Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática

Universidade do Porto Faculdade de Ciências

Universidade do Minho Departamento de Informática

2018

Eurico Farinha Pedrosa

Localização e Mapeamento Eficiente para Robótica: Algoritmos e Ferramentas

Efficient Localization and Mapping for Robotics: Algorithms and Tools

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática

Universidade do Porto Faculdade de Ciências

Universidade do Minho Departamento de Informática

2018

Eurico Farinha Pedrosa

Localização e Mapeamento Eficiente para Robótica: Algoritmos e Ferramentas

Efficient Localization and Mapping for Robotics: Algorithms and Tools

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Doutor em Informática, realizada sob a orientação científica de José Nuno Panelas Nunes Lau e Artur José Carneiro Pereira, Professores Auxiliares do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Prof. Doutor João Carlos Matias Celestino Gomes da Rocha
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Urbano José Carreira Nunes
Professor Catedrático da Universidade de Coimbra

Prof. Doutor Rodrigo Martins de Matos Ventura
Professor Auxiliar da Universidade de Lisboa

Prof. Doutor Gabriel de Sousa Torcato David
Professor Associado da Universidade do Porto

Prof. Doutor Vítor Manuel Ferreira dos Santos
Professor Associado da Universidade de Aveiro

Prof. Doutor José Nuno Panelas Nunes Lau
Professor Auxiliar da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

Quero começar por agradecer ao Professor Nuno Lau e ao Professor Artur Pereira pelos vários anos de orientação e pela confiança em mim depositada. Foi simplesmente um prazer.

Aos meus pais e ao meu irmão por todo o apoio oferecido.

A todos os membros do laboratório de robótica do grupo IRIS pela camaradagem e boas discussões ao longo destes vários anos. Em particular, quero agradecer ao João Cunha pelas discussões científicas, conversas de café e entreajuda, não só durante os vários projetos em que participamos em conjunto, mas também durante o desenvolvimento das nossas teses de doutoramento.

À Mariana por todo o apoio, carinho e muita paciência.

palavras-chave informática, robótica, localização, mapeamento, slam, algoritmos

resumo

Um dos problemas fundamentais em robótica é a capacidade de estimar a pose de um robô móvel relativamente ao seu ambiente. Este problema é conhecido como localização robótica e a sua exatidão e eficiência têm um impacto direto em todos os sistemas que dependem da localização. Nesta tese, abordamos o problema da localização propondo um algoritmo baseado em *scan matching* com otimização robusta de mínimos quadrados não lineares em *manifold* com a utilização de um campo de verosimilhança contínuo como modelo de perceção. Esta solução oferece uma melhoria percetível na eficiência computacional sem perda de exatidão.

Associado à localização está o problema de criar uma representação geométrica (ou mapa) do meio ambiente recorrendo às medidas disponíveis, um problema conhecido como mapeamento. No mapeamento a representação geométrica mais popular é a grelha volumétrica que discretiza o espaço em volumes cúbicos de igual tamanho. A implementação direta de uma grelha volumétrica oferece acesso direto e rápido aos dados mas requer uma quantidade substancial de memória. Portanto, propõe-se uma estrutura de dados híbrida, com divisão esparsa do espaço combinada com uma subdivisão densa do espaço que oferece tempos de acesso eficientes com alocações de memória reduzidas. Além disso, também oferece um mecanismo integrado de compressão de dados para reduzir ainda mais o uso de memória e uma estrutura de partilha de dados implícita que duplica dados, de forma eficiente, quando necessário recorrendo a uma estratégia *copy-on-write*. A implementação da solução descrita é disponibilizada na forma de uma biblioteca de software que oferece um *framework* para a criação de modelos baseados em grelhas volumétricas, e.g. grelhas de ocupação. Como existe uma separação entre o modelo e a gestão de espaço, todas as funcionalidades da abordagem esparsa-densa estão disponíveis para qualquer modelo implementado com o *framework*.

O processo de mapeamento é um problema complexo considerando que localização e mapeamento são resolvidos simultaneamente. Este problema, conhecido como localização e mapeamento simultâneo (SLAM), tem tendência a de consumir recursos consideráveis à medida que a exigência na qualidade do mapeamento aumenta. De modo a contribuir para o aumento da eficiência, esta tese apresenta duas solução de SLAM. Na primeira abordagem, o algoritmo de localização é adaptado ao mapeamento incremental que, em combinação com o *framework* esparso-denso, oferece uma solução de SLAM *online* computacionalmente eficiente. O resultados obtidos são comparados com outras soluções disponíveis na literatura recorrendo a um *benchmark* de SLAM. Os resultados obtidos demonstram que a nossa solução oferece uma boa eficiência sem comprometer a exatidão. A segunda abordagem combina o nosso SLAM *online* com um filtro de partículas *Rao-Blackwellized* para propor uma solução de *full* SLAM com um grau elevado de eficiência computacional. A solução inclui propostas de distribuição melhorada com refinamento de pose através de *scan matching*, re-amostragem adaptativa com pesos de amostragem suavizados, partilha eficiente de dados entre partículas da mesma geração e suporte para *multi-threading*.

keywords informatics, robotics, localization, mapping, slam, algorithms

abstract

One of the most basic perception problems in robotics is the ability to estimate the pose of a mobile robot relative to the environment. This problem is known as mobile robot localization and its accuracy and efficiency has a direct impact in all systems than depend on localization. In this thesis, we address the localization problem by proposing an algorithm based on scan matching with robust non-linear least squares optimization on a manifold that relies on a continuous likelihood field as measurement model. This solution offers a noticeable improvement in computational efficiency without losing accuracy.

