finding the posterior distribution function of state that consists of either robot pose x_t alone or the ensemble of robot trajectory in time and map $[x_{0:t}, m_{1:M}]^T$. In localization problems the posterior probability distribution is

$$p(x_t | u_{1:t}, z_{1:t}, m) \tag{2.1}$$

The above conditioned probability density function assumes that the map is known along with observations and robot motion. On the other hand the posterior for SLAM

[1] This notation sometimes appears unclear when compared to a specific coordinate.

problems is

$$p(x_t, m|u_{1:t}, z_{1:t}) \tag{2.2}$$

Sometimes additional association variables appear in the conditioning variables in the above conditioned distributions. Association variables model uncertainty on the matching between the observations and the map. They will be omitted to focus more on the significant issues than on notation.

The two posteriors differs in the composition of the variables to be estimated. Map is a known term and is part of the conditioning terms in posterior of localization problems, and part of the state when simultaneous localization and mapping is addressed. While the two problems may look totally apart, posterior Eq. (2.2) can be decomposed into the product of Eq. (2.1) and the probability density function of map variables m. Thus, probabilistic approach allows a unique formulation of different problems: an estimation problem can be reduced to another problem by explicitly acknowledging dependencies between variables. Such relationship may be derived by applying Bayes formula, conditioning, independence and conditional independence hypotheses or marginalization of joint distributions.

Therefore, it is convenient to provide a more general formulation of SLAM: in *full SLAM problem* the issue is the estimation of the posterior

$$p(x_{0:t}, m|u_{1:t}, z_{1:t}) \tag{2.3}$$

The aim is then the computation of map and whole robot trajectory. Depending on the specific application we may not be interested in the evaluation of the whole joint distribution. Anyway Eq. (2.3) may be reduced by variable elimination. Sometimes only a marginal distribution of the full state is required or the selective elimination of variables allows more efficient estimation.

Full SLAM formulation can be conveniently represented by an intuitive *graphical model* [23, 19, 74, 14] like the one depicted in figure 2.1. The graphical model is a graph whose vertices represent the random variables of the problem and arcs represent the relationships between these variables. Each arc is directed from the conditioning variable to the conditioned one. Such a graphical model is sometimes called *belief network* [14] and is similar for several aspects to Bayes networks and Markov

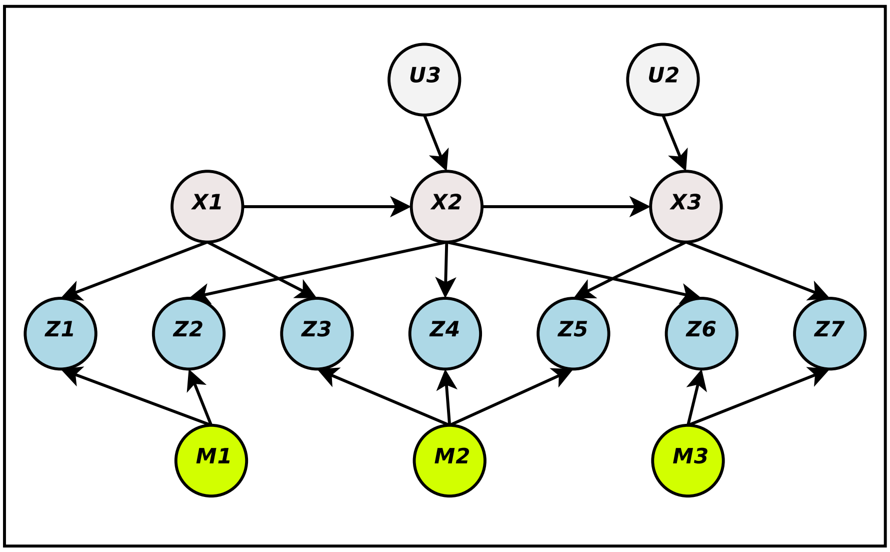


Figure 2.1: Graphical model of SLAM problem.

random fields [75]. Such representation has the advantage of being sufficiently abstract to provide a general interpretation of all approaches to SLAM. For example recursive bayesian filters may be viewed as a recursive elimination of the latest robot pose.

The graphical representation has also the advantage of catching dependencies among variables in term of connectivity of the graph. Such graph can also be interpreted topologically because the state variables to be estimated consist of robot poses and map features. The dependencies among variables have a significant impact on the estimation of posterior Eq. (2.3), since they affect the computational complexity of the estimating algorithms and the possible distribution of computational load in time. While the connectivity of the graphical model is given and depends on the specific problem, the solving algorithm may perform selective variable elimination or may select the part of the network to be processed. In online mapping problem the state is augmented after each step and the updated estimation should involve only the variables affected by each addition. Thus, the time required for the update respects real-time constraints, if the variables affected by the addition are local. The graphical model is then effective in emphasizing this locality condition.

2.1.2 Bayesian filters

Bayesian filtering is one of the most used approaches to stochastic state estimation problems. In order to estimate the posteriors of state, either Eq. (2.1) or Eq. (2.2), this method reverts the conditioning and the conditioned variables using Bayes formula. In particular, the inference of robot state x_t given the current sensor observation z_t is achieved by comparing the observation with the expected input given the robot pose x_t. In such comparison state variables x_t are the conditioning term. The expression rewritten with Bayes formula has the advantage of being easier to compute. Given the robot pose the expected observation can be obtained with a *generative model*. Generative models emulate the physical laws that rule the motion and perception of the robot.

The formal derivation of recursive bayesian filters is now an essential introduction to localization and mapping problems treatment [27, 74, 19]. First, the general posterior Eq. (2.2) is decomposed according to Bayes formula

$$\begin{aligned} p(x_t, m | u_{1:t}, z_{1:t}) &= \frac{p(z_t | x_t, m, u_{1:t}, z_{1:t-1}) \; p(x_t, m | u_{1:t}, z_{1:t-1})}{p(z_t | u_{1:t}, z_{1:t-1})} && (2.4) \\ &= \eta \; p(z_t | x_t, m) \; p(x_t, m | u_{1:t}, z_{1:t-1}) && (2.5) \end{aligned}$$

Since term on denominator contributes to normalization and is difficult to evaluate, it is commonly substituted with a normalization constant to be determined for each specific technique. The first term in Eq. (2.5) can be simplified due to the *Markov assumption* that postulates that the past and future data are independent if the current state x_t is known. Markov assumption is another name of complete state hypothesis. Indeed, the observation z_t does not depend on previous observations and commands, which can be removed from conditioning terms. The second factor in Eq. (2.5) represents the *prediction* on state that takes into account only the odometry and is then *corrected* using the feedback from perception. In order to achieve a recursive formulation the predictive term is further decomposed according to the theorem of total probability

$$
\begin{aligned}
Eq.\ (2.2) &= \eta\ p(z_t|x_t,m)\cdot \\
&\quad \int_{\mathbb{R}^{d+2M}} p(x_t,m|x_{t-1},u_{1:t},z_{1:t-1})\ p(x_{t-1},m|u_{1:t},z_{1:t-1})dx_{t-1} && (2.6) \\
&= \eta\ p(z_t|x_t,m)\cdot \\
&\quad \int_{\mathbb{R}^{d+2M}} p(x_t,m|x_{t-1},u_t)\ p(x_{t-1},m|u_{1:t-1},z_{1:t-1})dx_{t-1} && (2.7)
\end{aligned}
$$

Simplification of Eq. (2.6) is due once again to the complete state hypothesis. The resulting equation formally defines recursive Bayes filters. Starting from the posterior $p(x_{t-1},m|u_{1:t-1},z_{1:t-1})$ estimated in the previous time instant $t-1$, a Bayes filter performs a prediction step using the transition model that describes the dynamics of system. The distribution is then corrected through the *likelihood function* $p(z_t|x_t,m)$ and the posterior for step t is achieved. The previous description refers to the SLAM problem, but a straightforward extension holds also for localization problem since the focus of bayesian filters is on the estimation of the dynamical part x_t of the state. Map m is immutable in the previous description because decision on map augmentation and other important details have been omitted.

The derivation of Bayes filters illustrated above hides a marginalization of complete state distribution. In Eq. (2.6) robot pose in the previous time instant x_{t-1} is introduced without previous notice. The implicit premise lies in the complete graphical model of the problem: the whole robot trajectory is included and recursively removed from the marginal distribution that is estimated. Thus, marginalization is a unifying concept for all the approaches to localization and mapping problems.

2.1.3 Marginalization and conditioning

The most general formulation of localization and mapping problems is provided by the graphical model of full SLAM. All random variables and relationships between pairs of variables are represented. However, each specific method operates on a reduced subset of variables to estimate the solution. For example, Bayes filters illustrated in previous section recursively remove the latest robot pose and keep the updated value. Formally, the distribution obtained from a subset of joint distributions is called marginal distribution, hence the name of *marginalization*. Marginalization is achieved by integrating the variables to be removed on their domains.

Furthermore, a decomposition of joint distribution can be achieved through *conditioning*. Sometimes conditioning allows factorization of joint variables that are conditionally independent. While independence is a strong assumption, conditional independence is often a proper hypothesis based on the structure of the problem. An important case is the mutual conditional independence among map features when conditioned by robot pose: such reasonable assumption is the foundation of Rao-Blackwellized Particle Filters [59, 58]. The result of factorization is that the complete posterior is decomposed into marginal distributions and such marginals can be computed independently.

Reduction of the number of variables achieved by marginalization or decomposition allowed by factorization are required for several reasons. Several methods decompose the joint distribution because only elementary distributions have a closed form mathematical formulation. For example Bayes filters split the estimation into prediction and correction steps that are performed using elementary generative laws. Motion and sensor models are used to update the value of robot pose, or more formally to update the associated marginal posterior.

Another important reason for focusing on marginals is the limitation of SLAM problem dimension. Curse of dimensionality is a main issue in mapping problems because the map size increases with the exploration of unknown regions of the environment. State augmentation affects solution algorithms in many ways, not only their computational complexity. Approaches based on importance sampling are sensitive to state size more than parametric Bayes filters since they rely on sampling. The number of samples required for an accurate estimation depends on the size of the domain of variables and the systematic increase of the number of variables causes an explosion of complexity and hinders convergence speed to the solution. Hence, Rao-Blackwellization has been introduced to split state in two parts: the dynamical one is estimated with sequential importance sampling, while the map is treated with parametric representations (usually gaussian). An explicit example of marginalization is found in the *GraphSLAM* algorithm [60, 74]. This offline algorithm computes the full posterior that consists of the whole robot trajectory and map features. Solution is achieved by iterating the elimination of map variables and the update of estimation through numeric methods. In this case the convergence speed and the complexity of iterative numerical methods motivate marginalization.

In this thesis marginalization and conditioning have been discussed because these operations change the dependencies between problem variables. In particular they affect the connectivity of the graphical model. When the state is represented by multivariate normal random variables, the connectivity is apparent from the second order statistics, i.e. from the covariance or information matrix. On the matrix variable elimination is then equivalent to removing the rows and columns corresponding to the deleted variables and to substituting the terms associated to the remaing variables with the *Schur complement* [74, 14]. If two variables were connected to a third removed variable, this matrix operation connects the first two variables. The resulting matrix tends to be less and less sparse within variable elimination.

The connectivity of the network clearly has an impact on the computation time and on the feasibility of time constraints. When new variables are added to the graphical model, the full posterior of the modified network has to be re-estimated. An incremental method would update the previously computed posterior by limiting the modification of previous estimation. The extent of the update depends both on the network topology and on the manipulation performed by the solution algorithm. Figure 2.2 shows two different iterations of incremental estimation of an artificial map consisting of poses. The figure does not represent the graphical model, but the correspondence of the poses and connecting lines of the map to the random variables and conditioned constraints in the graphical model is straightforward. When the poses added to the network are loosely connected with the other part of the network (Figure 2.2, top), only a local modification is required. On the other hand, the update involves a large portion when a loop is closed (Figure 2.2, bottom). While no control of the topology of the belief network resulting from the exploration of the environment is possible, mapping algorithms manage the structure of reduced networks through marginalization. Recursive filtering techniques remove past robot trajectory without questioning whether the elimination of robot pose has an impact on the network.

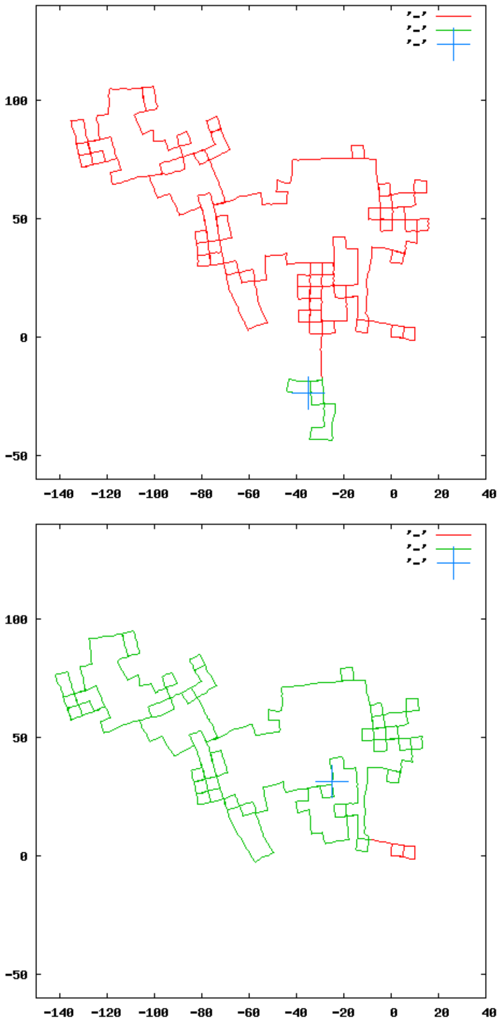


Figure 2.2: Examples of incremental map estimation when a small (top) or large portion (bottom) of the previously estimated map is recomputed. The modified part is shown in green color.

Therefore, the approaches that explicitly evaluate variable connectivity in a graphical model grant larger control on real-time feasibility.

2.2 Real-Time Feasibility

The main issue of this thesis is the adaptation of localization and mapping algorithm in order to meet time constraints. Localization and mapping estimation is performed iteratively after a new observation is available to the robot. The task should be accomplished before the next acquisition. The time interval between two sensor acquisitions gives the relative deadline for each iteration. Here, we assume that the sensor acquisition is periodic and that sensor data are delivered as a single observation regardless for the potentially different sources of perception. Furthermore, several details about other robotic tasks that may interferes with the execution are not considered. The focus is entirely on the algorithmic requirements to meet real-time constraints, although soft constraints. The preliminary question is whether there are general design principles that hold both for localization and mapping problems. Apparently, the answer is negative since the sources of complexity are different for the two problems.

Posterior estimation in localization is on a fixed size state: the aim is the evaluation of current robot pose distribution given by Eq. (2.1). Bayes filters are then suitable to solve global localization because the map of the environment is known and there is no need to grow the dimension of state space. The most effective method to address localization is Monte Carlo Localization (MCL). MCL represents the posterior using a set of samples of the robot state according to the well established technique of *importance sampling*. Such sampled representation is the reason both of its flexibility and of its complexity when compared to other parametric Bayes filters. The number of samples is a measure both of the accuracy of the representation of localization hypotheses and of the time required to perform a filter iteration. Roughly speaking a time constraint on the execution of MCL corresponds to a bound on the number of samples. Thus, real-time feasibility depends on an internal parameter of the algorithm rather than on the structure of the problem.

Conversely, the complexity of mapping algorithms lies in size and correlation of state variables. The discussion in the previous section clearly illustrates how the complexity of the mapping problem depends on the connectivity of the graphical model. Moreover, there is no control on the topology of the network and no real bound can be established on the execution time. A solving technique may keep the network as sparse as possible through careful variable elimination. Even if no real-time constraint is granted, the incremental update of the network is limited to *local* interaction when possible.

In the two cases discussed above the limitations on real-time feasibility depend on different variables: the samples for MCL and the state variables (robot trajectory and map features) for mapping. We call them complexity variables hence after. However, the solutions proposed in this thesis to meet time constraints have several similarities.

First, the complexity variables are subdivided into subsets. Each subset collects the variables that are in the proximity of the event that changes the estimation of the posterior. Such proximity may be *temporal* or *spatial*. In MCL the samples represent the robot location hypotheses on the whole map, but the event that modifies the estimation is the periodic acquisition of a measurement: the new observation is used to compute the importance weights. The addition of a new map feature (or robot pose) is the event that causes the update of the rest of the map. The perturbation should be limited to the subset of variables near the changed map changed. The results of processing in "local" subsets are then composed together to achieve the complete estimation. The complete estimation of posterior is somehow delayed. We will see when and how this could be achieved for specific algorithms.

2.3 Particle filters

This section illustrates the localization methods based on *sequential importance sampling* (SIR), also known as *Particle Filters* (PF) or *Monte Carlo Localization*. Such methods have become the most widely used approaches to localization due to their ability of approximating a large range of probability distributions, in contrast to Kalman filters. The previous discussion of real-time constraints in MCL has shown the difficulties due to importance sampling technique. In the following we present a brief account of literature on PFs for robot localization and the details of the algorithm.

2.3.1 Related works

Earlier approaches to probabilistic robot localization with on-board sensors focused on *position tracking*. The initial pose of the robot is given and the aim is to keep this information. Position tracking has been addressed with the well established Kalman filter and a beacon based map and sensor model [45]. However, gaussianity assumption of Kalman filters on state distribution does not held in global localization problems. To overcome this limitation solutions like *Multi-Hypothesis Tracking* have been proposed [2].

Concrete advancements in the solution of general localization problem have been made with the Particle Filter, a discrete Bayes filter technique relying on importance sampling. Historically, the origin of importance sampling traces back to the works of Metropolis and Hastings [57, 35], but sequential importance sampling for the estimation of dynamic systems has been introduced in early ’90s and has been extensively studied only in the last decade [16, 48, 17]. Particle filters have been then proposed to solve robot localization [26, 27]. Several aspects have been investigated in the development of grid-based localization, a predecessor of MCL: for example active localization [6], correlation-based sensor models [42], and feature-based sensor models for cameras [72, 20].

Three significant contributions to particle filter algorithm will be discussed with greater attention in this thesis. The contribution in [27] includes a thoughtful discussion of the effects of inefficient proposal distribution[2] on localizer performance and several alternative proposals are illustrated. A sampling method based on *Kull-Back Leibler divergence* (KLD) has been proposed in [25] to reduce to the minimum the number of samples used in the estimation of posterior. The KLD measures the difference between two distributions. The number of samples required to approximate the true posterior with a divergence less than ε with a confidence of at least $1-\delta$ is com-

[2] Proposal distribution will be defined formally in next subsection.

puted with a closed form formula. Finally, the *Real-Time Particle Filter* (RTPF) [44] algorithm represents the most significant effort to adapt PFs to real-time constraints. For a deeper discussion of RTPF we refer to chapter 3.

Particle filters have been also adapted to solve SLAM problem as briefly reminded in previous discussion on posterior factorization. To overcome the curse of dimensionality for SIR techniques, *Rao-Blackwellized Particle Filters* (RBPFs) have been proposed. *FastSLAM* is the first RBPF to handle features map [58, 59]. A version of RBPF for efficient occupancy grid map is presented in [33]. The work on asynchronous multirate RBPF in [1] is interesting for the discussion on real-time feasibility, since its aim is the management of sensors and actuators with different sampling rates.

2.3.2 Monte Carlo Localization

Monte Carlo Localization is a localization method based on importance sampling. Importance sampling is a Monte Carlo technique originally conceived for evaluating integrals related to the distribution of a random variable. In a typical formulation of the problem [46, 17], the aim is the estimation of the quantity $I(h) = E_p\left[h(x)\right]$, where x is a random variable, $h(\cdot)$ a function of interest and $p(x)$ is the *target distribution*. Since the probability density function $p(x)$ is unknown or difficult to manipulate in a closed form expression, $I(h)$ is computed by generating samples $x^{(i)}$ $(i = 1 \dots N)$ distributed according to a *proposal distribution* $\pi(x)$ that approximates the target distribution. Each sample $x^{(i)}$ approximates $p(x)$ with a different degree of precision and the contribution of each sample $x^{(i)}$ is measured by the *importance weight* defined as

$$w^{(i)} = w(x^{(i)}) = \frac{p(x^{(i)})}{\pi(x^{(i)})} \tag{2.8}$$

Thus, the value of $I(h)$ is then estimated by the following weighted discrete approximation

$$\hat{I}_N(h) = \frac{\sum_{i=1}^{N} h(x^{(i)}) w(x^{(i)})}{\sum_{i=1}^{N} w(x^{(i)})} \tag{2.9}$$

Note that the term at the denominator is a normalization constant.

This formulation of importance sampling method cannot be directly applied to estimate posterior Eq. (2.1) in robot localization. A robot is a dynamical system and a localizer has to update recursively the value of robot location, when new measurements are acquired. Importance sampling has to be adapted to Bayes filter framework: the target distribution is conditioned by control data and observations and the computation is decomposed in prediction and correction steps. Moreover, the purpose of robot localization is not the estimation of a specific value like the expected value $I(h)$, because the dynamic evolution of the robot is fully represented by the sampled distribution.

Sequential importance sampling has been proposed to extend the above method to sequential estimation problems. According to this formulation the target distribution at iteration t is approximated by a set of weighted samples $S_t = \{(x_t^{(i)}, w_t^{(i)})\}_{i=1}^N$ and both the distribution and the samples set evolve in time. The importance weight of each sample is updated after the evolution of the sample value. In the specific case of localization, the expression of weight is given by

$$w_t^{(i)} \propto \frac{p(x_t^{(i)}|u_{1:t}, z_{1:t}, m)}{\pi(x_t^{(i)}|u_{1:t}, z_{1:t}, m)} \, w_{t-1}^{(i)} \tag{2.10}$$

The target distribution Eq. (2.1) can be decomposed as shown in Bayes filters derivation (section 2.1.2). A natural choice for proposal $\pi(\cdot)$ is then the distribution obtained after the prediction step. Thus, importance weights are updated by the remaining term $p(z_t|x_t^{(i)})$ which represents the likelihood function. This proposal allows a remarkable simplification of expression and it is easy to compute, but it leads also to poor performance when it places too few samples in regions where the likelihood function is large. Several improvements in sampling criteria have been proposed for both localization and mapping problems [27, 63, 31]. The choice of proposal will be further discussed in chapter 3.

The computation of weights suggested by Eq. (2.10) is prone to numerical divergence. The values of importance weights tend to be smaller and smaller because a small portion of the sample set represents the most likely estimation hypotheses. Bootstrap provides a solution to the divergence of sample distribution from the correct posterior. Samples with low importance weight are removed and replaced by particles with high importance weight. The weighted distribution is replaced by a new set of samples with equal weights. Bootstrap is operatively performed by resampling.

```
Data: $S_{t-1}$: set of $N$ samples $\left\langle x_{t-1}^{(i)}, w_{t-1}^{(i)} \right\rangle$ with $i = 1 \ldots N$; $u_t$: motion
    command; $z_t$: observation.
Result: $S_t$: updated set of samples
$S_t =$;
$s = 0$;
foreach $i = 1 \ldots N$ do
    pick index $j$ from $S_{t-1}$ and corresponding sample $x_{t-1}^{(j)}$;
    sample $x_t^{(i)}$ from $p(x_t | x_{t-1}, u_t)$ using $x_{t-1}^{(j)}$ and $u_t$;
    $w_t^{(i)} = p(z_t | x_t^{(i)})$;
    $S_t = S_t \cup \{ \left\langle x_t^{(i)}, w_t^{(i)} \right\rangle \}$;
    $s = s + w_t^{(i)}$;
end
foreach $i = 1 \ldots N$ do
    $w_t^{(i)} = w_t^{(i)} / s$;
end
```

Algorithm 1: Standard Monte Carlo Localization algorithm.

Algorithm 1 shows a single iteration of the standard Particle Filter for robot localization. The input data consist of the set of samples S_{t-1} obtained after the previous iteration and of the current motion command u_t and observation z_t. The result consist of the updated set of samples S_t. The posterior distribution defined by sample set is often noted as belief. The execution is centered on the cycle that performs the update of sampled distribution. First, a new sample is drawn from the previous distribution. This operation is the resampling step and is usually better performed on the whole distribution. Several resampling techniques have been proposed and a satisfactory overview can be found in [4]. Resampling step is conceptually the last step of the algorithm to be placed after normalization, but the pseudo-code is more compact if its execution is put inside the update cycle.

The prediction of the new robot location hypotheses is represented by the sampling from motion model $p(x_t|x_{t-1},u_t)$. The motion model is usually composed by the kinematic model of the robot and an additive noise. Possible modelling inaccuracies in the motion equations are sometimes included in the noise. An outcome of noise representing the uncertainty on motion is generated as a pseudo-random number and can be added to the command u_t used to compute the new pose or directly to the computed robot pose. The predicted distribution can then be computed without linearizing motion equations. On the contrary, EKFs and parametric filters in general require the linearization of equations to update the distribution.

The correction step corresponds to the computation of importance weight $w_t^{(i)}$. The sensor model $p(z_t|x_t^{(i)})$ has a significant role in the convergence of the filter. Importance weights measure the likelihood of the hypotheses contained in the samples. As noted before, if the regions of the state domain with large likelihood are too small and too few samples fall inside them, the correct hypothesis may be dropped. This drawback can be avoided with a careful choice of proposal, but also of the sensor model. Sensor models have been proposed for range sensors [74], correlation models [42], vision landmarks [20]. An observation z_t is often composed of several measurements, e.g. a laser scan is a vector of ranges. Independence of such measurements is usually assumed to factorize the likelihood function and to compute the weight. Such assumption has a remarkable impact on likelihood, but it does not always hold and alternative sensor models have been proposed [69].

The above description of Particle Filters shows that the time required to execute the algorithm mainly depends on the number of samples. Prediction, correction and resampling are performed on each sample. Estimation of the number of samples needed for an accurate approximation of true state distribution is difficult, since it changes during the various phases of localization. At the beginning the particles should be distributed on all the map because no information on robot position is available, but after few filter iterations clusters of particles appear. Given a target density of samples, i.e. the bin size, the KLD sampling technique [25] mentioned in previous subsection allows the computation of the number of samples required to approximate the true distribution with the desired precision. However, such solution does not ensure real-time feasibility, even though it reduces the filter complexity. Observations are acquired periodically in spite of the needed samples.

2.3.3 Effective Sample Size

The *effective sample size* is an heuristic measure of the efficiency for an importance sampling scheme. In a standard estimation problem like Eq. (2.9), the efficiency depends on the variance of the estimated value, $Var_{\pi}[\hat{I}_N(h)]$. The smaller is the variance of estimation, the smaller is the number of samples needed to reach a stable value. Since a direct evaluation of variance for the resulting quantity is difficult, a "rule of thumb" approximating $Var_{\pi}[\hat{I}_N(h)]$ has been proposed in [47, 46]. Using standard delta methods for ratio statistics, the efficiency is approximatively given by

$$n_{eff} = \frac{N}{1+Var_{\pi}[w(x)]} \tag{2.11}$$

Since the expected value $E_{\pi}[w(x)] = 1$, the variance of importance weight function can be expressed using the square values of weight samples

$$\begin{aligned} n_{eff} &= \frac{N}{1+\frac{1}{N}\sum_{i=1}^{N}\left(w^{(i)2}-1\right)} && (2.12) \\ &\approx \frac{1}{\sum_{s=1}^{N}\tilde{w}^{(i)^2}} && (2.13) \end{aligned}$$

The final result Eq. (2.13) has been achieved by substituting the weights $w^{(i)}$ with the normalized weights $N\ \tilde{w}^{(i)}$. This heuristic rule can be applied also to sequential importance sampling.

The effective sample size may be interpreted as the number of samples that should be drawn from the unknown target distribution to achieve an approximation as accurate as the current one obtained by sampling from the proposal. The samples drawn from a proposal equal to the target distribution would have all unitary weights (see Eq. (2.8)), normalized weights $\tilde{w}^{(i)} = 1/N$ and null variance. Thus, the n_{eff} would be then equal to the cardinality of the whole set. In particle filter practice, the effective sample size is commonly used as a criterion for deciding when to perform resampling step. A resampling step is necessary both to avoid numerical divergence and

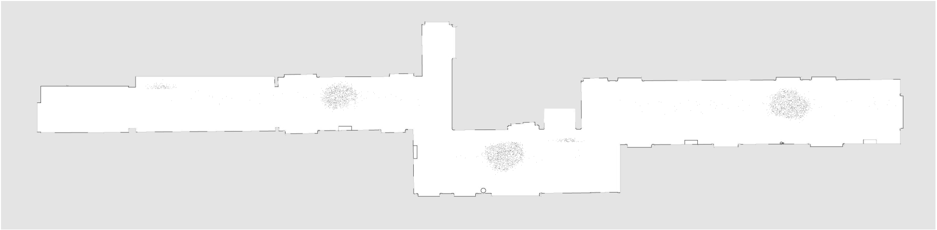


Figure 2.3: Example of sample set after a single particle filter iteration. The starting distribution was uniform on all the free space of themap.

to replace the samples with low importance weight, but it may remove good samples causing particle impoverishment [31]. Resampling is then performed when the effective sample size is less than a given threshold typically equal to $N/2$ or $3N/2$.

The effective sample size can be interpreted as a measure of the accuracy required by the likelihood function used by the particle filter. The more the likelihood function is peaked around its maxima, the less is the size of the domain regions with an high likelihood. Figure 2.3 shows an example of the first particle filter iteration for a specific problem. The proposal distribution is uniform on the free space of the map in the figure, but after the resampling step samples remain only in the regions where the importance weights are higher. The experiment can be repeated many times using a range sensor model with different numbers of beams [74]. Figure 2.4 depicts the relative effective sample size with respect to the number of beam ranges considered in the evaluation of the importance weights. The effective sample size obviously decreases with the number of beams because the likelihood function is more restrictive.

Thus, the value of effective sample size is related both to the concentration of samples in most likely domain regions due to the proposal and to the size of these regions due to the likelihood function. The geometric interpretation of this parameter will be extensively used in chapter 3 for the analysis of mixture representation in RTPF.

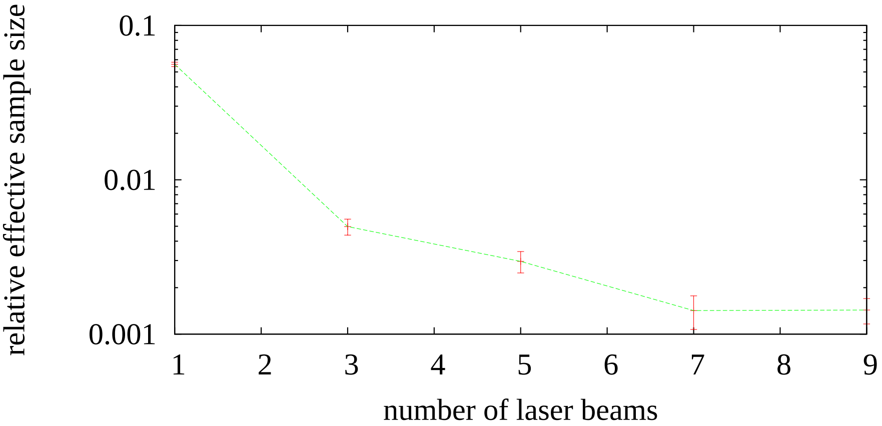


Figure 2.4: Effective sample size after resampling with respect to the number of beams considered by a range sensor model.

2.4 Maximum Likelihood Mapping

This section illustrates the principles of maximum likelihood (ML) techniques for mapping. This approach is more closely correlated to graphical model discussed in section 2.1.3 than recursive Bayes filtering. For ML methods the graphical model is not only a theoretical reference model, but a framework for every operation over the map. For this reason some of these algorithms are called *GraphSLAM* or *Graphical SLAM* [61, 23, 24, 60]. Even though they have been originally formulated for offline mapping, ML algorithms are convenient candidates for real-time and distributed execution. An important consequence of graphical formulation is the sparse structure of the problem. Such sparsity may be exploited by ML algorithms to decompose the network in the most convenient form.

After a discussion of related works, this section provides the derivation of the approach from the general probabilistic formulation. The focus is more on the two possible models for the problem, the feature-based and the delayed-state models.

2.4.1 Related Works

The first techniques proposed for ML mapping were batch algorithms for the solution of offline SLAM. Offline methods require all data to be available right from the beginning. When a robot pose or observation is added to the map, the map is computed again without exploiting previously computed estimation. Lu and Milios [55, 56] pioneered the maximum likelihood approach proposing a brute force technique to align range scans. The map consists of a collection of scans and the constraints are recovered through scan matching [54]. A variant of the algorithm with a metric-topological hybrid map and an effective loop detection based on correlation was presented in [34]. To improve the efficiency of estimation, gradient descent and conjugate gradient techniques have been applied [41, 61, 60].

Other solutions are based on Gauss-Seidel relaxation [18, 23, 24, 29]. In particular, Duckett *et al.* [18] iteratively solve the pose of each node fixing the position of other nodes and assuming that the orientation of each pose is known. Folkesson *et al.* [23, 24] propose a technique to group adjacent nodes into a star node in order to accelerate the convergence of the algorithm. Multi-Level Relaxation (MLR) [29] performs Gauss-Seidel iteration at different levels of resolution.

Dellaert *et al.* proposed different algorithms based on factorization of information matrix [15, 14, 39]. The method in [15] relies on QR factorization of linearized information matrix and on decomposition of factorization problem into small subproblems. Square Smoothing And Mapping (SAM) may use either Cholesky or QR factorization with back-substitution [14]. Efficiency can be obtained with variable reordering and variable elimination. An incremental version of SAM is illustrated in [39].

ML methods generally manipulate the exact graphical model, but some techniques prune the graphical model and force a tree structure. Thin Junction Tree Filters (TJTF) [68] and Treemap [28] both ignore the weak correlations between distant locations.

Olson *et al.* [66] proposed to apply *Stochastic Gradient Descent* (SGD) for the estimation of the map. SGD selects one constraint at a time and solves it independently from other constraints. SGD is particularly efficient when the incremental parameterization is used. A tree parameterization that speeds up convergence has been proposed in [32]. The incremental versions of this technique [67, 30] will be further described in chapter 4.

Frameworks like ATLAS [5] are hybrid solutions that combine linearized submaps estimated with EKF in a unique submap. Since ML methods are less prone to linearization error, they can be used to adjust local map and handle large environments.

The evolution of ML briefly depicted above shows the progressive transition from offline to incremental algorithms. First, researchers directed their effort to the reduction of the computational complexity of batch estimation. Efficiency has been reached by decomposing the algorithm in either a hierarchical or a parallel way. Several advanced offline methods are so fast that can be used in online estimation when the size of the map is not too large. Finally, incremental versions have been proposed to update the map estimation without recomputing the whole map.

2.4.2 Probabilistic Derivation

The probabilistic interpretation of Maximum Likelihood methods is less apparent than Bayes filters. Sometimes these techniques have been presented with more emphasis on the nonlinear optimization problem that has to be solved in order to estimate the map. The function to be optimized is a quadratic function that represents the weighted error of map configuration with respect to given constraints. The probabilistic origin of the approach lies in the interpretation of the error function that has to be minimized and the "weighting" terms directly related to the second order statistics.

In the following, the derivation of the optimization problem from the estimation of full SLAM distribution Eq. (2.3) is briefly described according to [74]. The full posterior can be decomposed with Bayes formula as shown for recursive Bayesian filters Eq. (2.5)

$$
\begin{aligned}
p(x_{0:t}, m | u_{1:t}, z_{1:t}) &= \eta \; p(z_t | x_{0:t}, m, u_{1:t}, z_{1:t-1}) \; p(x_{0:t}, m | u_{1:t}, z_{1:t-1}) && (2.14) \\
&= \eta \; p(z_t | x_t, m) \; p(x_{0:t}, m | u_{1:t}, z_{1:t-1}) && (2.15)
\end{aligned}
$$

As usual, the irrelevant conditioning variables have been removed. The only difference is that in this case the variables representing the previous robot trajectory $x_{0:t-1}$

are removed. The contribution of robot motion u_t can be pointed out simply by conditioning current robot pose x_t

$$\begin{aligned} p(x_{0:t}, m | u_{1:t}, z_{1:t-1}) &= p(x_t | x_{0:t-1}, m, u_{1:t}, z_{1:t-1}) p(x_{0:t-1}, m | u_{1:t-1}, z_{1:t-1}) \quad (2.16) \\ &= p(x_t | x_{t-1}, u_t) p(x_{0:t-1}, m | u_{1:t-1}, z_{1:t-1}) \quad (2.17) \end{aligned}$$

Since the estimation concerns full SLAM, no marginal has to be computed and the above equation does not contain integrals. When the two steps described above are applied recursively, an *exact* factorization of full SLAM is achieved

$$p(x_{0:t}, m | u_{1:t}, z_{1:t}) = \eta \ p(x_0, m) \prod_{t=1}^{T} p(z_t | x_t, m) \ p(x_t | x_{t-1}, u_t) \quad (2.18)$$

An important consequence of the above factorization is that two non-consecutive robot poses are not correlated. Since map features m are locally related to the poses where they have been observed, the graphical model associated to the problem is naturally *sparse*.

If a different map representation or sensor model are exploited, the structure of the graphical model may not be as illustrated above. In particular, when sensor returns an observation related to the view of the robot in a given pose instead of a set of features, map variables of vector m are not explicitly represented. A view based approach is typically used with laser scans [55], but also with a vision based system [21]. In the case of laser scans, odometry information u_t and observations $z_{t-1:t}$ may be implicitly used to compute the distribution of consecutive poses $x_{t-1:t}$ with scan matching [54, 11]. Formally, the *delayed-state formulation* of mapping problem is obtained from the full SLAM formulation by computing the marginal distribution of $x_{0:t}$ [22]. For example, when a map feature m_k is observed by robot both in pose x_i (observation z_i) and in x_j (observation z_j), the integration over m_k transforms the terms of factorized posterior Eq. (2.18) related to m_k as follows

$$\begin{aligned}
[\mathit{integr.\ Eq.}\ (2.18)] &= \int_{\mathbb{R}^d} p(z_i|x_i,m_k)\ p(z_j,x_i|m_k)\ p(x_0,m_k)\ dm_k && (2.19)\\
&= \int_{\mathbb{R}^d} \frac{p(z_i,x_i|m_k)}{p(x_i|m_k)}\ \frac{p(z_j,x_j|m_k)}{p(x_j|m_k)}\ p(x_0,m_k)\ dm_k && (2.20)\\
&= \int_{\mathbb{R}^d} \frac{p(z_i,z_j,x_i,x_j)|m_k)}{p(x_i,x_j|m_k)}\ p(x_0,m_k)\ dm_k && (2.21)\\
&= p(z_i,z_j|x_i,x_j)\ p(x_0) && (2.22)\\
&= p(z_i|x_i,x_j)\ p(z_j|x_i,x_j)\ p(x_0) && (2.23)
\end{aligned}$$

In the original posterior poses x_i and x_j were only conditionally independent (see Eq. (2.20)) and after marginalization z_i and z_j are conditioned both by the two poses. Since a map feature can be observed only when the robot moves in the neighborhood of the feature, a robot pose is correlated with near poses even after marginalization. Thus, the global network remains sparse also for the delayed-state formulation.

Each observation may be viewed as a constraint between a pair poses. In the following c_{ji} will be used for the constraint $\langle j,i \rangle$ computed by combining two observations z_i and z_j, sometimes using also the odometric information. As observed in [65], in this context a constraint is not an inviolable condition that must be satisfied by the solution. Such condition may be violated, but each violation has a cost that is proportional to the magnitude of the violation. The form of constraints depends on the provided sensor information. Bearing only or range models are among the most commonly used ones. In this thesis, a constraint c_{ji} represents the observation of pose j from node i, i.e. the relative transformation between the two reference frames. The value of relative transformation from i to j can also be computed as a function of the current configuration of the network $\mathbf{x}$, $f_{ji}(\mathbf{x})$. Map estimation should reduce the conflict between $f_{ji}(\mathbf{x})$ and constraint c_{ji}.

Notation for state variables $\mathbf{x}$ has been slightly changed both to simplify formalism and to point out that in delayed-state formulation poses vector $\mathbf{x}$ does not always represent robot trajectory. In particular some methods exploit a different parameterization for robot poses like the incremental parameterization in [66]. Furthermore, poses in $\mathbf{x}$ may refer to the reference frames of local maps extracted from several observations acquired in the area [34, 18, 51].

A solution for the problem formulated above requires the choice of the distribution of map variables. A common assumption is that constraints $\langle j,i \rangle$ are distributed according to multivariate normal $c_{ji} \propto \mathscr{N}(\delta_{ji}, \Omega_{ji}^{-1})$, where δ_{ji} is the expected value of constraint and Ω_{ji} is its information matrix. The constraint set is noted with the symbol $\mathscr{C}$. The map configuration $\mathbf{x}^*$ that maximizes the likelihood

$$\mathbf{x}^* = \operatorname*{argmax}_{\mathbf{x}} p(\mathbf{x}|c_{ji} \in \mathscr{C}) \tag{2.24}$$

is the solution to the estimation problem. The distribution in Eq. (2.24) is the marginal of Eq. (2.18). In the equation the distribution is in a compact form, but it can be factorized as shown by Eq. (2.23)

$$p(\mathbf{x}|c_{ji} \in \mathscr{C}) \quad = \quad p(x_0) \prod_{\langle j,i \rangle \in \mathscr{C}} p(c_{ji}|\mathbf{x}) \tag{2.25}$$

The conditioning terms $z_{1:t}$ and $u_{1:t}$ of posterior have been substituted by the derived constraints.

The estimation problem Eq. (2.24) is usually simplified by applying logarithm to remove the exponential function of multivariate distribution. Furthermore, the factorized posterior is transformed into a sum of quadratic terms representing the Mahalanobis distances between each constraint and the map configuration. In the equivalent formulation of the problem the aim is the minimization of *negative log-posterior*

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \sum_{\langle j,i \rangle \in \mathscr{C}} r_{ji}(\mathbf{x})^T \Omega_{ji} r_{ji}(\mathbf{x}). \tag{2.26}$$

where $r_{ji}(\mathbf{x}) = \delta_{ji} - f_{ji}(\mathbf{x})$ are the residuals. Negative log-likelihood can also be interpreted as an error function. The methods listed in previous subsection differ in the techniques used to solve the nonlinear optimization problem given by Eq. (2.26).

Chapter 3

Real-Time Particle Filter for Robot Localization

In this chapter the Real-Time Particle Filter (RTPF) for robot localization is introduced. The method is a variant of standard Monte Carlo Localization (MCL) designed to achieve a trade-off between time constraints related to sensor management and filter performance.

3.1 Motivation and Problem Formulation

In chapter 2 Monte Carlo Localization (MCL) has been presented. Such method inherits advantages of *sequential importance sampling with resampling* (SIR) techniques: it allows flexibility in representation of the posterior, which usually does not have a given parametric model, and limits linearization errors in motion and sensor model equations, which often lead to poor performance and divergence of filter.

Unfortunately, particle filter (PF) complexity and performance both depend on the number of samples: in global localization a high density of samples helps to discover and to converge towards the correct localization hypothesis. However, for each additional sample a prediction, a correction and a resampling step are performed. Furthermore, localization performance also depends on sensor information, which could be acquired at a rate higher than the filter update rate. Possible solutions to this mismatch between sensing rate and processing time include reduction of the number of samples, e.g. adapting the size of the mixture [25], or of the number of observations, i.e. discarding sensor data.

The *Real-Time Particle Filter* (RTPF) [25, 44] provides a tradeoff between time constraints related to sensor management and filter performance. The tradeoff is achieved applying a particular form of the locality criterion. Samples are partitioned into subsets, each of them corresponding to an observation that is used to compute the weights of the samples of the partition. Samples are then decomposed in small portions *temporally local* to sensor data. The size of each partitioned subset is chosen so that a particle filter iteration can be performed before a new observation is acquired.

The difference with standard PF with smaller sample set lies in the representation of the posterior as a mixture of samples obtained during resampling step. Indeed, the partition sets are grouped in estimation windows of k consecutive sets. At the end of an estimation window the new sets of samples are obtained by resampling from each of the k subsets of the window. The decomposition of distribution in local aggregations is then composed into a new mixture distribution.

Critical parameters of RTPF are the mixture weights. Mixture weights determine how each partition set contributes to the posterior and are computed in order to minimize the approximation error of the mixture distribution. However, the original proposal for computation of mixture weights, based on minimization of Kullback-Leibler (KL) divergence [25, 44], is prone to bias problems and numerical instability arising from the need to perform a numerical gradient descent.

In this chapter, we provide two main contributions: a formal analysis for the evolution of mixture of posterior in RTPF and a novel solution for the computation of mixture weights [49, 52]. Each partition set posterior consists of samples, which are drawn from motion model as proposal on the estimation windows and whose importance weight depends only on a single observation. Since the correction step is performed at different time instants for each partition set, differences among partition posteriors introduce a bias in estimation. We show that the bias is an outcome of prominence of partition set that minimizes KL-divergence and has poor *effective sam-*

ple size simultaneously. We then present an improved approach for the computation of mixture weights based on *effective sample size* of the partition sets.

3.2 Real-Time Particle Filters

In particle filters, updating the particles used to represent the probability density function (potentially a large number) usually requires a time which is a multiple of the cycle of sensor information arrival. Naive approaches, yet often adopted, include discarding observations arriving during the update of the sample set, aggregating multiple observations into a single one, and halting the generation of new samples upon a new observation arrival [44]. These approaches can affect filter convergence, as either they loose valuable sensor information, or they result in inefficient choices in algorithm parameters.

An advanced approach dealing with such situations is the Real-Time Particle Filters (RTPF) [25, 44], which will be briefly described in the following. Consider k observations. The key idea of the Real-Time Particle Filter is to distribute the samples in sets, each one associated with one of the k observations. The distribution representing the system state within an estimation window will be defined as a *mixture* of the k sample sets as shown in Figure 3.1. At the end of each estimation window, the weights of the mixture belief are determined by RTPF based on the associated observations in order to minimize the approximation error relative to the optimal filter process. The *optimal belief* could be obtained with enough computational resources by computing the whole set of samples for each observation. Formally:

$$Bel_{opt}(x_{t_k}) \propto \int \dots \int \prod_{i=1}^{k} \; p(z_{t_i}|x_{t_i}) \cdot p(x_{t_i}|x_{t_{i-1}}, u_{t_{i-1}}) \cdot Bel(x_{t_0}) dx_{t_0} \cdots dx_{t_{k-1}} \quad (3.1)$$

where $Bel(x_{t_0})$ is the belief generated in the previous estimation window, and z_{t_i}, u_{t_i}, x_{t_i} are, respectively, the observation, the control information, and the state for the $i-th$ interval.

Within the RTPF framework, the *belief* for the $i-th$ set can be expressed, simi-

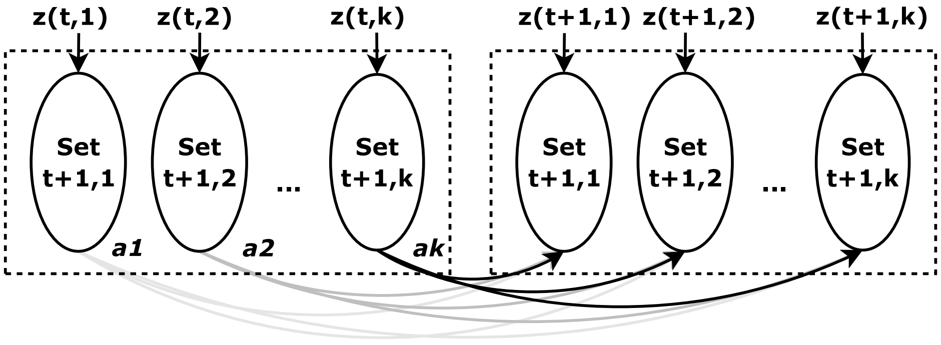


Figure 3.1: RTPF operation: samples are distributed in sets, associated with the observations. The distribution is a mixture of the sample sets based on weights α_i (labeled a_i in figure).

larly, as:

$$Bel_i(x_{t_k}) \propto \int \dots \int p(z_{t_i}|x_{t_i}) \cdot \prod_{j=1}^{k} p(x_{t_j}|x_{t_{j-1}}, u_{t_{j-1}}) \cdot Bel(x_{t_0}) dx_{t_0} \dots dx_{t_{k-1}} \tag{3.2}$$

containing only observation-free trajectories, since the only feedback is based on the observation z_{t_i}, sensor data available at time t_i. The weighted sum of the k believes belonging to an estimation window results in an approximation of the optimal belief:

$$Bel_{mix}(x_{t_k}|\alpha) \propto \sum_{i=1}^{k} \alpha_i Bel_i(x_{t_k}) \tag{3.3}$$

An open problem is how to define the optimal mixture weights minimizing the difference between the $Bel_{opt}(x_{t_k})$ and $Bel_{mix}(x_{t_k}|\alpha)$. In [44], the authors propose to minimize their Kullback-Leibler distance (KLD). This measure of the difference between probability distributions is largely used in information theory [13] and can be expressed as:

$$J(\alpha) = \int Bel_{mix}(x_{t_k}|\alpha) \log \frac{Bel_{mix}(x_{t_k}|\alpha)}{Bel_{opt}(x_{t_k})} dx_{t_k} \tag{3.4}$$

To optimize the weights of mixture approximation, a gradient descent method is proposed in [44]. Since gradient computation is not possible without knowing the optimal belief, which requires the integration of all observations, the gradient is obtained by Monte Carlo approximation: believes Bel_i share the same trajectories over the estimation windows, so we can use the weights to evaluate both Bel_i (each weight corresponds to an observation) and Bel_{opt} (the weight of a trajectory is the product of the weights associated to this trajectory in each partition). Hence, the gradient is given by the following formula:

$$\frac{\partial J}{\partial \alpha_i} \simeq 1 + Bel_i \log \frac{\sum_{j=1}^{k} \alpha_j Bel_j}{Bel_{opt}} \tag{3.5}$$

$$\simeq 1 + \sum_{s=1}^{N_p} w_{t_i}(x_{t_i}^{(s)}) \frac{\sum_{j=1}^{k} \alpha_j \, w_{t_j}(x_{t_j}^{(s)})}{\prod_{j=1}^{k} w_{t_j}(x_{t_j}^{(s)})} \tag{3.6}$$

where Bel_i is substituted by the sum of the weights of partition set $i - th$ and Bel_{opt} by the sum of the weights of each trajectory.

Unfortunately, the mixture posterior computed with Eq. (3.6) suffers from a *bias problem*. The mixture weights computed using the approximate gradient of KL-divergence tend to increase over the estimation window as shown in Figure 3.3. However, the sample sets tend to be more and more spread. Figure 3.2 shows two partition sets, one at the beginning of the window (top) and the other at the end (bottom). The latter set has less samples that fall in a region with significant likelihood, i.e. there are less samples in the correct robot location. After the resampling step, the distribution obtained with Eq. (3.6) contains a portion of localization hypotheses loosely related with the correct robot pose. These samples causes the bias in the estimation.

The effect of bias is even more dramatic when the posterior is a multi-modal distribution. In this scenario samples are clustered in the proximity of local maxima of the likelihood function. If an observation discerning the correct localization hypothesis is acquired, the particle filter should converge to the correct estimation. However, if the partition set representing the correct localization hypothesis contains a bias, then the choice of the correct localization hypothesis is more difficult. The solution proposed in [44] to mitigate the effect of bias is the clustering of samples. The contribution of each cluster to the gradient Eq. (3.6) is computed separately. Clustering does not remove the bias, but it limits the ambiguity. In the next section, an alternative

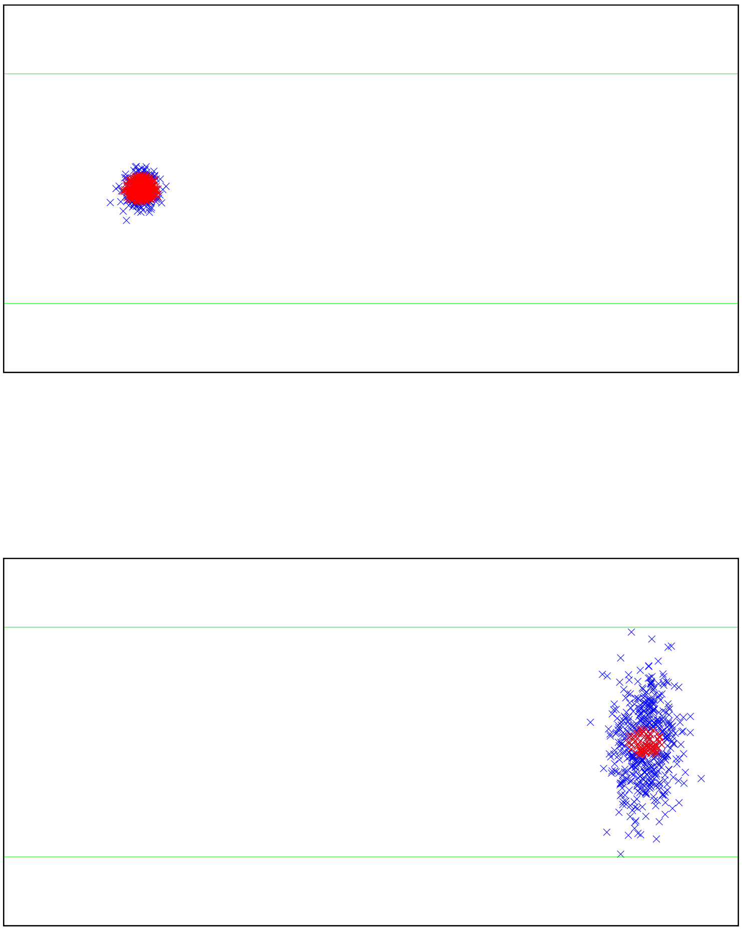


Figure 3.2: Examples of RTPF partition sets in a simulated environment at the beginning (partition set 4, top) or at the end (partition set 15, bottom) of the estimation window ($k = 15$). Samples with larger weights are colored with red.

solution is proposed.

3.2.1 Algorithm Overview

Algorithm 2 illustrates a single iteration of RTPF for the computation of partition set S_{t_l}. The normal sample update (cycle iterating on index variable m) consisting of resampling, prediction and correction steps, is included inside the cycle that collects a contribution from each partition of the previous estimation window. Mixture weights α_i are used to compute the number of samples n_i drawn from each set.

In the initial version of RTPF [43] a different resampling schema is applied by the so called Monte Carlo gradient descent. In particular, the first partition set of the estimation window S_{t_1} is obtained by drawing samples from the partitions S_{t-1_i} $(i = 1 \dots k)$ of the previous window. On the other hand, sets S_{t_l} $(l > 1)$ consist of the samples of S_{t_1} updated with the proper motion commands. The intuitive justification for Monte Carlo gradient descent is that samples are used to represent the whole robot trajectory on the estimation window. This solution is computationally efficient, but it stesses the bias problem. The sample set is the same for all partitions of the estimation window, but more spread for the last sets. Thus, accumulation of samples that are less representative of the localization hypotheses is faster.

3.3 An Enhanced RTPF

In this section we provide a formal investigation on the motivation of bias in RTPF estimation in [44], and we propose a new solution for mixture weights computation.

3.3.1 Bias in RTPF Mixture

In RTPF, samples belonging to different partition sets are drawn from the same proposal, but their importance weights depend on different observation likelihood functions $p(z_{t_i}|x_{t_i})$, which are computed in different time instants t_i. Hence, the first source of disparity among partition sets is the degree of proposal dispersion during the correction step. A suitable measure of proposal dispersion at iteration t_i is provided by

```
Data: $k$: partition number; $l$: current partition; $N_p$: partition size;
      $S_{(t-1)_1}, \cdots, S_{(t-1)_k}$: set of samples; $\alpha^{t-1}$: mixture weights;
      $u_{(t-1)_1}, \cdots, u_{(t-1)_k}, u_{t_1}, \cdots, u_{t_{j-1}}$: motion commands; $z_{t_l}$: observation
Result: $S_{t_l}$: updated set of samples for $t_l$
$S_{t_l} = \emptyset$;
$s = 0$;
foreach $i = 1, \cdots, k$ do
    $n_i = \alpha_{(t-1)_i} N_p$
    foreach $m = 1, \cdots, n_i$ do
        sample index $j$ from distribution of $S_{(t-1)_i}$;
        sample $x_{t_l}^{(m)}$ from $p(x_{t_l} | x_{(t-1)_i}, u_{(t-1)_i}, \cdots, u_{t_{l-1}})$ given $x_{(t-1)_i}^{(j)}$ and
        motion commands on intervals from $(t-1)_i$ and $t_{l-1}$;
        $w_{t_l}^{(m)} = p(z_{t_l} | x_{t_l}^{(m)})$;
        $s = s + w_{t_l}^{(m)}$;
        $S_{t_l} = S_{t_l} \cup \{ < x_{t_l}^{(m)}, w_{t_l}^{(m)} > \}$;
    end
    foreach $m = 1 \ldots N_p$ do
        $w_t^{(i)} = w_t^{(i)} / s$;
    end
    if $l == 0$ then
        compute mixture weights $\alpha_{t_1}, \cdots, \alpha_{t_k}$;
    end
end
```

Algorithm 2: An iteration of Real-Time Particle Filter.

the radius of the ball set $B(\eta_{x_{t_i}}, r) \subseteq \mathbb{R}^d$, which is centered on expected value $\eta_{x_{t_i}}$ and includes a consistent portion of the distribution of x_{t_i}. The probability that a sample falls in $B(\eta_{x_{t_i}}, r)$ can be bound by r and the trace of the covariance matrix $\Sigma_{x_{t_i}}$, since the following Chebychev-like inequality holds:

$$P\left(x_{t_i} \in B(\eta_{x_{t_i}}, r)\right) > 1 - \frac{tr(\Sigma_{x_{t_i}})}{r^2} \tag{3.7}$$

In the following, the probability of event given by $B(\eta_{x_{t_i}}, r)$ will refer to a proposal density function arrested in t_i:

$$\pi(x_{t_i}) = \int_{\mathbb{R}^{d\times i}} \prod_{j=1}^{i} p(x_{t_i}|x_{t_{i-1}}, u_{t_{i-1}})\, dx_{t_0} \ldots dx_{t_{i-1}} \tag{3.8}$$

Then, given $0 < \varepsilon < 1$, a sample falls in a ball with at least probability ε when its radius is larger than the *dispersion radius*:

$$r_{t_i,\varepsilon} = \sqrt{tr(\Sigma_{x_{t_i}})/(1-\varepsilon)} \tag{3.9}$$

Parameter $r_{t_i,\varepsilon}$ provides a rough estimation for dispersion because only for unimodal PDF the ball $B(\eta_{x_{t_i}}, r_{t_i,\varepsilon})$ (briefly B hereafter) limits a region around a local maximum. Furthermore, it is often the case that x_{t_i} is a vector of heterogeneous random variables (e.g. cartesian coordinates and angular values), whose variances are mixed in the trace, with the result that bound Eq. (3.9) largely overestimates the region. However, the dispersion radius is a synthetic value and can be adapted to multimodal distributions after decomposition into a sum of unimodal hypotheses. Empirically, this decomposition is achieved by clustering on samples.

By applying command control and updating robot position, the dispersion radius increases together with the trace of covariance matrix. If G_{t_i} is the Jacobian of motion model computed in $(\eta_{x_{t_i}}, u_{t_i})$, with $G_{t_i}\, G_{t_i}^T \geq 0$ and $tr(G_{t_i}\, G_{t_i}^T) \geq 1$ (hypotheses verified by a standard model like [74]), and $\Sigma_{w_{t_i}}$ is the covariance matrix of additive noise, then

$$tr(\Sigma_{x_{t_{i+1}}}) \approx tr(G_{t_i}\, \Sigma_{x_{t_i}}\, G_{t_i}^T) + tr(\Sigma_{w_{t_i}}) \tag{3.10}$$

Thus, we conclude that $tr(\Sigma_{x_{t_i}}) \leq tr(\Sigma_{x_{t_{i+1}}})$ and that the dispersion radius increases over the estimation window. A more accurate estimation of how it increases could be obtained with further hypotheses on the motion model, e.g. Lipschitz continuity.

Since the proposal is more and more spread in the estimation window and correction is performed at different times for each partition, we want to investigate how the dispersion affects importance weights. Observation likelihood $w_{t_i}(x) = p(z_{t_i}|x)$ is usually more concentrated than the proposal, sometimes peaked as shown in [31]. We assume that, given a proper $\delta > 0$, region

$$L = \{x \in B \mid w_{t_i}(x) > \delta\} \tag{3.11}$$

covers a consistent portion of $w_{t_i}(x)$. Thus, observation likelihood is bound in L by $M = \sup_{x \in L} w_{t_i}(x) < \infty$ (envelope condition) and in $B \setminus L$ by δ. Hence, $w_{t_i}(x) \leq \lambda(x)$ over B, with

$$\lambda(x) = \begin{cases} M & x \in L \\ \delta & else \end{cases} \tag{3.12}$$

The bounding function $\lambda(x)$ and set L are defined on ball B, and in the following we will restrict the sampling domain to B using $\pi(x_{t_i}|x_{t_i} \in B)$ as proposal. This assumption allows us to consider the dispersion radius in the following discussion. Moreover, this approximation is not so rough when ε is close to 1.

The *effective sample size* [46] is a measure of the efficiency of a set of samples in the representation of a target posterior:

$$\begin{aligned} n_{eff_{t_i}} &= \frac{1}{\sum_{s=1}^{N} \tilde{w}_{t_i}^2(x_{t_i}^{(s)})} & (3.13) \\ &= \frac{\left(\sum_{s=1}^{N} w_{t_i}(x_{t_i}^{(s)})\right)^2}{\sum_{s=1}^{N} w_{t_i}^2(x_{t_i}^{(s)})} & (3.14) \end{aligned}$$

The above expression is achieved by substituting normalized weights $\tilde{w}_{t_i}(x)$ with their expression. Maximizing the effective sample size is equivalent to minimizing the variance of the weights: it is easy to show with Jensen inequality that n_{eff} is bounded by the number of samples N, which is obtained when each weight is equal to 1 and the variance is small. Bounds on observation likelihood allow an approximation of expected values of weight and square weight:

$$\begin{aligned} E_\pi\left[w_{t_i}(x_{t_i})|x_{t_i} \in B\right] &\leq M\,H_L + \delta\,H_{B\setminus L} & (3.15) \\ E_\pi\left[w_{t_i}^2(x_{t_i})|x_{t_i} \in B\right] &\leq M^2\,H_L + \delta^2\,H_{B\setminus L} & (3.16) \end{aligned}$$

where $H_L = E_\pi[I_L(x)]$ and $H_{B\setminus L} = E_\pi[I_{B\setminus L}(x)]$ are the *visit histograms* of bins L and $B \setminus L$ respectively; in our notation $I_D(x)$ is the indicator variable with value 1 when x falls in D, zero otherwise. Equations Eq. (3.15) and Eq. (3.16) can be used to approximate numerator and denominator of Eq. (3.14):

$$n_{eff_{t_i}} \approx N \frac{\left(M\, H_L + \delta\, H_{B\setminus L}\right)^2}{M^2\, H_L + \delta^2\, H_{B\setminus L}} \qquad (3.17)$$

$$\approx N \left(H_{B\setminus L} + 2\, \frac{M}{\delta} H_L + \frac{M^2\, H_L^2}{\delta^2\, H_{B\setminus L}}\right) \qquad (3.18)$$

The approximation given by Eq. (3.18) follows from the assumption that $H_L/H_{B\setminus L} << (\delta/M)^2$. When dispersion is large, proposal can be considered almost constant on region L and its visit histogram H_L decreases proportionally with the ratio of hypervolumes of L and $B \setminus L$: $H_L \propto 1/r^d_{t_i,\varepsilon}$ in d-dimensional space. Thus, the last partition sets in the estimation window, i.e. those approximating better the distribution at the end of the estimation window, have a spread proposal and are represented by few effective samples, as shown by the trend of Eq. (3.18). From difference between effective sample size and KLD reduction, the bias in estimation follows.

The solution proposed in [44] mitigates the effects of bias by considering the multimodal structure of samples distribution in KL-distance gradient estimation. The estimation of gradient given by Eq. (3.6) ignores samples dispersion in different *bins*. Formally, gradient Eq. (3.6) is the result of underestimation of KL-divergence: call $Bel_{mix}(C_j)$ and $Bel_{opt}(C_j)$ the mixture and optimal histograms for cluster C_j respectively; from the convexity of KLD [13], Jensen inequality holds

$$KL(\sum_{j=1}^{M} Bel_{mix}(C_j) \parallel \sum_{j=1}^{M} Bel_{opt}(C_j)) \leq \sum_{j=1}^{M} KL(Bel_{mix}(C_j) \parallel Bel_{opt}(C_j)) \qquad (3.19)$$

Gradient estimation based on the second term of inequality Eq. (3.19) is better than the previous one based on the first term, but no optimality can be claimed since bin subdivision is empirical and gradient descent approaches easily incur in local minima problems. Furthermore, even if cluster detection is usually performed in PF to group localization hypotheses and no additional computational load is required, sample management is not at all straightforward.

3.3.2 Alternative computation of Mixture Weights

This section proposes an alternative criterion to compute the values of the weights for the mixture belief. Instead of trying to reduce the Kullback-Leibler divergence, our approach considers mixture weights as the assigned measure of relative importance of partitions that is transformed by processing at the end of estimation window. RTPF prior distribution is the result of two main steps: resampling of samples and propagation of trajectories along previous estimation window. The effect of resampling is the concentration of previous estimation window samples in a single distribution carrying information from each observation. Conversely, the trajectories update given by odometry and observation spreads the particles on partition sets.

Our attempt is to build a linear map modeling the change of relative importance, i.e. mixture weights α, due to resampling and propagation of samples. This map should depend on sample weights. Let w_{ij} be the weight of the $i-th$ sample (or trajectory) of the $j-th$ partition set. Then, the *weight partition matrix* is given by

$$W = \begin{bmatrix} w_{11} & \dots & w_{1k} \\ \dots & & \dots \\ w_{N_p1} & \dots & w_{N_pk} \end{bmatrix} \tag{3.20}$$

The weights on a row of this matrix trace the history of a trajectory on the estimation window; a group of values along a column depicts a partition handling sensor data in a given time. Resampling and trajectory propagation steps can be shaped using matrix W and mixture weights α.

- *Resampling*. The effect of resampling is the concentration of each trajectory in a single sample whose weight is the weighted mean of the weights of the trajectory. In formula, the vector of trajectory weights is given by $t = W \cdot \alpha$.

- *Propagation*. Projecting a sample along a trajectory is equivalent to the computation of the weight of the sample (i.e., the posterior) for each set, given the proper sensor information. Again, matrix W gives an estimation of the weight. Trajectories projection can thus be done with a simple matrix product

$$\hat{\alpha} = W^T \cdot t = W^T \, W \cdot \alpha \tag{3.21}$$

> Vector $\hat{\alpha}$ is a measure of the relative amount of importance of each partition set after resampling and propagation depending on the choice of coefficient α. Hence, $\hat{\alpha}$ is the new coefficient vector for the new mixture of believes.

Some remarks can be made about the matrix $V = W^T\ W$ in Eq. (3.21). First, since we assume $w_{ij} > 0$, V is a symmetric and positive semi-definite (SPSD) matrix. Moreover, each element j on the main diagonal is the inverse of the effective sample size of set j. The effective sample size is a measure of the efficiency of importance sampling on each of the partition sets. Therefore, the off-diagonal elements of V correspond to a sort of importance covariances among two partition sets. Thus we will refer to this matrix as *weights matrix*.

Hence, a criterion to compute the mixture weights consists of choosing the vector α that is left unchanged by map Eq. (3.21) except for scale. Since Eq. (3.21) depends on square of sample weights, the resulting mixture weights reflect the importance of each partition set according to the effective sample size. The vector is thus obtained by searching for an eigenvector of matrix V. To achieve better stability we choose the eigenvector corresponding to the largest eigenvalue. The eigenvector can be computed using the power method or the inverse power method. This criterion can be interpreted as an effort to balance the effective number of samples keeping the proportion among different partition sets.

Figure 3.3 illustrates the differences in mixture weights computed according to the original algorithm (RTPF-Grad) and the proposed variant (RTPF-Eig) with an example. When RTPT-Eig is used to compute mixture weights, the weights of the last partition sets in the estimation window decrease with the effective sample size of the sets, while they increase with RTPF-Grad. Thus, the proposed criterion takes into account the effectiveness of representation provided by partition sets.

3.4 Delayed Estimation in RTPF

The presentation of RTPF has been focused on the problem and on algorithmic issues of the method. To ensure real-time feasibility, RTPF exploits a complex resampling scheme that generates a mixture distribution. Since the number of samples used in

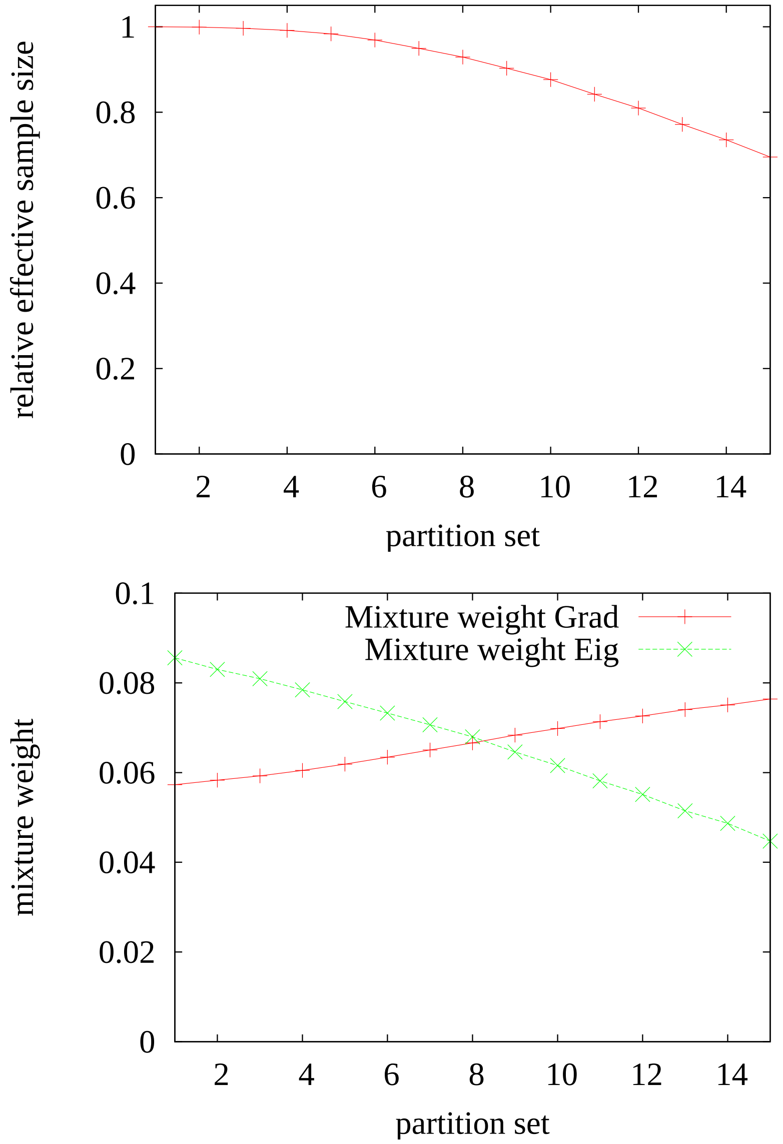


Figure 3.3: Effective sample size (high) and mixture weights computed according to the original algorithm and to the proposed variant (low) in an estimation window of 15 partitions.

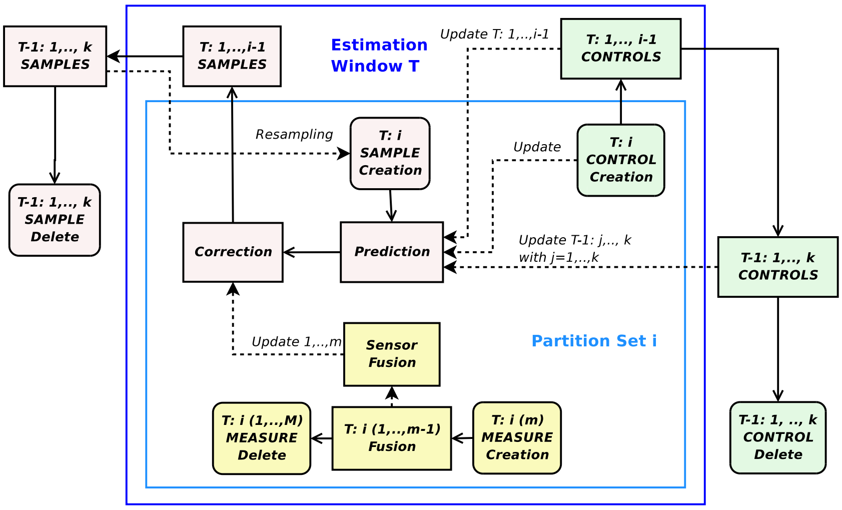


Figure 3.4: Life cycle for particle, measurement, and control variables within a single step in a real-time particle filter.

the estimation window is almost equal to the number of samples used by a standard particle on the same period, the complexity of RTPF remains unchanged except for the overhead due to the mixture weights computation and generation of samples.

However, there is a small price to be paid in term of estimation delay and intermediate storage of problem variables. A first remark concerns the mixture of posteriors Bel_{mix} that is never explicitly computed in the concrete implementation of RTPF as illustrated in Algorithm 2. The complete mixture could be obtained only at the end of the estimation window. Furthermore, since the samples of each partition set S_{t_l} represent the distribution of robot location at time t_l ($l = 1 \dots k$), samples of S_{t_l} should be updated by applying motion commands $u_{t_l:t_k}$ in order to estimate the complete mixture at time t_k. Each partition set represents the whole distribution at time t_l and there is no need to explicitly build the complete mixture. However, this estimation delay makes difficult to adapt the number of samples to the desired accuracy with the rule in [25]. The adaptation criterion suggested in [44] is too empirical to be easily applied to different contexts.

Local decomposition into partition sets and delayed resampling at the end of estimation window affect also the lifecycle and the storage policy of the variables of the problem. Figure 3.4 shows the lifecycle of samples, sensor measurements and motion controls. Note that measurements are used "locally" and are discarded immediately after the estimation of importance weights of the related samples. On the other hand, samples and motion commands are stored and used after the end of the estimation window. While this is not very expensive in term of computation resources, a management policy and proper data structures for storage are required in a flexible implementation. From the perspective of computer science, it is interesting to observe that algorithmic solutions relying on dynamical or delayed computation often require storage for intermediate results (for example, compare with the case of dynamic programming [12]).

3.5 Results

We report RTPF performance evaluation both in simulated environments and using experimental data collected by navigating a robot in a known environment. These results have been obtained exploiting the localization system described in [53]. Tests have compared the effectiveness of the two solutions previously described for computation of RTPF mixture weights by assessing their impact on localization performance.

3.5.1 Simulation

Several tests were performed in the environments shown in figures 3.5 and 3.6. They correspond to the main ground floor hallway in the Computer Engineering Department of the University of Parma (figure 3.5) and to the hallway of the Department of Computer Science and Engineering of the University of Washington (figure 3.6, map adapted from [44]). These environments allow verification of RTPF correctness while coping with several symmetric features, which may cause ambiguities in the choice of correct localization hypotheses. The environment of figure 3.6 had been

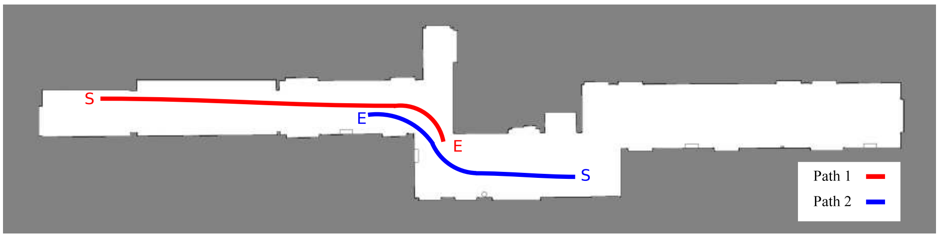


Figure 3.5: Map 1 – Hallway and simulated paths in the Computer Engineering Department, University of Parma.

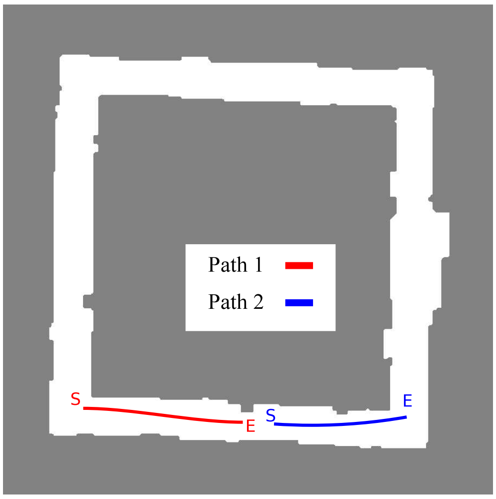


Figure 3.6: Map 2 – Hallway and simulated paths in the Department of Computer Science and Engineering, University of Washington.

exploited in [44] to verify RTPF correctness and has therefore been considered as a reference.

In simulation, the map is stored as a grid with a given resolution (0.20 m) and is used both to create simulated observations and to compute importance weights in correction steps. Data provided to the localizer consist of a sequence of laser scans and measurements: scanned ranges are obtained by ray tracing a beam on the discretized map. The measurement model is also based on ray tracing according to standard beam models for laser scanner [74]. In our tests we have used only three laser beams measuring distances to left, right and frontal obstacles; such poor sensor data stress the role of algorithm instead of sensor data. A gaussian additive noise was added to both range beams and robot movements representing environment inputs and robot state in simulation. Thus simulation tests are performed in an environment known in detail and are best suited for comparing performance between algorithms. The task of the robot is to achieve localization while moving in the environments of figures 3.5 and 3.6 along assigned trajectories. Simulated trajectories, labeled as Path 1 and Path 2 in figures 3.5 and 3.6, correspond to lengths of approximately 5 to 8 *m*.

Localization algorithms investigated are the original steepest descent-based one (RTPF-Grad) and the proposed RTPF based on the effective number of samples (RTPF-Eig). During these tests the partition set size was 1000 samples.

A summary of simulation results is reported in figures 3.7 and 3.8, where curves show the localization error for the two algorithms at each iteration by considering convergence to the maximal hypothesis. For both curves, each value is obtained by averaging the distances of the estimated pose from the real pose over 10 trials where localization eventually converged to the correct hypothesis within the maximum number of iterations (set to 40). For both algorithms there were also a few instances where localization did not converge to the correct hypothesis within the length of the path, although the correct hypothesis was the second best. These unsuccessful experiments were approximately 10% of all simulated localization trials. We did not verify whether the robot would eventually recover its correct pose in the environment with further navigation.

On the average, the two versions of the RTPF-based localizer converge to some few hypotheses after three iterations, and the common samples distribution is multimodal. Hence, cluster search leads to few hypotheses with different weight. In our tests a hypothesis close to the correct robot pose always exists, and when this hypothesis prevails there is a sudden change in localization error, as shown in figures 3.7 and 3.8. Convergence is helped by recognizable features, e.g. the shape of scans, but when the environment is symmetric it can be difficult to reach, especially with limited or noisy sensoriality. Of course, the mean error in figures 3.7 and 3.8 does not correspond to any of the simulated trials; rather, it is the result of averaging trials with quick convergence and trials where the convergence requires many more iterations.

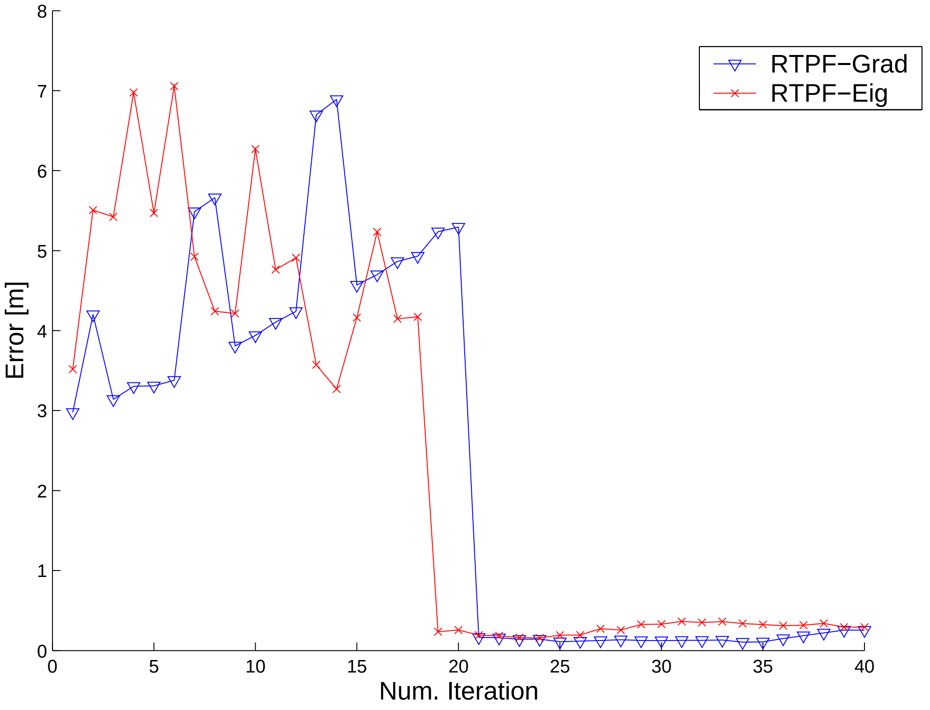


Figure 3.7: Performance of the two RTPF versions in the simulated environment of Map 1. The x-axis represents the iterations of the algorithm. The y-axis shows the average error distance of the estimated pose from robot pose.

Figure 3.9 provides an alternative view of the same data, as curves show the percentage of simulation trials converging to the correct hypothesis (i.e. with localization error less than 1.5 m) at each iteration. In a few simulations, the correct robot pose is recovered only after about 20 or 30 iterations, i.e. after sensing map features that increase the weight of the correct samples.

Empirically, for the examined environments RTPF-Eig seems to exhibit a slightly faster convergence, on the average, to the correct localization hypothesis, even though its average error at the last recorded iteration appears somewhat larger.

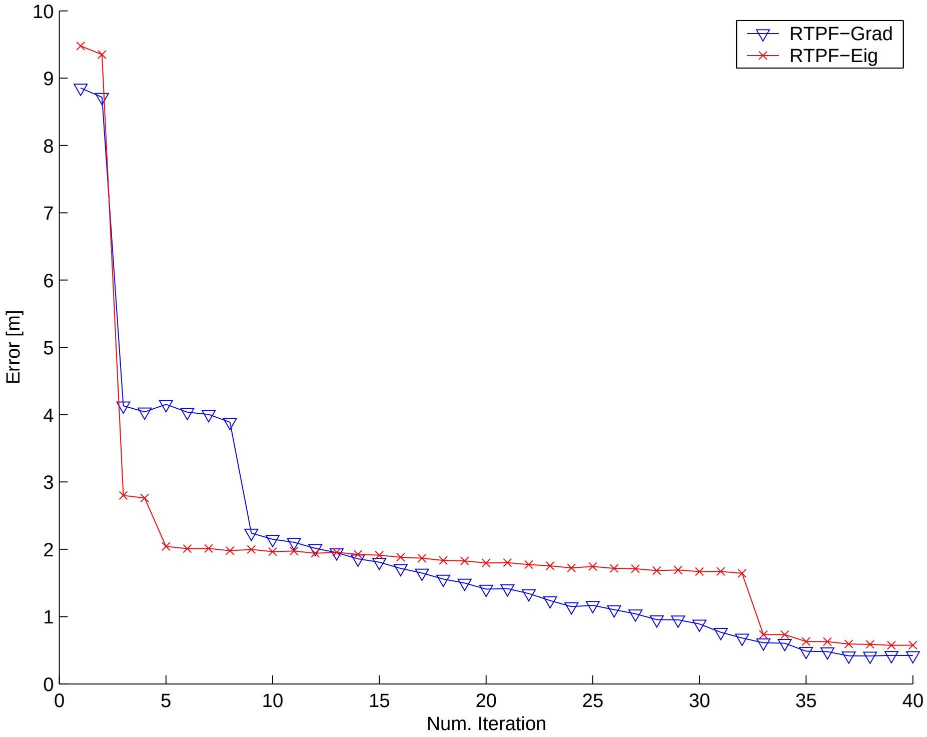


Figure 3.8: Performance of the two RTPF versions in the simulated environment of Map 2. The x-axis represents the iterations of the algorithm. The y-axis shows the average error distance of the estimated pose from robot pose.

3.5.2 Experiments

Real experiments were run in the environment of figure 3.5 collecting data with a Nomad 200 mobile robot equipped with a Sick LMS 200 laser scanner. The localizer was integrated in a real-time robot architecture [8]. The robot moved along Path 1 for about 5 m, from the left end of the hallway in steps of about $15-20$ cm and reading three laser beams from each scan in the same way of the simulation tests.

To assess the consistency of the localizer's output in an automated way, we compared the robot pose computed by the localizer (using the RTPF-Eig algorithm) with

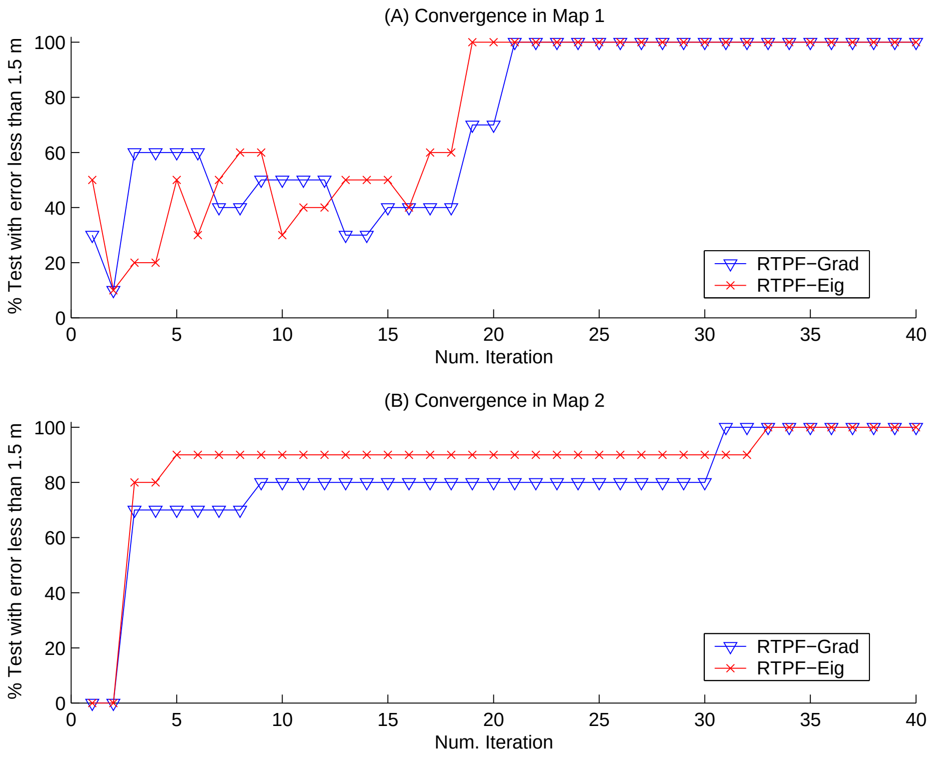


Figure 3.9: Percentage of simulation trials converged to the correct hypothesis, i.e. with localization error less than 1.5 m, during iterations for Map 1 (a) and Map 2 (b).

the one provided by an independent localization methodology. To this purpose, some visual landmarks were placed in the environment and on the mobile robot, and a vision system was exploited to triangulate the robot position based on these landmarks. The vision system provided an independent, coarse estimate of the robot pose at any step, and hence allowed to establish convergence of the RTPF-based localizer. The two localization estimates were computed concurrently at each location and stored by the robot.

Figure 3.10 shows the results of 10 tests of RTPF-Eig over about 20 iterations. In these real experiments RTPF-Eig achieves localization to the correct hypothesis

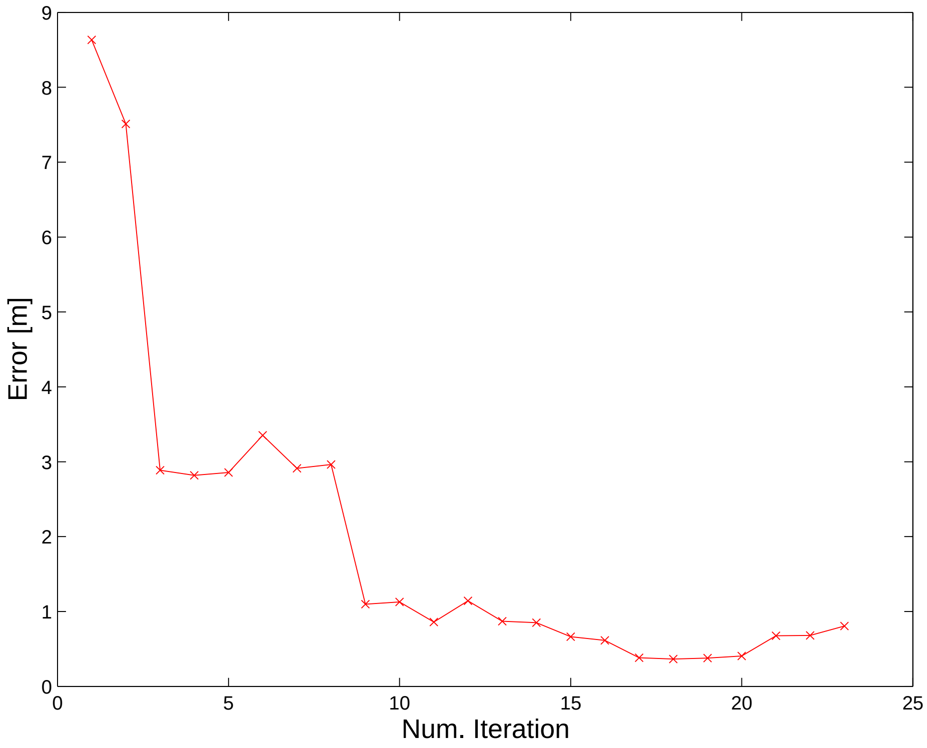


Figure 3.10: Performance of RTPF-Eig using real data collected in the hallway of Map 1.

very fast in most cases. After convergence, the maximum distance between RTPF-based and vision based estimates is about 70 *cm*, due to the compound error of the two systems. In real experiments in Map 1 localization was always successful within the length of the path. Moreover, results in figure 3.10 show that localization to the correct hypothesis was always reached in less than 10 iterations.

3.6 Discussion

In this chapter, we have presented a formal discussion of computation of mixture weights in RTPFs, along with an improved approach overcoming potential problems associated with the existing technique. The method proposed in this chapter computes mixture weights as the eigenvector of a matrix and thus avoids gradient descent, possibly prone to numerical instability. The method provides a balance of the effective sample size of partition sets on an estimation window.

The proposed approach has been implemented in a RTPF for localization with a mobile robot equipped with a laser range scanner, and evaluated in both simulation tests and real experiments. In two simulated environments, the new approach has achieved a localization performance similar to the original algorithm based on gradient of KLD, while avoiding the potential problems associated with gradient search methods. In real experiments with the mobile robot, the modified RTPF-based localization system has proven very effective, yielding correct localization within a small number of filter iterations.

Chapter 4

Incremental Maximum Likelihood Mapping

This chapter presents the incremental version of a maximum likelihood algorithm for robot mapping. The method uses a stochastic gradient descent to find the configuration that minimizes the negative log-likelihood of the map. By combining stochastic gradient descent with an efficient tree parameterization, this technique converges faster than several other approaches presented in literature. The incremental method is derived from an earlier ML algorithm [66, 32] where the map is solved offline and all the data must be available before the optimization.

The incremental tree network optimizer has been adapted to the online construction of the map. When a pose or a constraint is added to the network, the configuration of map variables is updated exploiting the previously computed result. The solutions adopted to avoid a complete re-evaluation of the network follow the locality criterion: the perturbation due to map augmentation is limited to the portion affected by the update, which is in the proximity of the inserted element. Unfortunately, feasibility of time constraints is not granted because the magnitude of the update depends on the topology of the network. However, the proposed optimizer limits update time when possible –i.e. when the closed loops are small compared with the connectivity in the parameterization tree illustrated later.

4.1 Stochastic Gradient Descent for Maximum Likelihood Mapping

In chapter 2 maximum likelihood approaches to graph-based SLAM have been introduced. The approach presented in this chapter also belongs to this class of methods. The goal of graph-based ML mapping algorithms is to find the configuration of the nodes that maximizes the likelihood of the observations.

Let $\mathbf{x} = (x_1 \cdots x_n)^T$ be a vector of parameters which describes a configuration of the nodes. Let δ_{ji} and Ω_{ji} be respectively the mean and the information matrix of an observation of node j seen from node i. Let $f_{ji}(\mathbf{x})$ be a function that computes a zero noise observation according to the current configuration of the nodes j and i.

Given a constraint between node j and node i, we define the *error* e_{ji} introduced by the constraint as

$$e_{ji}(\mathbf{x}) \quad = \quad f_{ji}(\mathbf{x}) - \delta_{ji} \tag{4.1}$$

as well as the *residual* $r_{ji} = -e_{ji}(\mathbf{x})$. Let $\mathscr{C} = \{\langle j_1, i_1 \rangle, \ldots, \langle j_M, i_M \rangle\}$ be the set of pairs of indices for which a constraint $\delta_{j_m i_m}$ exists. The goal of a ML approach is to find the configuration $\mathbf{x}^*$ of the nodes that minimized the negative log likelihood of the observations. Assuming the constraints to be independent, this can be written as

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{\langle j,i \rangle \in \mathscr{C}} r_{ji}(\mathbf{x})^T \Omega_{ji} r_{ji}(\mathbf{x}). \tag{4.2}$$

In the remainder of this section we describe how the general framework of stochastic gradient descent can be used for minimizing Eq. (4.2) and how to construct a parameterization of the network which increases the convergence speed.

4.1.1 Network Optimization using Stochastic Gradient Descent

Olson *et al.* [66] propose a variant of the preconditioned stochastic gradient descent (SGD) to compute the most likely configuration of the network nodes. The approach minimizes Eq. (4.2) by iteratively selecting a constraint $\langle j, i \rangle$ and moving the nodes of the network in order to decrease the error introduced by the selected constraint. Compared to the standard formulation of gradient descent, the constraints are not optimized as a whole but individually. The nodes are updated according to the following equation:

$$\mathbf{x}^{t+1} = \mathbf{x}^{t} + \lambda \cdot \mathbf{H}^{-1} J_{ji}^{T} \Omega_{ji} r_{ji} \tag{4.3}$$

Here $\mathbf{x}$ is the set of variables describing the locations of the poses in the network and $\mathbf{H}^{-1}$ is a preconditioning matrix. J_{ji} is the Jacobian of f_{ji}, Ω_{ji} is the information matrix capturing the uncertainty of the observation, r_{ji} is the residual, and λ is the learning rate which decreases with the iteration. For a detailed explanation of Eq. (4.3), we refer the reader to [32, 66].

In practice, the algorithm decomposes the overall problem into many smaller problems by optimizing subsets of nodes, one subset for each constraint. Whenever a solution for one of these subproblems is found, the network is updated accordingly. Obviously, updating the different constraints one after each other can have antagonistic effects on the corresponding subsets of variables. To avoid infinite oscillations, the learning rate λ reduces the fraction of the residual which is used for updating the variables. This strategy makes the solutions of the different sub-problems to asymptotically converge towards an equilibrium point that is the solution reported by the algorithm.

4.1.2 Tree Parameterization

The poses $\mathbf{p} = \{p_1, \ldots, p_n\}$ of the nodes define the configuration of the network. The poses can be described by a vector of *parameters* $\mathbf{x}$ such that a bidirectional mapping between $\mathbf{p}$ and $\mathbf{x}$ exists. The parameterization defines the subset of variables that are modified when updating a constraint. An efficient way of parameterizing the nodes is to use a tree. One can construct a spanning tree (not necessarily a minimum one) from the graph of poses. Given such a tree, we define the parameterization for a node as

$$x_i = p_i - p_{\text{parent}(i)}, \tag{4.4}$$

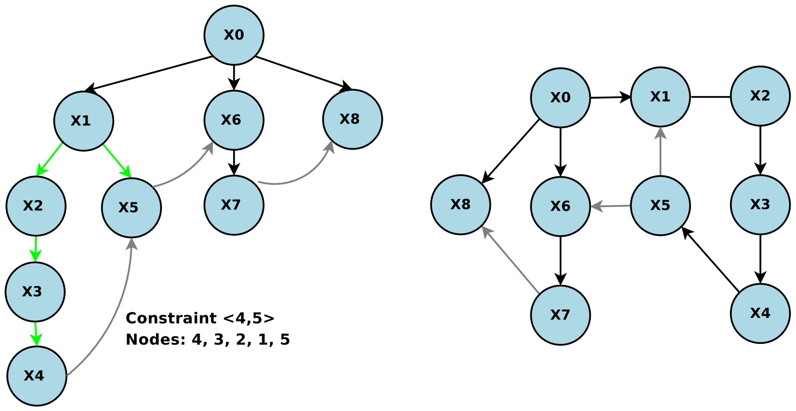


Figure 4.1: An example of tree parameterization.

where $p_{\text{parent}(i)}$ refers to the parent of node i in the spanning tree. Figure 4.1 shows an example of tree parameterization. As defined in Eq. (4.4), the tree stores the differences between poses. This choice is similar in the spirit to the incremental representation used in the Olson's original formulation, in that the difference in pose positions (in global coordinates) is used rather than pose-relative coordinates or rigid body transformations.

To obtain the difference between two arbitrary nodes based on the tree, one needs to traverse the tree from the first node upwards to the first common ancestor of both nodes and then downwards to the second node. The same holds for computing the error of a constraint. We refer to the nodes one needs to traverse on the tree as the path of a constraint. For example, $\mathscr{P}_{ji}$ is the path from node i to node j for the constraint $\langle j,i \rangle$. The path can be divided into an ascending part $\mathscr{P}_{ji}^{[-]}$ of the path starting from node i and a descending part $\mathscr{P}_{ji}^{[+]}$ to node j. We can then compute the residual in the global frame by

$$r'_{ji} \quad = \quad \sum_{k^{[-]} \in \mathscr{P}_{ji}^{[-]}} x_{k^{[-]}} - \sum_{k^{[+]} \in \mathscr{P}_{ji}^{[+]}} x_{k^{[+]}} + R_i \delta_{ji}. \tag{4.5}$$

Here R_i is the homogeneous rotation matrix of the pose p_i. It can be computed according to the structure of the tree as the product of the individual rotation matrices along the path to the root. Note that this tree does not replace the graph as an internal representation. The tree only defines the parameterization of the nodes.

Let $\Omega'_{ji} = R_i \Omega_{ji} R_i^T$ be the information matrix of a constraint in the global frame. According to [66], we compute an approximation of the Jacobian as

$$J'_{ji} \quad = \quad \sum_{k^{[+]} \in \mathscr{P}_{ji}^{[+]}} \mathscr{I}_{k^{[+]}} - \sum_{k^{[-]} \in \mathscr{P}_{ji}^{[-]}} \mathscr{I}_{k^{[-]}}, \tag{4.6}$$

with $\mathscr{I}_k = (0 \cdots 0 \underbrace{I}_{k^{\text{th}} \text{ element}} 0 \cdots 0)$. Then, the update of a constraint turns into

$$\mathbf{x}^{t+1} \quad = \quad \mathbf{x}^t + \lambda |\mathscr{P}_{ji}| \mathbf{M}^{-1} \Omega'_{ji} r'_{ji}, \tag{4.7}$$

where $|\mathscr{P}_{ji}|$ refers to the number of nodes in $\mathscr{P}_{ji}$. In Eq. (4.7), we replaced the preconditioning matrix $\mathbf{H}^{-1}$ with its scaled approximation $\mathbf{M}^{-1}$ as described in [66]. This substitution avoids a computationally expensive matrix inversion.

Let the *level* of a node be the distance in the tree between the node itself and the root. We define the *top node* of a constraint as the node on the path with the smallest level. Our parameterization implies that updating a constraint will never change the configuration of a node with a level smaller than the level of the top node of the constraint.

In principle, one could apply the technique described in this section as a batch algorithm to an arbitrarily constructed spanning tree of the graph. However, our proposed method uses a spanning tree which can be constructed incrementally, as described in the next section.

4.2 Overview of Tree Network Optimizer

This section presents an overview of the method described in previous sections. Algorithm 3, which illustrates a single gradient descent iteration, is essentially the same algorithm reported in [66] except for the tree parametrization. The algorithm requires the set of constraints $\mathscr{E}$ that are updated and the set of nodes $\mathscr{N}$ involved in the update. The batch version of this technique solves all the network constraints for the first time, so $\mathscr{E} = \mathscr{C}$. $\mathscr{N}$ is the set of nodes whose learning rate is updated according

```
Data: $\mathscr{E}$: set of affected constraints (default $\mathscr{C}$); $n_p$: number of nodes; $n_{iter}$:
    iteration number
Result: new node configuration $\mathbf{x}$
                                        /* Compute preconditioner $\gamma$ */
foreach $i = 1 \ldots n_p$ do
    $M_i = [0,0,0]$;
end
$\gamma = [\infty, \infty, \infty]$;
foreach $\langle i, j \rangle \in \mathscr{E}$ do
    compute rotation matrix $R_i$ from orientation of pose $p_i$;
    $\Omega'_{ji} = R\, \Omega_{ji}\, R^T$;
    foreach node $n \in \mathscr{P}_{ji}$ do
        $M_{index(n)} += diag(\Omega'_{ji})$;
        $\gamma = min(\gamma, diag(\Omega'_{ji}))$;
    end
end
                                        /* Gradient Descent */
foreach $\langle i, j \rangle\, in \mathscr{E}$ do
    compute rotation matrix $R$ from orientation of pose $p_i$;
    $\hat{p}_j = p_i \oplus \delta_{ji}$;
    $r = \hat{p}_j - p_j$;
    $r_\theta = atan2(\sin r_\theta, \cos r_\theta)$;
                                        /* Weight vector over the path */
    $tot = \frac{1}{n_{iter}} \left[ \frac{1}{\gamma_x} \left( \sum_{n \in \mathscr{P}_{ji}} \frac{1}{M_{n,x}} \right)^{-1}, \frac{1}{\gamma_y} \left( \sum_{n \in \mathscr{P}_{ji}} \frac{1}{M_{n,y}} \right)^{-1}, \frac{1}{\gamma_\theta} \left( \sum_{n \in \mathscr{P}_{ji}} \frac{1}{M_{n,\theta}} \right)^{-1} \right]$;
    foreach node $n \in \mathscr{P}_{ji}$ do
        $\lambda = learning\_rate(n, \mathscr{N})$;
        $w = \left[ \frac{tot_x}{M_{n,x}}, \frac{tot_y}{M_{n,y}}, \frac{tot_\theta}{M_{n,\theta}} \right]^T$;
        $x_n += \lambda |\mathscr{P}_{ji}| w^T \Omega'_{ji} r'_{ji}$;
    end
    update poses using new values of incremental parameters $x_n$
    for each node $n \in \mathscr{P}_{ji}$;
end
```

Algorithm 3: Computation of preconditioning matrix.

to adaptive learning rate as will be shown in next section. The batch version of the optimizer uses a constant learning rate γ.

A significant part of the algorithm is devoted to the computation of matrix $\mathbf{M}^{-1}$ approximating Hessian preconditioner $\mathbf{H}^{-1}$. Approximation with $\mathbf{M}^{-1}$ uses the diagonal of the global information matrix Ω'_{ji} as required by Jacobi preconditioning. Terms M_n accumulate the diagonal elements associated to the constraints that affect node n. The inverse of the matrix is approximated by inverting the each values of vector M_n. When gradient descent is performed, parameters M_n are used to distribute gradually the residual all over the path $\mathscr{P}_{ji}$ from node i to node j on the spanning tree. Since incremental tree parameterization is used, the error on constraint $\langle i, j\rangle$ is weighted according to the ratio of $1/M_n$ and $\sum_{k\in\mathscr{P}_{ji}} 1/M_k$. In the final step, the value of incremental paramers x_n is used to update the value of the poses.

4.3 Online Network Optimization

The algorithm presented in the previous section is a batch procedure. At every iteration, the poses of all nodes in the network are optimized. The fraction of the residual used in updating every constraint decreases over time with the learning rate λ, which evolves according to an harmonic progression. During online optimization, the network is dynamically updated to incorporate new movements and observations. In theory, one could also apply the batch version of our optimizer to correct the network. This, however, would require to compute a solution from scratch each time the robot moves or makes an observation, which would obviously lead to an inefficient algorithm.

In this section we describe an incremental version of our optimization algorithm, which is suitable for solving on-line mapping problems. As pointed out in [67] an incremental algorithm should have the following properties:

1. Every time a constraint is added to the network, only the part of the network which is affected by that constraint should be optimized. For example, when exploring new terrain, the effects of the optimization should not perturb distant parts of the graph.

2. When revisiting a known region of the environment it is common to re-localize the robot in the previously built map. One should use the information provided by the re-localization to compute a better initial guess for the position of the newly added nodes.

3. To have a consistent network, performing an optimization step after adding each constraint is often unnecessary. This happens when the newly added constraints are adequately satisfied by the current network configuration. Having a criterion for deciding when to perform unnecessary optimizations can save a substantial amount of computation.

In the remainder of this section, we present four improvements to the algorithm so that it satisfies the properties discussed above.

4.3.1 Incremental Construction of the Tree

When constructing the parameterization tree online, we can assume that the input is a sequence of poses corresponding to a trajectory of the robot. In this case, subsequent poses are located closely together and there exist constraints between subsequent poses resulting from odometry or scan-matching. Further constraints between arbitrary nodes result from observations when revisiting a place in the environment.

We proceed as follows: the oldest node is the root of the tree. When adding a node *i* to the network, we choose as its parent the oldest node for which a constraint to the node *i* exists. Such a tree can be constructed incrementally since adding a new node does not require to change the existing parts of the tree.

The pose p_i and parameter x_i of a newly added node *i* is initialized according to the position of the parent node and the connecting constraint as

$$p_i = p_{\mathrm{parent}(i)} \oplus \delta_{i,\mathrm{parent}(i)} \tag{4.8}$$

$$x_i = p_i - p_{\mathrm{parent}(i)}. \tag{4.9}$$

The parent node represents an already explored part of the environment and the constraint between the new node and the parent can be regarded as a localization event in an already constructed map, thus satisfying Property 2. As shown in the experiments described later in the chapter, this initialization appears to be a good heuristic for determining the initial guess of the pose of a newly added node.

4.3.2 Constraint Selection

When adding a constraint $\langle j, i \rangle$ to the graph, a subset of nodes needs to be updated. This set depends on the topology of the network and can be determined by a variant of breadth first visit. Let $\mathscr{G}_{j,i}$ be the minimal subgraph that contains the added constraint and has only one constraint to the rest of the graph. Then, the nodes that need to be updated are all nodes of the minimal subtree that contains $\mathscr{G}_{j,i}$. The precise formulation on how to efficiently determine this set is given by Algorithm 4.

Note that the number of nodes in $\mathscr{G}_{j,i}$ does depend only on the root of the tree and on the overall graph. It contains all variables which are affected by adding the new constraint $\langle i, j \rangle$.

4.3.3 Adaptive Learning Rates

Rather than using one learning rate λ for all nodes, the incremental version of the algorithm uses spatially adaptive learning rates introduced in [67]. The idea is to assign an individual learning rate to each node, allowing different parts of the network to be optimized at different rates. These learning rates are initialized when a new constraint is added to the network and they decrease with each iteration of the algorithm. In the following, we describe how to initialize and update the learning rates and how to adapt the update of the network specified in Eq. (4.7).

Initialization of the learning rates When a new constraint $\langle j, i \rangle$ is added to the network, we need to update the learning rates for the nodes $\mathscr{N}_{ji}$ determined in the previous section. First, we compute the learning rate λ'_{ji} for the newly introduced information. Then, we propagate this learning rate to the nodes $\mathscr{N}_{ji}$.

A proper learning rate is determined as follows. Let β_{ji} be the fraction of the residual that would appropriately fuse the previous estimate and the new constraint.

```
Data: $\langle j,i\rangle$: the constraint; $\mathscr{G}$: the graph; $\mathscr{T}$: the tree.
Result: $\mathscr{N}_{ji}$: the set of affected nodes; $\mathscr{E}_{ji}$: the affected constraints.
Queue $f = \text{childrenOf}(\text{topNode}(\langle j,i\rangle))$;
$\mathscr{E}_{ji} := \text{edgesToChildren}(\text{topNode}(\langle j,i\rangle))$;
foreach $\langle a,b\rangle \in \mathscr{E}_{ji}$ do
    $\langle a,b\rangle .mark = true$;
end
while $f \neq \{\}$ do
    Node $n := \text{first}(f)$;
    $n.mark := true$
    foreach $\langle a,b\rangle \in \text{edgesOf}(n)$ do
        if $\langle a,b\rangle .mark = true$ then
            continue;
        end
        Node $m := (a = n)?b : a$;
        if $m = \text{parent}(n)$ or $m.mark = true$ then
            continue;
        end
        $\langle a,b\rangle .mark = true$;
        $\mathscr{E}_{ji} := \mathscr{E}_{ji} \cup \{\langle a,b\rangle\}$;
        if $\langle a,b\rangle \in \mathscr{T}$ then
            $f := f \cup \{m\}$;
        else
            $f := f \cup \text{childrenOf}(\text{topNode}(\langle a,b\rangle))$;
        end
    end
    $f := \text{removeFirst}(f)$;
    $\mathscr{N}_{ji} := \mathscr{N}_{ji} \cup \{n\}$;
end
```

Algorithm 4: Construction of the set of nodes affected by a constraint. For readability we assume that the frontier f can only contain nodes which are not already marked.

Similar to a Kalman filter, β_{ji} is determined as

$$\beta_{ji} = \Omega_{ji}(\Omega_{ji} + \Omega_{ji}^{\text{graph}})^{-1}, \tag{4.10}$$

where Ω_{ji} is the information matrix of the new constraint, and $\Omega_{ji}^{\text{graph}}$ is an information matrix representing the uncertainty of the constraints in the network. Based on Eq. (4.10), we can compute the learning rate λ'_{ji} of the new constraint as

$$\lambda'_{ji} = \text{maxrow}\left(\frac{1}{|\mathscr{P}_{ji}|}(\beta_{ji} \oslash \mathbf{M}\Omega'_{ji})\right). \tag{4.11}$$

Here $\oslash$ represents the row by row division (see [67] for further details). The learning rate of the constraint is then propagated to all nodes $k \in \mathscr{N}_{ji}$ as

$$\lambda_k \quad \leftarrow \quad \max(\lambda_k, \lambda'_{ji}), \tag{4.12}$$

where λ_k is the learning rate of the node k. According to Eq. (4.11) constraints with large residuals result in larger learning rate increases than constraints with small residuals.

Update of the network When updating the network, one has to consider the newly introduced learning rates. During an iteration, we decrease the individual learning rates of the nodes according to a generalized harmonic progression [70]:

$$\lambda_k \quad \leftarrow \quad \frac{\lambda_k}{1 + \lambda_k} \tag{4.13}$$

In this way, one guarantees the strong monotonicity of λ_k and thus the convergence of the algorithm to an equilibrium point.

The learning rates of the nodes cannot be directly used for updating the poses since Eq. (4.7) requires a learning rate for each constraint and not for each node. When updating the network given the constraint $\langle j, i \rangle$, we obtain an average learning rate $\tilde{\lambda}_{ji}$ from the nodes on $\mathscr{P}_{ji}$ as

$$\tilde{\lambda}_{ji} \quad = \quad \frac{1}{|\mathscr{P}_{ji}|} \sum_{k \in \mathscr{P}_{ji}} \lambda_k. \tag{4.14}$$

Then, the constraint update turns into

$$\Delta \mathbf{x}_k \quad = \quad \tilde{\lambda}_{ji} |\mathscr{P}_{ji}| \mathbf{M}^{-1} \Omega'_{ji} r'_{ji}. \tag{4.15}$$

4.3.4 Scheduling the Network Optimization

When adding a set of constraints $\langle j,i\rangle \in \mathscr{C}_{\text{new}}$ to a network without performing an optimization, we can incrementally compute the error of the network as

$$e_{\text{new}} = \sum_{\langle j,i\rangle \in \mathscr{C}_{\text{old}}} r_{ji}^T \Omega_{ji} r_{ji} + \sum_{\langle j,i\rangle \in \mathscr{C}_{\text{new}}} r_{ji}^T \Omega_{ji} r_{ji}. \tag{4.16}$$

Here e_{new} is the new error and $\mathscr{C}_{\text{old}}$ refers to the set of constraints before the modification.

To avoid unnecessary computation, we perform the optimization only if needed. This is the case when the newly incorporated information introduced a significant error compared to the error of the network before. The optimization is performed if

$$e_{\text{new}} - e_{\text{old}} > \alpha \max_{\langle j,i\rangle \in \mathscr{C}_{\text{old}}} r_{ji}^T \Omega_{ji} r_{ji} \tag{4.17}$$

The above condition is verified when the addition of new constraints causes an increment of global error greater than the error obtained with α worst constraints in the previous map. This criterion is different from the heuristic proposed in [30]

$$\frac{e_{\text{new}}}{|\mathscr{C}_{\text{new}}| + |\mathscr{C}_{\text{old}}|} > \alpha' \max_{\langle j,i\rangle \in \mathscr{C}_{\text{old}}} r_{ji}^T \Omega_{ji} r_{ji}. \tag{4.18}$$

Here α' is a user-defined factor that allows the designer of a mapping system to adapt the quality of the incremental solutions to the needs of the specific application. The interpretation of parameter α' is less clear.

If we assume that the network in $\mathscr{C}_{\text{old}}$ has already converged, this heuristic triggers an optimization only if a significant inconsistency is revealed. Furthermore, the optimization only needs to be performed for a subset of the network and not for the whole network. The subset is given by

$$\mathscr{E} \quad = \bigcup_{\langle j,i\rangle \in \mathscr{C}_{\text{new}}} \mathscr{E}_{ji}. \tag{4.19}$$

Here $\mathscr{E}_{ji}$ is the set of constraints to be updated given a new constraint $\langle j,i\rangle \in \mathscr{C}_{\text{new}}$. The sets $\mathscr{E}_{ji}$ are computed according to Algorithm 4. This criterion satisfies Property 3 and leads to an efficient algorithm for incrementally optimizing the network of constraints.

4.4 Results on Incremental Tree Network Optimizer

This section is designed to evaluate the effectiveness of the proposed methods to incrementally learn maximum likelihood maps. We first show that such a technique is well suited to generate accurate grid maps given laser range data and odometry from a real robot. Second, we provide simulation experiments to evaluate the evolution of the error and provide comparisons to our previously proposed techniques [32, 66, 67]. Finally, we illustrate the computational advantages resulting from our algorithm.

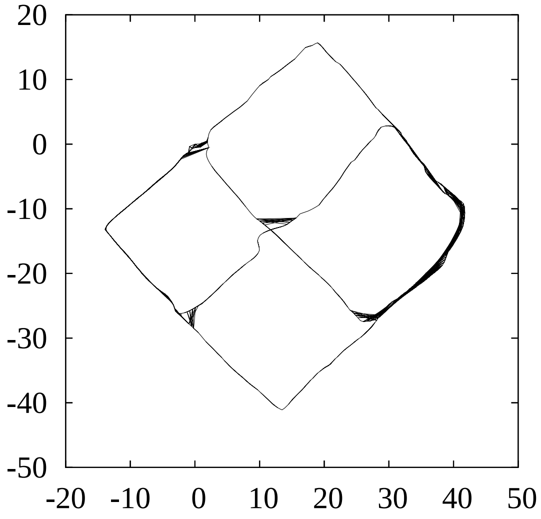
4.4.1 Real World Experiments

To illustrate that our technique can be used to learn maps from real robot data, we performed tests on two available datasets: the ACES building at UT Austin campus (ACES) and the Intel Research Lab (INTEL) [36]. During this experiment, constraints between consecutive poses have been extracted by means of pairwise scan matching. Loop closures were determined by localizing the robot in the previously built map by means of a particle filter. The above operations were performed using a part of GMapping algorithm by Grisetti *et al.* [31]. Figure 4.2 shows the optimized networks for the two datasets. Both the images do not represent the environment map, but the trajectory of the robot. In the case of INTEL dataset, the motion of robot from the hallway to each room is appearent. Even though the picture of map is not shown, several map details like orthogonality of walls and the lack of inconsistencies show that the proposed approach leads to accurate maps for real robot data.

4.4.2 Statistical Experiments on Error Evolution

In the these experiments, we moved a virtual robot on a grid world. An observation is generated each time the current position of the robot was close to a previously visited location. The observations are corrupted by a given amount of Gaussian noise. The network used in this experiment is depicted in Figure 4.3.

We compare our approach named *Tree Incremental* with its offline variant [32] called *Tree Offline* which solves the overall problem from scratch. In addition to that,



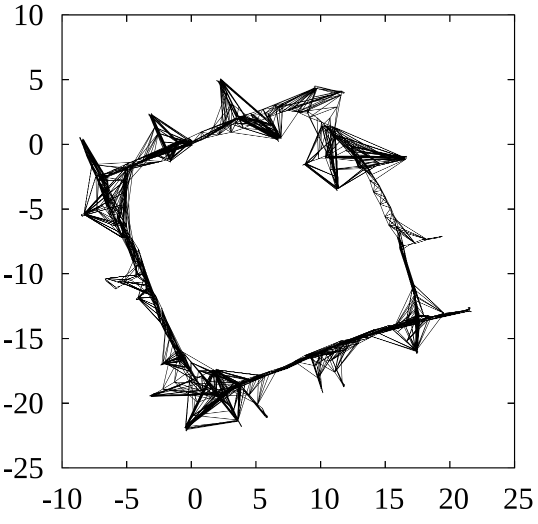


Figure 4.2: Results of incremental building and optimization for datasets ACES (top) and INTEL (bottom). The images represents networks of poses.

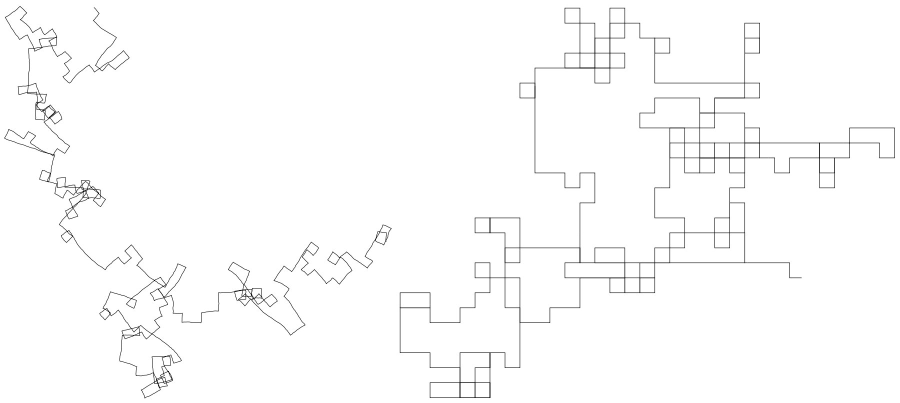


Figure 4.3: Network used in the simulated experiments. Left: initial guess. Right: ground truth.

we compare it to the offline version without the tree optimization [66] called *Olson Offline* as well as its incremental variant [67] referred to as *Olson Incremental*. For space reasons, we omit comparisons to LU decomposition, EKF, and Gauss-Seidel. The advantages of our method over these other methods is similar to those previously reported [66].

To allow a fair comparison, we disabled the scheduling of the optimization of Eq. (4.18) and we performed 30 iterations every time 16 constraints were added to the network. During the very first iterations, the error of all approaches may show an increase, due to the bigger correction steps which result from increasing the learning rates.

Figure 4.4 depicts the evolution of the error for all four techniques during a mapping experiment. We depicted two situations. In the first one, the robot closed a small loop. As can be seen, the introduced error is small and thus our approach corrects the error within 2 iterations. Both incremental techniques perform better than their offline variants. The approach proposed in this paper outperforms the other techniques. The same holds for the second situation in which the robot was closing a large loop. Note that in most cases, one iteration of the incremental approach can be carried out faster, since only a subpart of the network needs to be updated.

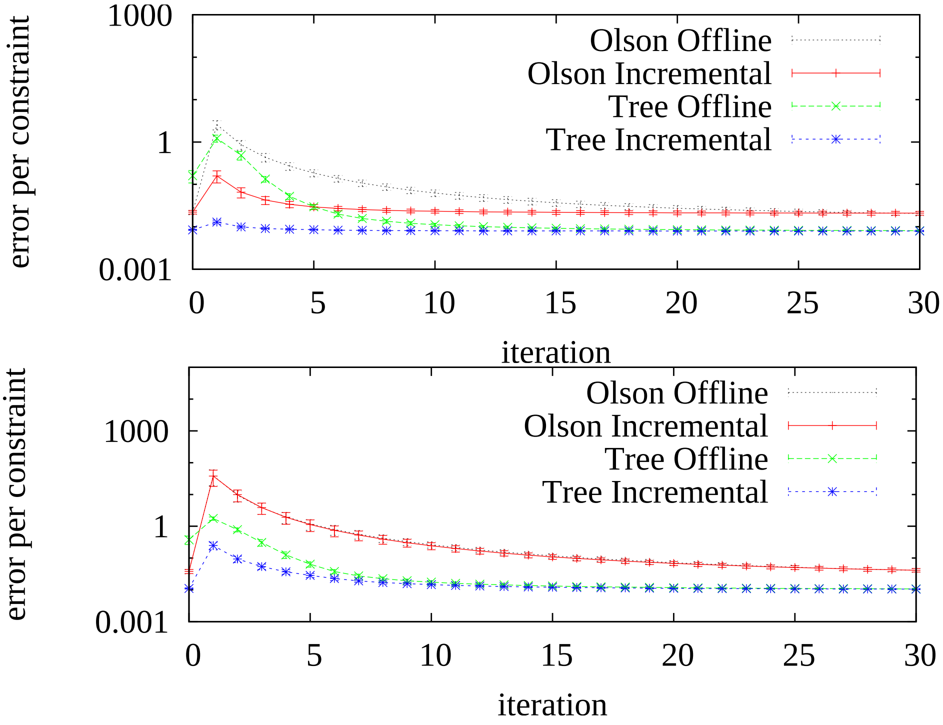


Figure 4.4: Statistical experiments showing the evolution of the error per iteration of the algorithm. Top: situation in which the robot is closes a small loop. Bottom: closure of a large loop. The statistics have been generated by considering 10 different realizations of the observation noise along the same path.

4.4.3 Runtime Comparison

Finally, we evaluated our incremental version and its offline variant with respect to the execution time. Both methods where executed only when needed according to our criterion specified by Eq. (4.18). We measured the time needed to run the individual approach until convergence to the same low error configuration, or until a maximum number of iterations (30) was reached. As can be seen in Figure 4.5(top), the incremental technique requires significantly less operations and thus runtime to provide equivalent results in terms of error. Figure 4.5(bottom) shows the error plot of a comparison of our approach and Treemap [28] proposed by Frese. As shown in the error-plot, in the beginning Treemap performs slightly better than our algorithm, due to the exact calculation of the Jacobians. However, when closing large loops Treemap is more sensitive to angular wraparounds (see increase of the error at constraint 2400 in Figure 4.5). This issue is typically better handled by our iterative procedure. Overall, we observed that for datasets having a small noise Treemap provides slightly better estimates, while our approach is generally more robust to extreme conditions.

4.5 Building the Constraint Network from Laser Scans

In this section we describe the part of the system that transforms raw sensor data into a network of constraints ready to be processed by the optimizer described in the previous section. The aim is to extract from laser scans a graph consisting of poses and constraints among pairs of poses. Therefore, map estimation does not depend explicitly on environment features, but only on robot poses like in [55].

A constraint between two poses consists of a relative transformation vector (translation and rotation) and second order statistics, i.e. the information matrix describing the uncertainty on the transformation. The primitive operation to extract these data from a pair of laser scans is a scan matching algorithm. Several scan matching algorithms have been proposed and many of them belong to the *iterative closest point* (ICP) approach. In this work, we adopted the ICP variant described in [9], whose implementation is also available.

However, scan matching is not enough to build the constraints network. This tech-

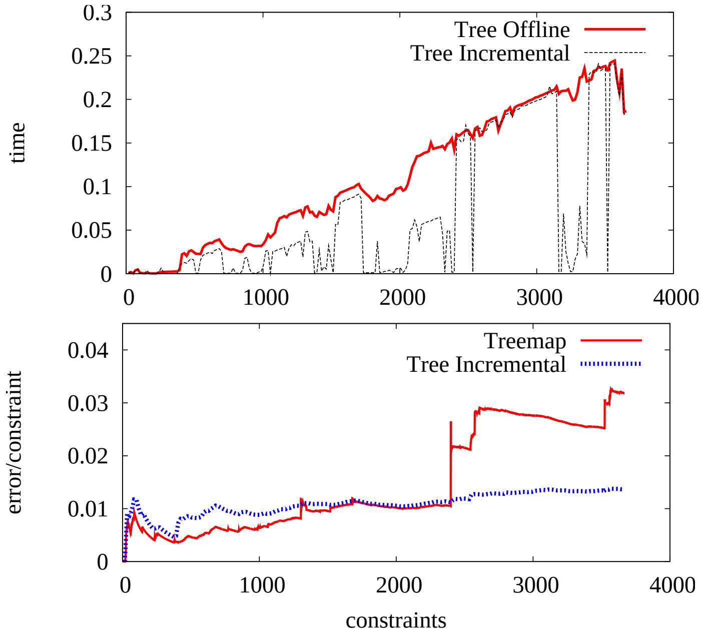


Figure 4.5: Top: runtime comparison of the offline and the incremental approaches using a tree parameterization. The optimization is performed only when the error condition specified by Eq. (4.18) was verified. Bottom: Comparison of the evolution of the global error between Treemap and the online version of our approach.

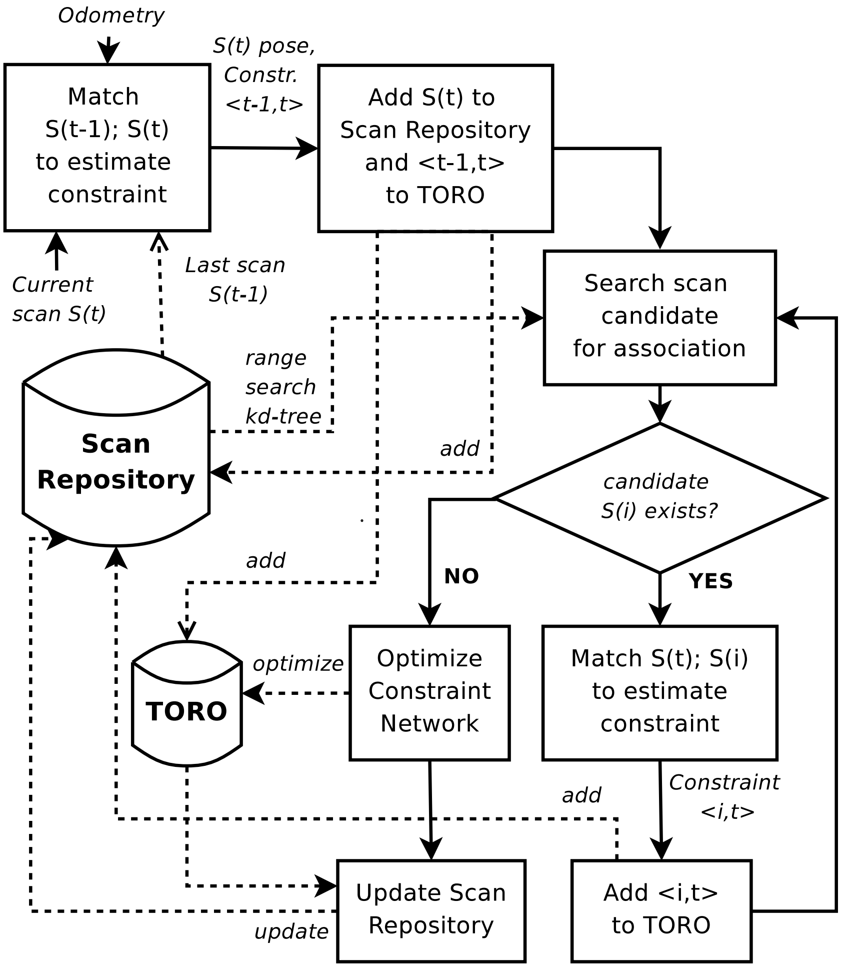

Odometry
Match S(t-1); S(t) to estimate constraint
S(t) pose, Constr. <t-1,t>
Add S(t) to Scan Repository and <t-1,t> to TORO
Current scan S(t)
Last scan S(t-1)
Search scan candidate for association
range search kd-tree
Scan Repository
add
candidate S(i) exists?
NO
YES
add
TORO
optimize
Optimize Constraint Network
Match S(t); S(i) to estimate constraint
add
Constraint <i,t>
Update Scan Repository
update
Add <i,t> to TORO

Figure 4.6: General schema of the `ScanMap` to build constraint network.

nique is effective when two scans significantly overlap and the initial guess of relative pose is not too far from the correct value. These conditions hold when consecutive poses are compared, but the comparison fails when the robot moves in a previously explored region and tries to match current observation with the stored map. Two different network builders, named `ScanMap` and `GraphMap` respectively, have been

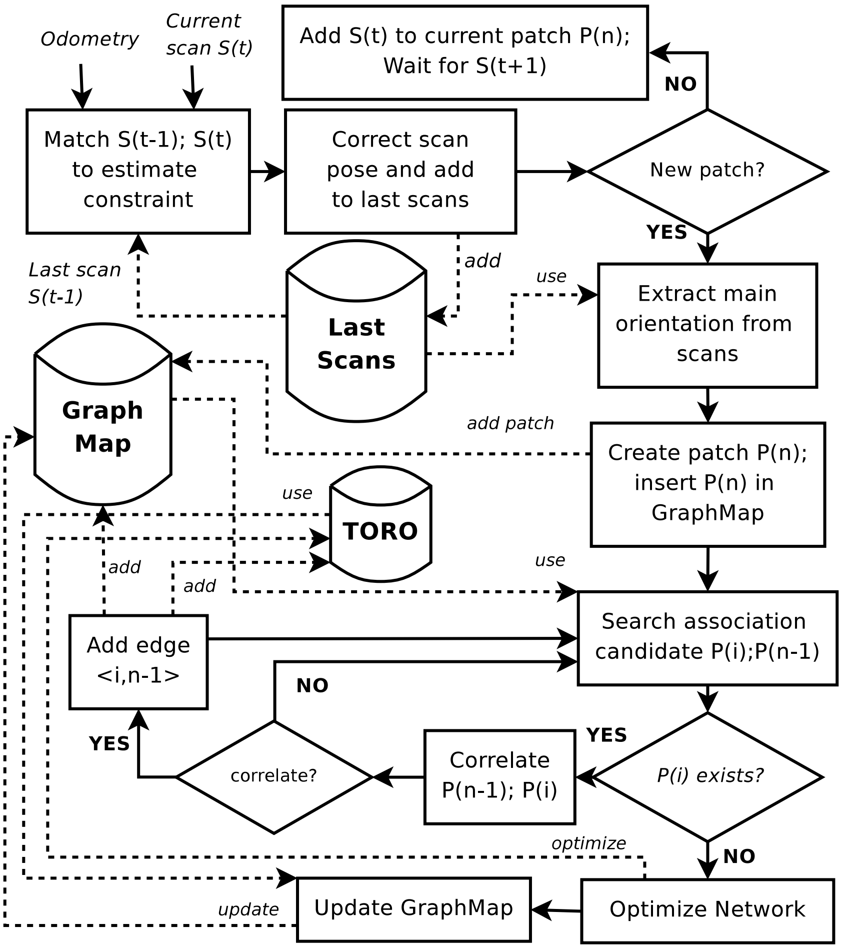

Odometry
Current scan S(t)
Add S(t) to current patch P(n); Wait for S(t+1)
NO
Match S(t-1); S(t) to estimate constraint
Correct scan pose and add to last scans
New patch?
YES
add
Last scan S(t-1)
use
Last Scans
Extract main orientation from scans
Graph Map
add patch
Create patch P(n); insert P(n) in GraphMap
use
TORO
add
add
use
Search association candidate P(i);P(n-1)
Add edge <i,n-1>
NO
YES
YES
correlate?
Correlate P(n-1); P(i)
P(i) exists?
optimize
NO
update
Update GraphMap
Optimize Network

Figure 4.7: General schema of the `GraphMap` to build constraint network.

developed.

`ScanMap` [50] exploits a straightforward representation of the map: each node of the graph stores a pose and the corresponding laser scan acquired when the robot visited the pose. The scan associated to a location provides a sort of local map. Figure 4.6 displays the main steps of the algorithm. Initially, the pose of new node is initialized with the relative motion refined by scan matcher. When a loop is closed, the algorithm selects candidate scans for data association, tries to match current pose and candidates and performs an optimization step on them. Unfortunately, the constraints extracted with this procedure are not always estimated correctly as will be shown later.

`GraphMap` [51] (Figure 4.7) has been developed to overcome `ScanMap` limitations and is similar to the method described in [34]. In particular, the local map stored in each node is an occupancy grid map obtained from several aligned consecutive scans. Thus, the number of nodes significantly decreases and loop closure becomes more effective since each local map patch better describes a location. Data association is performed with a correlation-like algorithm. In the following, the two algorithms are discussed thoroughly.

4.5.1 ScanMap

`ScanMap` represents the map as a collection of pairs of poses and scans. Such solution is quite straightforward and completely relies on scan matching technique for both pose initialization and loop closure. The initial value of a pose depends on the value of the previous pose and on the relative transformation between them due to the motion of the robot. Relative pose is first estimated with odometry and then it is corrected by scan alignment.

The core of the map builder is therefore the loop closure detector that performs in two steps. After the addition of a new pose, candidate cycles are found via range search on a *kd-tree* [3]. The size of the searching region depends on the size of the current map and on the uncertainty in the estimation. Furthermore, if a loop has been closed recently, the search is concentrated on a smaller area.

Candidate poses for data association are then selected from the poses inside searching area. Recent nodes, i.e. nodes directly connected to current node through a path that falls inside the searching area, are not considered to avoid too small or repeated loops. Several criteria are applied to reduce candidates and detect correct associations. Alignment of current scan with a candidate scan estimates their relative pose. Scan matching is performed starting from null initial guess. The most likely candidate is then selected by evaluating the match covariance values, the distance between the pair of poses, and the extent of the overlapping area.

Main drawbacks of the approach are the limited metric accuracy and the need to choose parameters for heuristic rules like the range size for the preliminary search and the threshold for covariance. Discussion follows in the next session.

4.5.2 GraphMap

`GraphMap` consists of a graph whose nodes stores poses with a local map and whose edges correspond to constraints between pairs of nodes like in `ScanMap`. The difference between the two algorithms lies in the interpretation of nodes. Indeed, the pose stored in each node is the local reference frame of the local map and does not correspond to the robot trajectory. Therefore, a local map consists of an occupancy grid map built from a set of consecutive laser scans. Each cell of the local map has one of the following values: *empty*, *occupied* or *unknown*. A laser scan is inserted in a map patch by placing the center of the scan in the map and tracing each beam until the measured distance is reached. Traversed cells are filled with *empty* value, until the last cell is labeled as *occupied*. Such a representation, also called *map patch*, is more expressive than a collection of raw scans and allows a more effective data association.

Figure 4.6(b) illustrates the main steps of the algorithm. First, the new scan is aligned with the previous one by scan matching to improve the estimation of relative motion provided by odometry. Then, the observation is stored in a temporary buffer containing the last scans available for any comparison. If the distance between the current robot pose and the center of map patch is less than a given threshold, then the scan is used to fill the current grid map. For the experiments illustrated in the next section the threshold was set to 6 m. Otherwise a new patch is created.

Since occupancy grid maps are suitable for the representation of obstacles extending along directions orthogonal to patch borders, the algorithm extracts the main orientation of the first scan added to the patch. The main orientation intuitively corresponds to the bundle of parallel lines that better fits the length of the obstacles and it is operatively computed by searching maxima in the *Hough spectrum* [7]. Finding a principal direction also helps when two grid maps are compared for data association. Main orientation is usually well-defined for indoor structured environment, but sometimes it is indistinguishable, as pointed out in the next section.

After the insertion of a new node, GraphMap searches for candidate nodes for data association with the now completed map patch. Candidates are found with range search as illustrated before and then are matched with the current patch by exploiting a correlation-like function. Assuming that relative orientation of two local maps is obtained by comparing their main orientations, two occupancy grids are better performed by a correlation function that operates on pairs of matching cells. In order to limit computational load, displacement between two patches is computed by correlating their separate histograms for axes x and y. A histogram bin counts the number of occupied cells whose projection falls inside the bin. Displacement between two maps is computed searching for maxima of correlations between x-histograms and y-histograms. Finally, the algorithm performs an acceptance test on the overlapping parts of the two patches placed according to the estimated relative position and orientation. This test counts the number of cells with equal values and ignores the cells with *unknown* value. If the outcome of the test is positive, i.e. the number of agreeing cells is larger than a given threshold, a link between the two nodes is added to GraphMap and to TORO and an optimization is performed.

4.5.3 Results

In this section, we evaluate and compare the effectiveness of the two proposed systems for map learning based on the tree network optimizer. To test the two map builders with real robot data, the freely available ACES dataset [36] was used.

This dataset provides a sequence of laser scans acquired sequentially as the robot moves in an environment. The reported experiment consists in the extraction of constraints relating consecutive poses from pairs of scans and in the detection of cycles with the previously discussed techniques.

Figure 4.8 illustrates the main problem with the first map estimator proposed in this chapter: when a loop closure is performed, ScanMap correctly recognizes a known region, but the scan matcher does not provide a good estimation of relative constraints between the scans stored in matching nodes. The result shown in Fig-

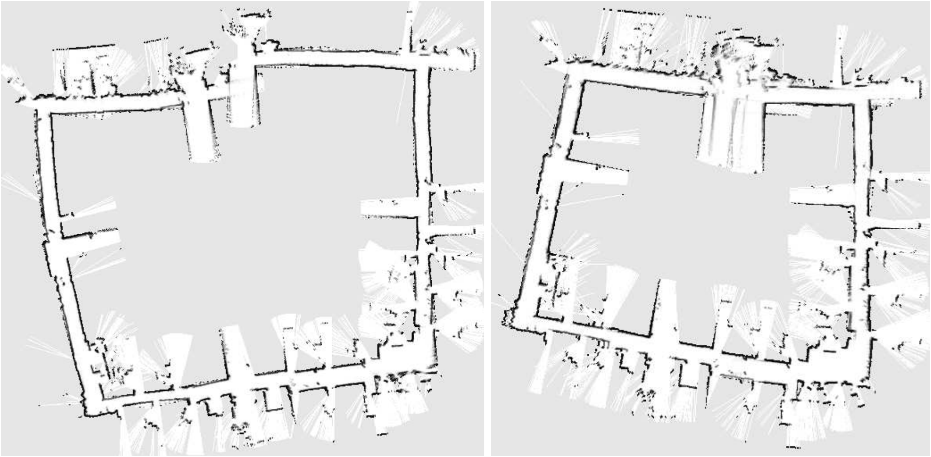


Figure 4.8: Map of the environment of dataset ACES before (left) and after (right) a loop closure is solved with `ScanMap`.

ure 4.8(right) shows the effect of the new constraint: the estimated distance between the two places is too short and the resulting map is pulled on the top part.

The map estimation performed by `GraphMap` is shown in figure 4.9. The internal representation adopted by the algorithm is apparent: the map consists of a set of occupancy grid maps associated to the nodes of the network. The boundaries of each map are represented by the rectangular bounding box and the *occupied* cells are represented by a point. For readability, the output does not distinguish *empty* cells from *unknown* ones. Note that the orientation of each submap is consistent with the orientation of map walls of the represented region.

The size and resolution of each grid map are fixed. In this experiment the value of the grid map size is equal to 20.0×10.0 *m* with a resolution of 0.2 *m*. The size of the map patch may appear large if compared to the size of the whole map (about 55×45 *m*), but it is a trade-off between the local adjustment of consecutive scans allowed by the scan matcher and the need for distinguishable map. Indeed, smaller map patches appear too similar to each other and the resulting loop closure detection is too prone to matching errors.

The execution of `GraphMap` with online optimization with TORO on an Intel

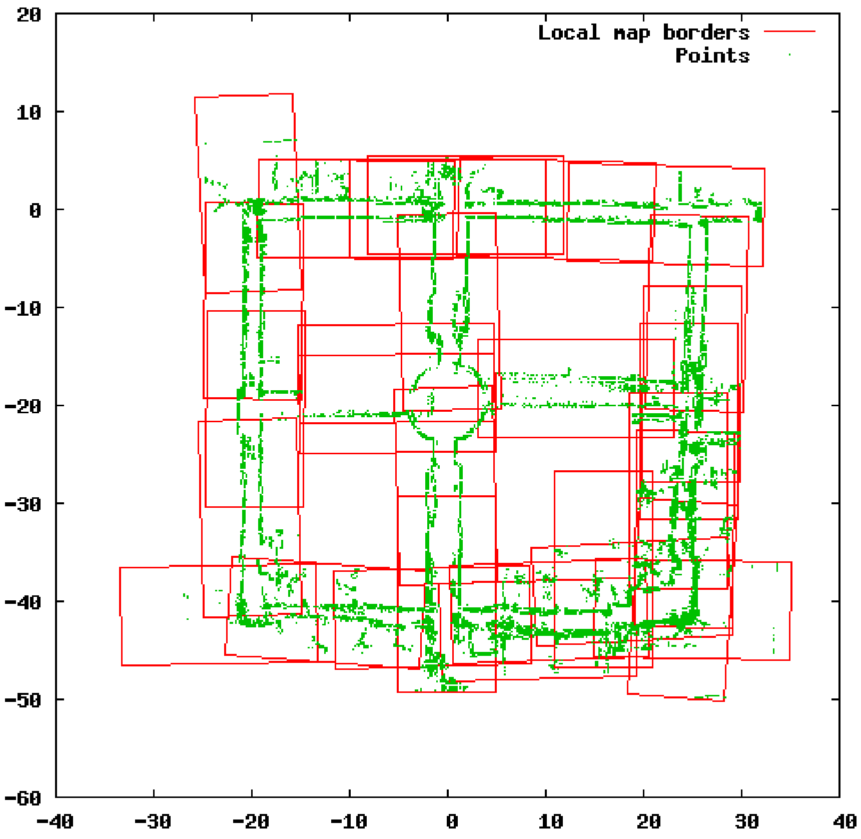


Figure 4.9: Map of the environment of dataset ACES after an optimization performed with GraphMap.

Core 2 Quad Q9300 2.5 GHz has required about 42.73 s. The network optimizer has no significant impact on the total amount of time. GraphMap generates only 41 nodes compared to the about 700 nodes used by ScanMap. Thus, GraphMap reduces the complexity of the network optimization.

4.6 Discussion

In this chapter, we have presented an incremental maximum likelihood algorithm for mapping tasks. The method improves over earlier maximum graph based approaches to SLAM by integrating several heuristics and optimizations which make it suitable for incremental online operation.

We have integrated the incremental tree network optimizer with a scan matcher operating on laser scans. After evaluating two alternative map representations, we have developed a complete mapping system, capable of building a reasonably accurate metric-topological map of an unknown indoor environment, as shown by tests on a benchmark dataset.

Chapter 5

Parallel decomposition of Mapping problem

In this chapter a parallel algorithm for map estimation based on Gauss-Seidel relaxation is presented. The map is given in the form of a constraints network and is partioned into clusters of nodes by applying a node-tearing technique. The identified clusters of nodes can then be processed independently as tasks assigned to different processors. This algorithm can be used to exploits the computational resources of multicore processors and is suitable for solving distributed mapping in multi-robot context.

5.1 Motivation and Problem

The discussion in the previous chapter has shown that the maximum likelihood approach is suitable for local decomposition due to the sparsity of the graphical model representing the full SLAM formulation. Such sparsity has been exploited by several maximum likelihood algorithms to reduce the computational complexity and to allow online execution. The graph decomposition can be hierarchical or concerns the portion of graph involved by the addition of new measurements, but it is allowed by the sparsity of the information matrix in ML approaches that avoids repeated marginalization. Given such decomposition, the update of network configurations focuses only on a portion of the map.

Decomposition of the network allows also the introduction of *parallelism* in algorithms for map estimation. There are several advantages in a parallel design. First, map estimation is sped-up by computing solutions for different graph clusters on different threads, when a proper hardware is available. The decomposition of the problem in tasks for different processors requires a distributed implementation. Recently, the diffusion of multi-core processors has also stimulated multi-threaded single process implementations. In any case, the impact of non-parallelizable portions of the algorithms and the overhead due to data sharing and synchronization should be carefully considered. Moreover, a parallel design of SLAM algorithms simplify their extension to a multi-robot context. When only one robot is involved, data acquisition and map-building are intrinsically serial operations. However, graph decomposition is required when the map is concurrently estimated by two or more robots. In multi-robot contexts, map decomposition is sometimes determined by the activity of each robot as shown in [15].

In this chapter, we present a parallel algorithm that solves SLAM problems where a set of constraints is provided as input. The algorithm combines Gauss-Seidel relaxation and a reordering of variables that transforms the information matrix in block-bordered-diagonal form. The graph associated to the information matrix is partitioned into clusters of connected nodes. A special cluster contains all nodes separating pairs of clusters. Hence, the Gauss-Seidel update of the value of a pose in a cluster depends either on another pose in the same cluster or on a separator node. The Gauss-Seidel update depends on variables whose value could have been already updated in current iteration or not, and the order of variables matter. Thus, each cluster can be computed independently if the separator set is computed after the other clusters.

Graph clustering is performed with a node-tearing heuristic algorithm. The efficiency of the proposed algorithm depends on the sparsity of the information matrix that determines the result of decomposition. Gauss-Seidel relaxation is chosen because easy to parallelize, but it has the drawback of being an offline method.

5.2 Gauss-Seidel Relaxation for Maximum Likelihood Mapping

This section discusses the general formulation of the maximum likelihood (ML) approach and a solution algorithm based on Gauss-Seidel relaxation to solve the resulting equations. The SLAM problem is formulated as a graph, whose nodes correspond to the variables of the map and whose edges represent the constraints between pairs of these variables.

In the following, *delayed-state* representation [22] is used instead of *feature based* one. According to delayed-state representation the map is represented by robot poses. Relations between poses are obtained by scan matching [55] or by reduction of features [74]. Then let $\mathbf{x} = (x_1 \ \cdots \ x_n)^T$ be the vector of robot poses. Let δ_{ji} and Ω_{ji} be respectively the mean and the information matrix of an observation of node j seen from node i. Let $f_{ji}(\mathbf{x})$ be a function that computes a zero noise observation according to the current configuration of the nodes j and i

$$f_{ij}(\mathbf{x}) = \left[\begin{array}{c} (x_{j,x} - x_{i,x}) \ \cos x_{i,\theta} + (x_{j,y} - x_{i,y}) \ \sin x_{i,\theta} \\ -(x_{j,x} - x_{i,x}) \ \sin x_{i,\theta} + (x_{j,y} - x_{i,y}) \ \cos x_{i,\theta} \\ x_{j,\theta} - x_{i,\theta} \end{array} \right] \tag{5.1}$$

Thus, the error on constraint $\langle j, i \rangle$ is given by

$$e_{ji}(\mathbf{x}) \quad = \quad f_{ji}(\mathbf{x}) - \delta_{ji} \tag{5.2}$$

Let $\mathscr{C} = \{\langle j_1, i_1 \rangle, \ldots, \langle j_M, i_M \rangle\}$ be the set of pairs of indices for which a constraint $\delta_{j_m i_m}$ exists. Then the aim of ML approach is to minimize the negative log-likelihood or error function on the observation

$$\chi^2(\mathbf{x}) \quad = \quad \sum_{\langle j,i \rangle \in \mathscr{C}} e_{ji}^T(\mathbf{x}) \ \Omega_{ji} \ e_{ji}(\mathbf{x}) \tag{5.3}$$

Several numeric techniques have been proposed in order to find the minimum of $\chi^2(\mathbf{x})$. In this section, we illustrate part of the relaxation algorithm proposed by Frese *et al* [29]. The algorithm consists of two steps. First, the observation functions $f_{ij}(\mathbf{x})$

are linearized around the current value of the network configuration $\hat{\mathbf{x}}$

$$e_{ij}(\mathbf{x}) \approx f_{ij}(\hat{\mathbf{x}}) - \delta_{ji} + J_{ij}^{i}(x_i - \hat{x}_i) + J_{ij}^{j}(x_j - \hat{x}_j) \tag{5.4}$$

where J_{ij}^{i} and J_{ij}^{j} are the Jacobians of the observation function with respect to x_i and x_j evaluated in point $\hat{x}_i$ and $\hat{x}_j$. Since Eq. (5.1) only depends on poses i and j, there are no additional terms.

Then, the resulting negative log-likelihood $\chi^2(\mathbf{x})$ is approximated by a quadratic function

$$\chi^2(\mathbf{x}) \quad \approx \quad x^T\, A\, x + 2bx^T\; b + c \tag{5.5}$$

The minimum of the linearized function is easily found by solving the linear system $A\ \mathbf{x} = b$. The method proposed in [29] to perform this final step is Gauss-Seidel relaxation. The value of each pose x_i is computed individually by solving the single block-row equation i with fixed value of x_j ($j \neq i$). Let A_{ij} be the block of matrix A corresponding to block-row i and block-column j; let b_i be the values for block-row i. Respectively, we have

$$A_{ij} \quad = \quad \sum_{\langle j,i\rangle \in \mathscr{C}} {J_{ij}^{i}}^T\ \Omega_{ij}\ J_{ij}^{j} \tag{5.6}$$

$$b_i \quad = \quad \sum_{\langle j,i\rangle \in \mathscr{C}} {J_{ij}^{i}}^T\ \Omega_{ij}\ (J_{ij}^{i}\hat{x}_i + J_{ij}^{j}\hat{x}_j) \tag{5.7}$$

The relaxed solution of equation i at step k is

$$x_i^{(k+1)} = A_{ii}^{-1}\left(b_i - \sum_{j<i} A_{ij}x_j^{(k+1)} - \sum_{j>i} A_{ij}x_j^{(k)}\right) \tag{5.8}$$

The estimated value of x_i is determined by the neighbor poses, either already updated ($j < i$) or not ($j > i$). These procedure is performed iteratively until solution is reached with enough precision. Since A is a symmetric positive defined matrix, the convergence of the algorithm is granted.

Gauss-Seidel relaxation is only the basic step of multi-level relaxation (MLR) algorithm. MLR defines a hierarchy between nodes to solve the problem at different levels of resolution in order to speed up the convergence. However, the Gauss-Seidel algorithm can be conveniently decomposed in separate tasks that are performed independently. In the next section, we describe a parallel version of Gauss-Seidel relaxation.

5.3 A Parallel Linear-Equation Solver

Identification of the parts of the algorithm that can be executed independently is the first step in making an algorithm parallel. The block-row update equation Eq. (5.8) of Gauss-Seidel depends on nodes x_j that are connected to the current node, i.e. the nodes with $A_{ij} \neq 0_{3\times3}$. The structure of the linearized information matrix A depends on the connectivity of the constraints network. Since the graph is built incrementally by one or more robots adding each pose in a trajectory or eventually closing loops, the resulting matrix is naturally sparse and locally connected. Hence, a decomposition of the network into clusters follows directly from the structure of the problem. Still the clusters are connected to each other: the computation of border nodes requires data from another cluster and the single algorithm tasks cannot be executed in parallel when such dependency exists.

An important remark concerns the role of order for node variables in equation Eq. (5.8). While the order does not change the value to which the algorithm converges, it determines the dependencies among variables. In particular, at a given iteration k the updated value of pose $x_i^{(k+1)}$ depends both on already updated poses, whose index is $j < i$, and on the poses yet to be updated ($j > i$). The value of the last ones is known before starting a new iteration, so their values and the value of pose i can be computed independently. Thus, in order to compute the nodes of a cluster independently from the nodes connected to the cluster and not belonging to it, the *contour nodes* have to be computed at the end.

The suggested reorder leads to the so called *block-bordered-diagonal form* (BBD) of a matrix [40]. Figure 5.1 shows how it is possible to reorder variables based on a cluster decomposition. Each simple cluster (labeled with $B1, B2, B3$) has no direct connection to other clusters, except than contour nodes (labeled with a, b, c) cluster.

By reordering nodes so that contour nodes are the last ones, the resulting information matrix assumes a block diagonal form with a border due to contour nodes. Each cluster can be solved independently by using the value of contour nodes at the previous step. It is therefore convenient that the contour partition be as small as possible in order to limit computation of the sequential part. In the following, we discuss how to find the clusters and the permutation that achieves BBD form.

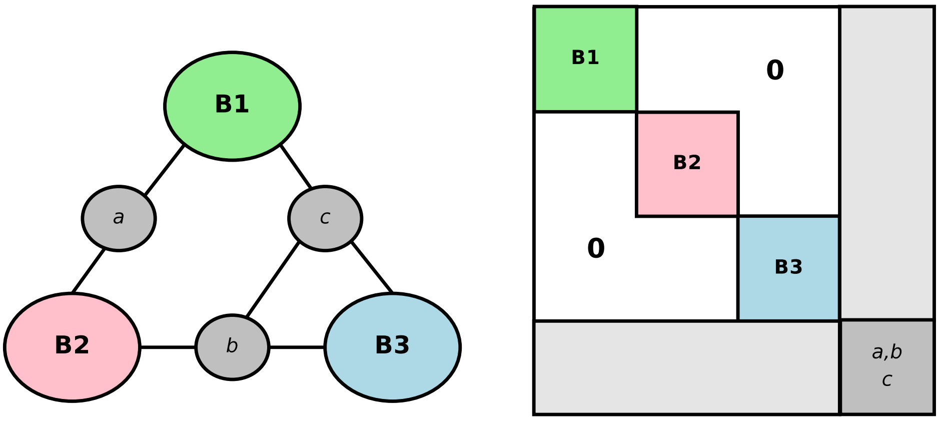


Figure 5.1: Decomposition of a graph in clusters of nodes (left) and information matrix in block-bordered-diagonal form after reordering (right). Note the position of contour nodes (a, b, c) in the matrix.

5.3.1 Clustering nodes and reordering variables

A clustering algorithm is needed to reorder the matrix into BBD form. There are two main requirements for this algorithm. First, the number of contour nodes should be limited as much as possible. This property is achieved when the clustering algorithm picks *bottleneck* nodes. Bottlenecks nodes correspond to the minima in the size of the contour when partition consists of sets of connected nodes. Therefore the size of the clusters could be chosen to balance the computational load of each task. We propose to meet these requirements by adopting a heuristic algorithm to perform node-tearing using a contour set [71]. Node-tearing methods decompose a network in smaller subnetworks that can be solved separately. Starting from a weakly connected node, the contour set is expanded putting the new nodes in a contour set. Bottleneck is detected by searching a minimum in the size of contour set. Heuristic techniques avoid local minima and bound the minimum and maximum size of a cluster to the interval $[perc\ n_{max}, n_{max}]$ $(0 < perc < 1)$.

Algorithm 5 illustrates how clustering is performed. Note that the condition for creating a new cluster is satisfied in two cases: when the contour is empty or when the candidate cluster set contains n_{max} nodes. Anyway, the final cluster set contains only the nodes found before the last detected bottleneck.

Each cluster is labeled with an integer, except for the contour nodes cluster that is labeled with ∞ (in real implementation, an integer larger than other labels is used). Thus, the numeric identifier of a cluster induces an order in the set of nodes. The identifiers of the simple clusters could be permuted without significant changes for parallel Gauss-Seidel relaxation. The order of the nodes belonging to the same cluster remains undefined and is chosen by implementation. Since the robot poses are usually numbered incrementally, it seems convenient to use their identifiers as second index of a lexicographic order criterion.

5.3.2 Parallel Implementation

The clustering algorithm identifies the independent groups of nodes that can be solved independently. Then, there is a set of tasks to be assigned to different threads or processes. A map estimator consisting of different processes would allow the execution of the algorithm on different hosts and could be applied in multi-robot contexts. The drawback of such a solution is the overhead due to the exchange of messages required at the end of each iteration. Nonetheless, the amount of exchanged data is limited to nodes of contour cluster.

In this chapter, the straightforward multi-threads solution is described. Algorithm 6 summarizes the mains step of the algorithm. A thread-pool, specifically a work crew, has been implemented in order to create and manage threads ready to accept tasks. This choice allows the creation of a correct number of threads depending on the available parallelism of the machine. After nodes reorder, the evaluation of

```
Data: $\mathscr{N}$: set of nodes, $n_{max}$: maximum size of cluster, $perc$: portion of $n_{max}$
        considered as minimum size of cluster.
Result: a label for each node
$\mathscr{I} = \{\}$;                              /* candidate cluster set */
$\mathscr{C} = \{\}$;                                /* current contour set */
$lastContourSize = 0$;
$iteratingSize = 0$;
foreach $n \in \mathscr{N}$ do
    $n.label = \{\}$;
end
while $\exists n \in \mathscr{N}$ : $n.label = \{\}$ do
    pick $n$, node with minimum degree in $\mathscr{N}$);
    while $\exists n \in \mathscr{C}$ : $n.label = \{\}$ do
        $\mathscr{I} = \mathscr{I} \cup \{n\}$;
        if $n \in \mathscr{C}$ then
            remove $n$ from $\mathscr{C}$;
        end
        foreach node $t$ adjacent to $n$ and $t.label = \{\}$ do
            $\mathscr{C} = \mathscr{C} \cup \{t\}$
        end
        if $card(\mathscr{I}) > perc\ n_{max}$ and $card(\mathscr{C}) < lastSize$ then
            $\mathscr{T} = \mathscr{C}$;                       /* save cutting set */
            $iteratingSize = card(\mathscr{I})$;
        end
        if $\mathscr{C} = \{\}$ then
            $iteratingSize = card(\mathscr{I})$;
        else

        end
        if $=\{\}$ or $card(\mathscr{I}) > n_{max}$ then
            foreach $t$ in first $iteratingSize$ of $\mathscr{I}$ do
                $t.label = newClusterLabel()$;
            end
            foreach node $t$ in cutting $\mathscr{T}$ do
                $t.label = \infty$;
            end
            $\mathscr{C} = \{\}$;
        end
        $lastSize = card(\mathscr{C})$;
        pick $n$ from $\mathscr{C}$, if exists;
    end
end
```

Algorithm 5: Partition of graph into clusters.

the poses of a cluster is inserted in the queue of tasks ready to be executed by an available thread. When the execution of all tasks is completed, the poses of contour nodes are updated. The Gauss-Seidel procedure within linearization is then repeated for an appropriate number of iterations.

Synchronization is required at the end of each iteration to update data for all threads. A development of this approach would consist in reducing the dependencies between the tasks. This could be achieved only with the adoption of a more hierarchical approach than simple Gauss-Seidel relaxation. The decomposition of the network suggests itself a hierarchy to be exploited in relaxation.

```
Data: $\mathscr{N}$: set of nodes, $\mathscr{E}$: set of constraints.
Result: the updated graph $(\mathscr{N},\mathscr{E})$
compute clusters;
reorder set $\mathscr{N}$ according to the following criterion:
$n_1 < n_2$ iff ($n_1.label < n_2.label$ or ($n_1.label == n_2.label$ and
$n_1.index < n_2.indes$));
initialize thread pool;
while algorithm not converged do
    foreach cluster $\mathscr{I}$ do
        assign cluster $\mathscr{I}$ to a thread of the pool;
                                        /* executed on thread */
        linearize constraints;
        perform Gauss-Seidel relaxation on constraints of $\mathscr{I}$;
    end
                                   /* thread synchronization */
    linearize constraints of countour set $\mathscr{C}$;
    perform Gauss-Seidel relaxation on constraints of $\mathscr{C}$;
end
```

Algorithm 6: Parallel Gauss-Seidel relaxation

5.4 Results

In this section, we evaluate the performance of the proposed parallel constraints solver. To test the algorithm we used the constraint networks extracted from the two datasets used in section 4.4: the ACES building at UT Austin campus (ACES), alreaady exploited in previous chapter, and the Intel Research Lab (INTEL) [36]. Constraints between pairs of poses have been obtained by matching laser scans and grid map patches. The resulting graph is the input for the experiments described in the following.

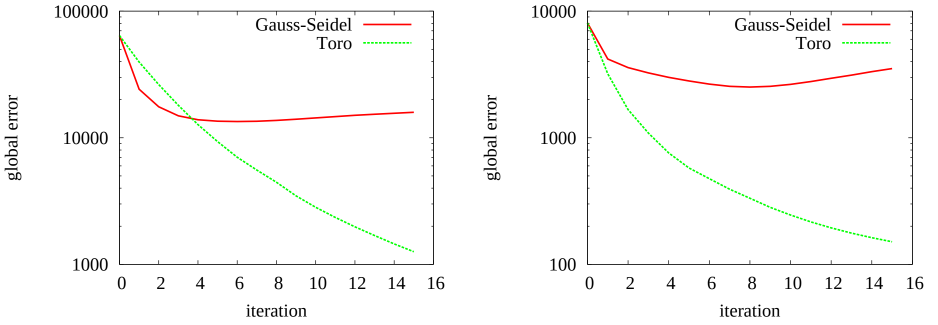


Figure 5.2: Global error χ^2 of Gauss-Seidel relaxation and stochastic gradient descent (Toro) after each iteration for datasets ACES (left) and INTEL (right).

First, the convergence rate of Gauss-Seidel relaxation has been evaluated to remark its advantages and drawbacks. In particular, we compared the relaxation method with the offline version of the tree network optimizer (Toro) [30]. Figure 5.2 depicts the global error of the network for the two algorithms at different iteration steps. After few iterations, the global error decreases faster with Toro than with Gauss-Seidel. This outcome was expected and confirms the results previously obtained in the comparison between Gauss-Seidel relaxation and stochastic gradient descent [66] with a new implementation.

Figure 5.3 shows the adjusted network for ACES. These experiments show that Gauss-Seidel relaxation is not the best algorithm to estimate a maximum likelihood map, but rather it is a simple method that can be easily adapted to a distributed con-

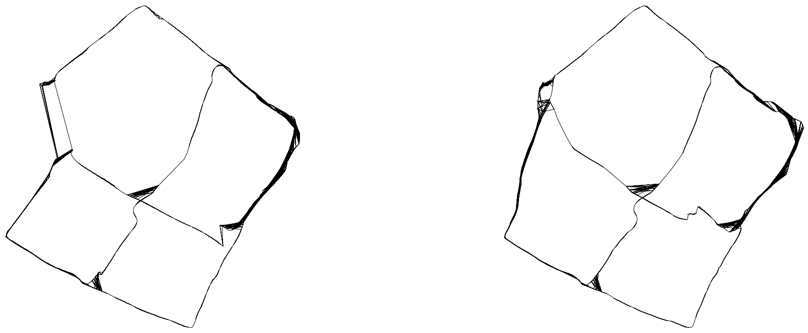


Figure 5.3: Network from ACES before the optimization (left) and after adjustment with Gauss-Seidel relaxation at iteration 5 (right). The effect of the adjustment is observable in the top left-most loop closure.

text.

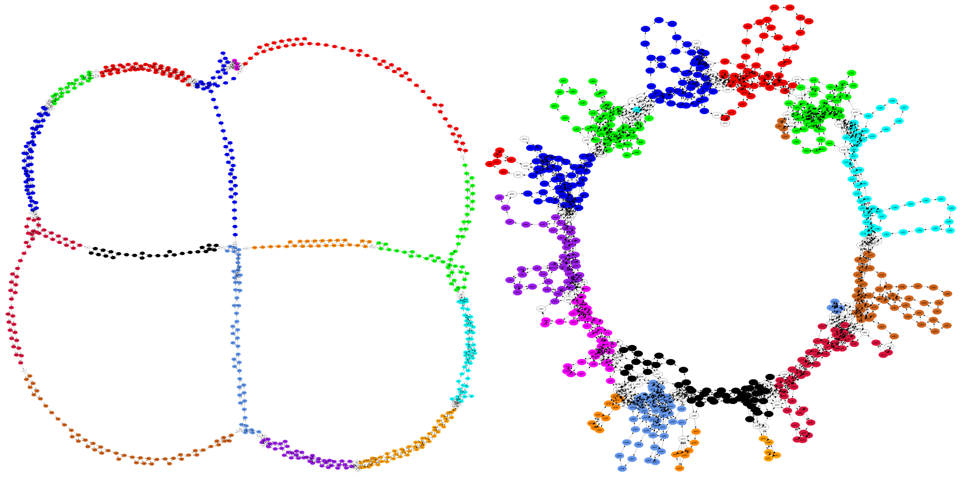


Figure 5.4: The clustered graph of datasets ACES (top) and INTEL (bottom). Clusters are identified by different colors and contour nodes appears as clear spots dividing clusters. The poses of nodes have been modified by the plotting program.

As pointed out in section 5.3, the main issue of a parallel algorithm is the choice of a suitable graph partitioning. In the proposed method, this operation is performed by a heuristic node-tearing technique. One of the main advantages of this approach is that it accepts bounds on the maximum size of a partition and on the preferred minimum size. Figure 5.4 shows the results of decomposition of graphs of ACES and INTEL with maximum size $n_{max} = 50$ and $perc = 0.6$. Table 5.1 reports data on the results of the clustering algorithm. The mean size of clusters and the size of contour partition depend on the topology of the constraints network. In particular, the ACES dataset has a smaller mean node degree and the number of contour nodes is limited.

Dataset	Nodes number	Mean size	Cluster number	Contour size
ACES	648	39.7	19	43
INTEL	729	30.2	14	106

Table 5.1: Results of clustering.

Despite the overhead there are advantages with the multi-thread implementation of the proposed algorithm. The system has been tested on an Intel Core 2 Quad Q9450 both with a sequential version of the algorithm and with a variable number of threads. Results in table 5.2 report the average time required to complete a Gauss-Seidel iteration for the ACES dataset. Node clustering has been formed with default parameter values $n_{max} = 50$ and $perc = 0.6$. With two threads the advantage of the multi-threaded version over the sequential one is remarkable. By further increasing the number of threads, the average time further decreases. It should be noted that no

Thread number	Time (ms)
1	17.668
2	5.659
3	4.507
4	4.316

Table 5.2: Average time of Gauss-Seidel relaxation for the ACES dataset.

advanced thread programming features have been exploited and thread priorities have not been modified. Hence, the parallel execution of tasks is not granted and relies on operating system scheduler. However, the improvement is significant and would have possibly been even more notable with a larger network.

5.5 Discussion

In this chapter, we have presented a parallel algorithm that estimates a map consisting of poses and constraints. The proposed method combines elementary Gauss-Seidel relaxation on linearized likelihood function and node clustering into partitions. Constraint network decomposition removes mutual dependencies among the node partitions, except for special nodes called contour nodes. Formally, this operation corresponds to a permutation of the variables of the network that transforms the information matrix in block-bordered diagonal form. Thus, a Gauss-Seidel iteration is decomposed in tasks to be executed in parallel.

A preliminary version of the algorithm in a multi-thread design exploiting commodity multi-core processors has been implemented. Experiments with two different datasets show the effectiveness of graph decomposition and a better exploitation of computational resources in map estimation. Although faster methods have been proposed, Gauss-Seidel relaxation has proven appropriate for a parallel design. Advantages of parallel design are clearer in multi-robot mapping and in general distributed contexts.

Chapter 6

Conclusion

This dissertation has presented a novel perspective of real-time feasibility in localization and mapping problems. Online localization and mapping methods update the estimation of robot location, environment representation, or both by integrating the incoming sensor information. Since observations are periodically captured by robot sensors, localization and mapping algorithms are constrained to complete the execution of an update before a new observation is available. In literature, several partial contributions have been presented, most of them focused on the reduction of computational complexity, but no comprehensive discussion of real-time feasibility had been previously proposed.

The reasons that make real-time feasibility difficult are different in the case of localization and of mapping problems. The execution time of a global localization method depends on the number of localization hypotheses tracked by the localizer before the convergence to the correct robot location. In mapping problems, the augmentation of map produces a global update, when the effect of integrated map elements is propagated to the whole existing map. Despite differences in the nature of each specific problem and in the solution methods, in this thesis we claim that a locality principle is a general design criterion for real-time or incremental execution of localization and mapping algorithms. The probabilistic robotics paradigm provides a unified formulation for the different problems and a conceptual framework for the application of the proposed criterion. Locality may be applied to perform temporal or spatial decomposition of the global estimation. Each subproblem collects the elements that are in the proximity of the event that changes the estimation. If the computation of update is locally bounded to this set, processing meets time constraints. The results of subproblems are then composed together to achieve the complete estimation, that in some cases may be delayed. The particular contributions of this thesis correspond to the application of the locality principle to specific problems.

The Real-Time Particle Filter is an advanced version of Particle Filter algorithm conceived to achieve a tradeoff between time constraints related to sensor management and filter accuracy depending on the number of samples. This goal is achieved by partitioning the overall samples required to obtain the required accuracy into sets, each of them corresponding to an observation. Such partitioning of samples is an example of temporal decomposition. The local estimations represented by partition sets are composed again when the resampling step is performed at the end of temporal estimation window. In the original version, the mixture posterior computed mixing samples from each partition suffers from a bias problem. In this thesis, we proposed two main contributions: first, an analysis of the efficiency of the resampling solution of the Real-Time Particle Filter through the concept of effective sample size; second, a method to compute the mixture weights that balances the the effective sample size of partition sets and is less prone to numerical instability.

The feasibility of time constraints is more difficult to achieve in mapping problems. Maximum Likelihood mapping is the most suitable approach for applying the locality principle thanks to the sparsity allowed by the full SLAM formulation and to the uniformity of the constraint network model. The technique adopted in this thesis combines stochastic gradient descent and incremental tree parameterization. This tree network optimizer exploits an efficient optimization technique and organizes the graph into a spanning tree structure suitable for a decomposition. In this thesis, the incremental version of the original algorithm has been adapted using again the locality principle. Local decomposition is achieved selecting the portion of the network perturbed by the addition of a new constraint. Furthermore, the perturbation of gradient descent iteration is limited for the region already converged by adapting the learning rate. Adapting learning rates have been already proposed in literature, and in this thesis this concept is applied to the tree parameterization. Finally, optimization is scheduled with an heuristic rule that controls the error increase in the constraint network. The algorithm converges faster than the previous version of the same algorithm and in several condition performs better than other state-of-the-art methods. Real-time feasibility is not granted since network update cannot always be locally bounded, e.g. when large loops involving a significant portion of spanning tree are closed. However, the proposed incremental tree network optimizer is suitable for online execution. The constraint solver has been integrated with a map builder that extracts the constraint network from a laser scan. This system uses a metric-topological hybrid map that associates a local map to the nodes of the network. The map builder further reduces the number of nodes by extracting an occupancy grid map from several observations.

The final contribution is a parallel maximum likelihood algorithm for robot mapping. Local decomposition, which has been exploited for the adaptation of maximum likelihood approach to online incremental mapping, allows also parallel decomposition. The proposed algorithm estimates the map iterating a linearization step and the solution of the linear system with Gauss-Seidel relaxation. Gauss-Seidel moves one pose at a time according to the local constraints and the neighbor poses, which can be already updated or not. Since this solution technique does not force the order of poses, a proper variable reorder allows decomposition of the general problem into independent parts. The proposed algorithm divides the network in connected clusters of local nodes. The reorder of network nodes induced by clusters transforms the linearized information matrix in block-border diagonal form. Each diagonal block of the matrix can then be solved independently. The border block of the matrix corresponds to the nodes that divide the clusters and represents the sequential part of the algorithm. The proposed parallel maximum likelihood algorithm can exploit the computation resources provided by commodity multi-core processor. Moreover, this solution can be applied to multi-robot mapping.

In conclusion, the contributions presented in this dissertation outline a novel perspective on real-time feasibility of robot localization and mapping methods, thus bringing these algorithmic techniques closer to applications.

Appendix A

Pose and Constraint Networks

A.1 Pose Compounding Operation

This section describes the commonly used notation of pose compounding introduced in [73]. This notation is used to represent transformations or relationships between planar poses. In the following pose $t = [t_x, t_y, t_\theta]$ represents the transformation from the pose $p_i = [p_{i,x}, p_{i,y}, p_{i,\theta}]$ to pose $p_j = [p_{j,x}, p_{j,y}, p_{j,\theta}]$. The compounding of pose p_i with pose t is usually denoted with

$$p_j \quad = \quad p_i \oplus t \tag{A.1}$$

Relationship between coordinates in pose compounding is defined as

$$\begin{bmatrix} p_{j,x} \\ p_{j,y} \\ p_{j,\theta} \end{bmatrix} = \begin{bmatrix} p_{i,x} + t_x \cos p_{i,\theta} - t_y \sin p_{i,\theta} \\ p_{i,y} + t_x \sin p_{i,\theta} + t_y \cos p_{i,\theta} \\ p_{i,\theta} + t_\theta \end{bmatrix} \tag{A.2}$$

Inverse compounding of pose p_j with p_i returns the relative transformation and is defined by

$$t \quad = \quad p_j \ominus p_i \tag{A.3}$$

The equations of inverse compounding can be easily derived from Eq. (A.2):

$$\begin{bmatrix} t_x \\ t_y \\ t_\theta \end{bmatrix} = \begin{bmatrix} (p_{j,x} - p_{i_x}) \, \cos p_{i,\theta} + (p_{j,y} - p_{i,y}) \, \sin p_{i,\theta} \\ -(p_{j,x} - p_{i_x}) \, \sin p_{i,\theta} + (p_{j,y} - p_{i,y}) \, \cos p_{i,\theta} \\ p_{j,\theta} - p_{i_\theta} \end{bmatrix} \tag{A.4}$$

In this thesis, constraints between poses i and j are expressed by relative transformation between the two poses. In particular, constraint $\langle i, j \rangle$ represents the observation of pose j from pose i, which is equivalent to $p_j \ominus p_i$. Since it is a stochastic constraint, $\langle i, j \rangle$ has an expected value δ_{ji} and an information matrix Ω_{ji} expressed as a relative transformation.

A.2 Pose Parameterization

In maximum likelihood methods for robot mapping the choice of parameters representing network configuration has a great importance. Parameters x_i are associated to the robot poses in a constraint network, but they are distinguished from poses. Three possible pose parameterizations have been proposed [66]. The three parameterizations are discussed in the following.

- *Global pose*. A global pose represents the value of robot pose with respect to global reference frame, i.e. $x_i = p_i$. This parameterization is the most commonly used one and has the advantage of a sparse Jacobian of constraints. In particular, the Jacobians of constraint $\langle i, j \rangle$ with respect to x_i and x_j are J_{ij}^i and J_{ij}^j respectively

$$J_{ij}^i = \begin{bmatrix} \cos x_{i_\theta} & \sin x_{i_\theta} & 0 \\ -\sin x_{i_\theta} & \cos x_{i_\theta} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

$$J_{ij}^j = \begin{bmatrix} -\cos x_{i_\theta} & -\sin x_{i_\theta} & -(x_{j_x} - x_{i_x}) \, \sin x_{i_\theta} + (x_{j_y} - x_{i_y}) \, \cos x_{i_\theta} \\ \sin x_{i_\theta} & -\cos x_{i_\theta} & -(x_{j_x} - x_{i_x}) \, \cos x_{i_\theta} - (x_{j_y} - x_{i_y}) \, \sin x_{i_\theta} \\ 0 & 0 & -1 \end{bmatrix} \tag{A.6}$$

The main drawback of global pose lies in its sensitivity to the adjacent constraints. Since the motion of the robot is cumulative, the modification of a single pose affects a large number of poses.

- *Relative pose*. A relative parameter x_i represents the relative transformation with respect to parameter x_{i-1}.

$$x_i = \begin{cases} p_0 \\ p_i \ominus p_{i-1} \end{cases} \tag{A.7}$$

  With this parameterization a modification of a single parameters affect the evaluation of several parameters, but the Jacobians are strongly nonlinear and dense.

- *Incremental pose*. Incremental parameterization is exploited in the maximum likelihood method based on stochastic gradient descent proposed by Olson *et al.* [66]. An incremental pose is defined by the difference between the current global pose and the previous one. The incremental tree parameterization uses the parent node of node i on spanning tree

$$x_i = \begin{cases} p_0 \\ p_i - p_{parent(i)} \end{cases} \tag{A.8}$$

  The resulting path from the root of spanning tree to the current node is shorter. Incremental parameterization is a sort of approximation of the global pose, but with a straightforward approximated expression of the Jacobians. The Jacobian J_{ji}^k of constraint $\langle i, j \rangle$ with respect to node k is the unit matrix $\mathscr{I}$, if the node k is on the path $\mathscr{P}_{ji}$ from i to j; otherwise the Jacobian is null. A more careful analysis [65] shows that for the node belonging to $\mathscr{P}_{ji}$ the Jacobian J_{ji}^k is better approximated by the rotation matrix associated to pose p_i. The discussion of incremental pose in [65] also illustrates that this representation introduces artifacts in the Jacobian.

A.3 Constraint Extraction from Laser Scans

In chapter 4 a mapping system using a metric-topological map representation is described. Several scans contribute to the creation of a map patch representing the local occupancy grid map. The local maps are then associated to a node of the network.

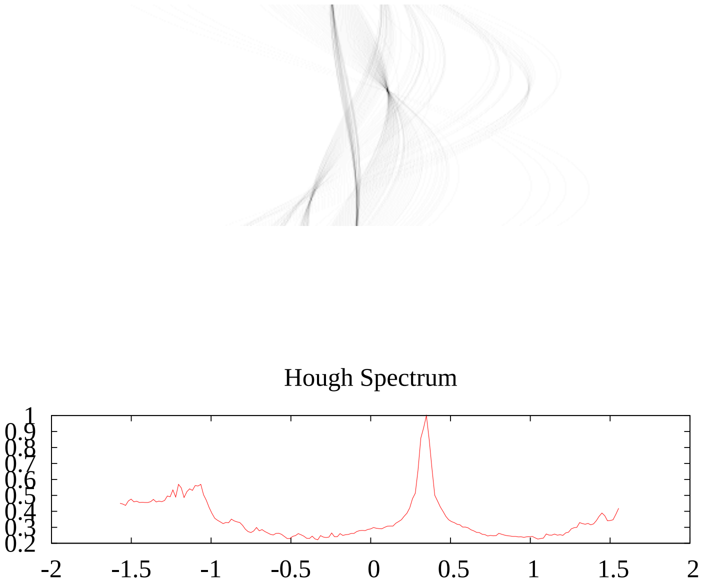


Figure A.1: An example of Hough transform computed on a laser scan (top) and the related Hough spectrum (bottom).

Spatial relationships between pairs of map reference frames correspond to the constraints of the network. Several operations are performed to achieve this result.

- *Scan Matching*. Scan matching is used to compute the local relationships between pairs of poses where the scans have been observed. The scan matcher used is a variant of the classic ICP proposed by Censi [9, 10].

- *Orientation*. In the adopted map occupancy grids are used to represent local maps. Data association is then performed with the correlation algorithm illustrated later. Since correlation is sensitive to orientation, the map patches are oriented according to the principal orientation of the scans. In indoor environment, walls and other architectural elements usually outline a major direction. This hypothesis of main orientation holds for the most common datasets of indoor environments. To extract such orientation the *Hough spectrum* has been exploited [11, 7]. Given the Hough transform $H(i_\rho, i_\theta)$ computed with the points of a laser scan, the Hough spectrum is computed as

  $$HS(i_\theta) = \sum_{i_\rho=0}^{n_\rho} H^2(i_\rho, i_\theta) \tag{A.9}$$

  Figure A.1 shows an example of Hough spectrum computed from a Hough plane.

- *Axis histograms*. In order to overlap two occupancy grid maps, that candidate occupancy grid maps are aligned along their principal orientation. or orthogonally. Both directions are investigated. The initial search of candidate translation vectors is performed by correlating axis histograms. An axis histogram counts the number of occupied cells on a specific direction. Figure A.2 shows an example of axis histogram for the x-axis.

- *Acceptance index*. An acceptance index is used to validate the computed translation vector similar to the one proposed by [7]. The two candidate maps are overlapped according to the candidate vector and the index is computed by counting the number of match and mismatch between the cells of the map.

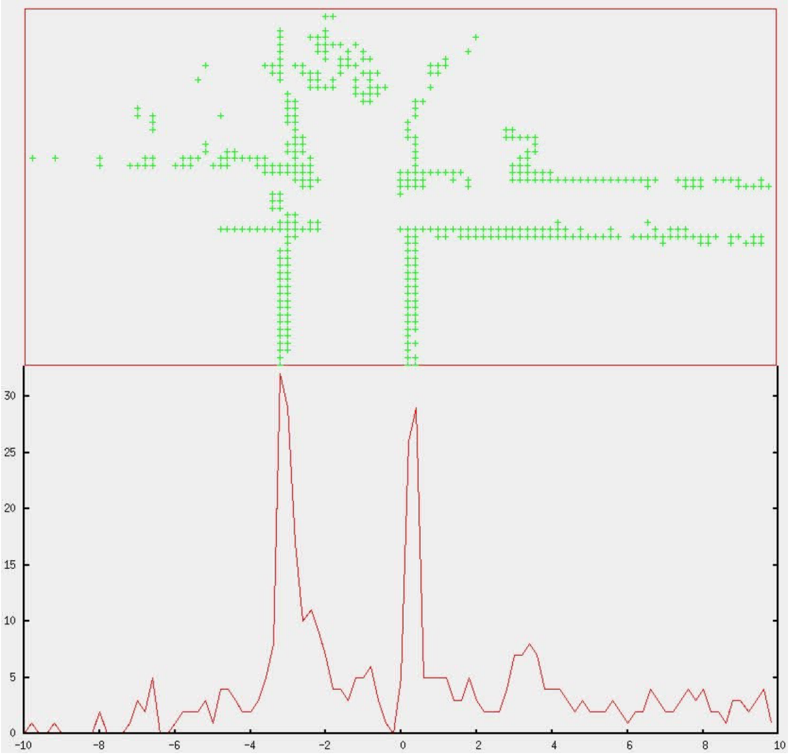


Figure A.2: An example of axis histogram of x-axis.

If the ratio between the number of matching cells and the total number of overlapping cells (excluding the cells with undefined values) is greater than a given threshold, then data association is validated. The chosen threshold value is equal to 0.5.

Bibliography

[1] L. Armesto, G. Ippoliti, S. Longhi, and J. Tornero. Probabilistic self-localization and mapping. *IEEE Robotics and Automation Magazine*, 15(2):77–88, June 2008.

[2] K. Arras, H. Castellanos, and R. Siegwart. Feature-based multi-hypothesis localization and tracking for mobile robots using geometric constraints. *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2:1371–1377, 2002.

[3] J. Bentley. Multidimensional binary search trees used for associative searching. *Com. ACM*, 18(9):509–517, September 1975.

[4] N. Bolic, N. Djuric, and N. Hong. New resampling algorithms for particle filters. In *Proc. of the IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2003.

[5] M. Bosse, P. Newman, J. Leonard, and S. Teller. An ALTAS framework for scalable mapping. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2003.

[6] W. Burgard, D. Fox, and S. Thrun. Active mobile robot localization. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, 1997.

[7] S. Carpin. Merging maps via Hough transform. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

[8] S. Caselli, F. Monica, and M. Reggiani. YARA: A software framework enhancing service robot dependability. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

[9] A. Censi. An accurate closed-form estimate of ICP's covariance. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[10] A. Censi. An icp variant using a point-to-line metric. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

[11] A. Censi, L. Iocchi, and G. Grisetti. Scan matching in the Hough domain. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2001.

[13] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[14] F. Dellaert and M. Kaess. Square root sam: Simultaneous location and mapping via square root information smoothing. *Int. Journal of Robotics Research*, 2006.

[15] F. Dellaert and P. Krauthausen. A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 1261–1266, 2005.

[16] A. Doucet. On sequential simulation-based methods for bayesian filtering. Technical report, Signal Processing Group, Dept. of Engeneering, University of Cambridge, 1998.

[17] A. Doucet, J. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[18] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Journal of Autonomous Robots*, 12(3):287 – 300, 2002.

[19] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping (SLAM): part i. *IEEE Robotics & Automation Magazine*, 13:99–110, June 2006.

[20] P. Elinas and J. Little. σMCL: Monte-Carlo localization for mobile robots with stereo vision. *Proc. of Robotics: Science and Systems (RSS)*, June 2005.

[21] R. Eustice, O. Pizarro, and H. Singh. Visually augmented navigation in an unstructured environment using a delayed-state history. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2004.

[22] R. Eustice, H. Singh, and J. Leonard. Exactly sparse delayed-state filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2428–2435, 2005.

[23] J. Folkesson and H. Christensen. Graphical SLAM-a self-correcting map. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 419–427, 2004.

[24] J. Folkesson and H. Christensen. Robust SLAM. In *Proc. of the IFAC Symp. on Intelligent Autonomous Vehicles (IAV)*, 2004.

[25] D. Fox. Adaptive real-time particle filters for robot localization. *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2:2836–2841, 2003.

[26] D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization: Efficient position estimation for mobile robots. *Proc. of the National Conference on Artificial Intelligence*, 1999.

[27] D. Fox, S. Thrun, W. Burgard, and F. Dellaert. *Sequential Monte Carlo Methods in Practice*, chapter Particle Filters for Mobile Robot Localization, pages 401–428. Springer, 2001.

[28] U. Frese. Treemap: An $O(logn)$ algorithm for indoor Simultaneous Localization and Mapping. *Journal of Autonomous Robots*, 21(2):103–122, 2006.

[29] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Transactions on Robotics*, 21(2):1–12, 2005.

[30] G. Grisetti, D. Lodi Rizzini, C. Stachniss, E. Olson, and W. Burgard. Online constraint network optimization for efficient maximum likelihood map learning. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2008.

[31] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[32] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proc. of Robotics: Science and Systems (RSS)*, Atlanta, GA, USA, 2007.

[33] G. Grisetti, D. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi. Fast and accurate SLAM with Rao-Blackwellized Particle Filters. *Journal of Robotics & Autonomous Systems*, 55:30–38, 2007.

[34] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE Int. Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, 1999.

[35] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[36] A. Howard and N. Roy. Radish: The robotics data set repository, standard data sets for the robotics community.

[37] S. Julier and J. Uhlmann. A counter example to theory of simultaneous localization and map building. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2001.

[38] S. Julier, J. Uhlmann, and H. Durrant-Whyte. A new approach for the nonlinear tranformation of means and and covariances in filters and estimators. *IEEE Transactions on Robotics*, 45(3):477–492, March 2000.

[39] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Fast incremental smoothing and mapping with efficient data association. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[40] D. P. Koester, S. Ranka, and G. C. Fox. A parallel Gauss-Seidel algorithm for sparse power system matrices. In *In SuperComputing '94*, pages 184–193, 1994.

[41] K. Konolige. Large-scale map-making. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 457–463, 2004.

[42] K. Konolige and K. Chou. Markov localization using correlation. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, 1997.

[43] C. Kwok, D. Fox, and M. Meilă. Real-time particle filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2003.

[44] C. Kwok, D. Fox, and M. Meilă. Real-time particle filters. *Proc. of the IEEE*, 92(3):469–484, March 2004.

[45] J. Leonard and H. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1991.

[46] J. Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6(2):113–119, June 1996.

[47] J. Liu and R. Chen. Blind deconvolution via sequential imputations. *Journal of the American Statistical Association*, 90(430):567–576, June 1995.

[48] J. Liu and R. Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, Sept. 1998.

[49] D. Lodi Rizzini and S. Caselli. Improved mixture representation in real-time particle filters for robot localization. In *Proc. of the European Conference on Mobile Robots (ECMR)*, 2007.

[50] D. Lodi Rizzini and S. Caselli. Map estimation with laser scan based on incremental tree network optimizer. In *Proc. of the Int. Conf. on Computational Intelligence, Robotics and Autonomous Systems (CIRAS)*, 2008.

[51] D. Lodi Rizzini and S. Caselli. Metric-topological map from laser scans adjusted with incremental tree network optimizer. *Proc. of the Int. Conf. on Computational Intelligence, Robotics and Autonomous Systems (CIRAS)*, page To appear, 2008.

[52] D. Lodi Rizzini and S. Caselli. *Robotics, Automation and Control*, chapter An Improved Real-Time Particle Filter for Robot Localization, pages 417–434. I-Tech Education and Publishing KG, 2008.

[53] D. Lodi Rizzini, F. Monica, M. Reggiani, and S. Caselli. Addressing complexity issues in a real-time particle filter for robot localization. In *Proc. of the Intl. Conf. on Informatics in Control, Automation and Robotics (ICINCO)*, 2007.

[54] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. In *IEEE Computer Vision and Pattern Recognition Conference (CVPR)*, pages 935–938, 1994.

[55] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Journal of Autonomous Robots*, 4, 1997.

[56] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 1998.

[57] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equations of state calculations by fast computing machines. *Journal Chemical Physics*, 21:1087–1091, 1953.

[58] M. Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association*. PhD thesis, Carnegie Mellon University, 2003.

[59] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2003.

[60] M. Montemerlo and S. Thrun. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. In *Proc. of Int. Sym. on Experimental Robotics (ISER)*, 2004.

[61] M. Montemerlo and S. Thrun. Large-scale robotic 3-D mapping of urban structures. In *Proc. of Int. Sym. on Experimental Robotics (ISER)*, 2004.

[62] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[63] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 1151–1156, Acapulco, Mexico, 2003.

[64] P. Newman. *On Structure and Solution of Simultaneous Localization and Map Building Problem*. PhD thesis, University of Sidney, 2000.

[65] E. Olson. *Robust and Efficient Robotic Mapping*. PhD thesis, Massachusetts Institute of Technology, 2008.

[66] E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2262–2269, 2006.

[67] E. Olson, J. Leonard, and S. Teller. Spatially-adaptive learning rates for online incremental SLAM. In *Proc. of Robotics: Science and Systems (RSS)*, Atlanta, GA, USA, 2007.

[68] M. Paskin. Thin junction tree filters for simultaneous localization and mapping. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003.

[69] P. Pfaff, C. Stachniss, C. Plagemann, and W. Burgard. Efficiently learning high-dimensional observation models for monte-carlo localization using gaussian mixtures. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2008.

[70] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[71] A. Sangiovanni-Vincentelli, L. Chen, and L. Chua. An efficient heuristic cluster algorithm for tearing large scale networks. *IEEE Transactions on Circuits and Systems*, 24(12):709–717, Dec 1977.

[72] S. Se, D. Lowe, and J. Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmark. *Int. Journal of Robotics Research*, 21(8):735–758, August 2002.

[73] R. Smith and P. Cheeseman. On the representation and estimation of spacial uncertainty. *Int. Journal of Robotics Research*, 5(4):56–68, 1986.

[74] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[75] J. Weiss and W. Freeman. Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural Computation*, 13(10):2173–2200, 2001.

Acknowledgements

I have to thank many people for the professional and human gifts received during these three years. First of all, I would like to thank my advisor Prof. Stefano Caselli. He gave me the opportunity of attending my PhD study program in his laboratory and guided me with his ideas and the brainstorming sections during these years. This thesis is also the result of his comments. The reader will judge how much I have exploited his suggestions.

My thanks to my colleagues or ex-colleagues of our group, and in particular to Jacopo Aleotti, Francesco Monica, Eleonora Fantini and Monica Reggiani. I would like to thank Jacopo Aleotti for the profitable discussions: it is always positive to exchange opinions with a researcher and a friend working on a different research topic. Francesco Monica helped to correct my theoretical nature with his experience with robotic platforms. I would like to thank Monica Reggiani for giving me the book *Probabilistic Robotics* just before the discussion of my master thesis: this present influenced my future research.

During my second year I visited the Autonomous and Intelligent Systems group of Prof. Wolfram Burgard at the Albert-Ludwigs-Universitaet of Freiburg. I am grateful to Prof. Burgard for this opportunity and to his group for the support. I learnt a lot during those six months. I want to thank in particular Giorgio Grisetti for the profitable discussions, the collaboration and for providing me the code of the tree network optimizer. He has the gift of resuming complex problems with few and rather straightforward assertions. I want to thank also Cyrill Stachniss for the collaboration during my visit. My thanks to all the members of the group of Freiburg: I don't list them because I am afraid of forgetting somebody. I am also grateful to Edwin Olson for his interesting opinions on his stochastic gradient descent. Additionally, I would thank those people that shared their code and experience with the Robotics community. Many of them are anonymous like the providers of datasets. I would like to thank Andrea Censi for providing his scan matcher.

I cannot forget the people from other groups of the Dipartimento di Ingegneria dell'Informazione of the Università degli Studi di Parma. In particular, I want to remember my current office mates Oscar Gerelli and Michele Amoretti, Stefano Ghidoni, Maria Chiara Laghi, Luca Consolini, Corrado Guarino Lo Bianco and Paolo Federici for his technical help, but the list is much longer. I am also grateful to Giovanni Capra, Bruno Ferrarini, Luca Domenichini, Gionata Boccalini, Nicola Bova, Giorgia La Rosa and Gabriele Camellini for their collaboration, while writing their thesis or doing their project of the Robotics Course.

Last but not least, I would like to thank my family members for their continous support.