Associated with localization is the problem of creating the geometric representation (or map) of the environment using the available measurements, a problem known as mapping. In mapping, the most popular geometric representation is the volumetric grid that quantizes space into cubic volumes of equal size. The regular volumetric grid implementation offers direct and fast access to data but requires a substantial amount of allocated memory. Therefore, in this thesis, we propose a hybrid data structure with sparse division of space combined with dense subdivision of space that offers efficient access times with reduced memory allocation. Additionally, it offers an online data compression mechanism to further reduce memory usage and an implicit data sharing structure that efficiently duplicates data when needed using a thread safe copy-on-write strategy. The implementation of the solution is available as a software library that provides a framework to create models based on volumetric grids, e.g. occupancy grids. The separation between the model and space management makes all features of the sparse-dense approach available to every model implemented with the framework.

The process of mapping is a complex problem, considering that localization and mapping have to be solved simultaneously. This problem, known as simultaneous localization and mapping (SLAM), has the tendency to consume considerable resources as the mapping quality requirements increase. As an effort to increase the efficiency of SLAM, this thesis presents two SLAM solutions. The first proposal adapts our localization algorithm to incremental mapping that, in combination with the sparse-dense framework, provides a computationally efficient *online* SLAM solution. Using a SLAM benchmark, the obtained results are compared with other solutions found in the literature. The comparison shows that our solution provides good efficiency without compromising accuracy. The second approach combines our *online* SLAM with a Rao-Blackwellized particle filter to propose a highly computationally efficient *full* SLAM solution. It includes an improved proposal distribution with scan matching pose refinement, adaptive resampling with smoothed importance weight, efficient sharing of data between sibling particles and multithreading support.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Robots play an important role in our lives, even if we are not aware of it. They are used in a variety of tasks that can be repetitive, heavy, of high precision, and even dangerous. Most robots are fixed with pre-programmed actions in a controlled environment. But on the other hand, mobile robots have the capability to move in their environment with some level of autonomy. Autonomous mobile robots have the ability to make decisions according to what they perceive from the environment and their goals.

One of the most basic perception problem in robotics is the capability to determine the pose of the robot relative to the given map of the environment (Cox, 1991). This problem is commonly known as **mobile robot localization**. Given the necessary representation (or map) of the environment the autonomous robot equipped the with appropriate sensor(s) should be able to infer its pose. Unfortunately, no sensor can provide error free measurements, making a single measurement usually insufficient to determine the correct pose of the robot. Instead, the robot's pose is integrated over time considering all relevant sources of information.

Early mobile robot localization solutions are based on the extended kalman filter for state estimation (e.g. Dickmanns and Graefe (1988)), that relies on feature map as representation of the environment. However, such representation usually requires modifications to the environment, such as the installation of artificial beacons. Additionally, proper identification and correspondence of perceived features in the environment with the actual features in the map is another research area. As an alternative, researchers moved from feature representation to geometric techniques for localization (e.g. Cox (1991)).

The concept of using scan matching as a geometric approach to localization was popularized by Lu and Milios (1997a). They utilize range scans, which capture the geometric representation of the "natural" features in the environment (e.g. walls, furniture), to find the rigid transformation between two consecutive matching scans. Another approach is to match the current scan with a map (e.g. Lauer et al. (2006)) that provides better results. Such approach utilizes an occupancy grid map as a geometric representation of the environment. Localization algorithms based on scan matching are typically fast and fairly accurate within an acceptable uncertainty. Beyond that acceptable uncertainty, localization algorithm based on scan matching can easily fail and never recover. The *Monte Carlo* localization algorithm (Fox, 2003) is an alternative that utilizes the same occupancy grid map as scan matching but has a higher resilience to uncertainty, although with lower computational efficiency and disputable accuracy. Nowadays, *Monte Carlo* localization is one of the most popular localization algorithms in robotics.

A truly autonomous robot should be able to build its map from scratch using only its sensors and mobile capabilities. Generating the map of the environment from the robot's sensor data is a problem known as **mapping**. If a ground truth of the robot's pose is provided at the moment a measurement is obtained the mapping problem is simplified to a problem of integrating measurements into a map representation, i.e. mapping with known poses. A popular example of mapping with known poses is the probabilistic occupancy grid mapping (Moravec, 1989, 1996). This method addresses the issue of integrating measurements with uncertainty on a volumetric grid. The dimensionality complexity of the volumetric grid is handled by its resolution, the lower the resolution the lower the space requirements. But on applications where a finer resolution is required, the necessary amount of space can easily outgrow the available memory. With the generalized availability of 3D sensors, space efficiency in mapping has become a pressing issue. The most successful approach was presented by Hornung et al. (2013) that proposes OctoMap, a mapping framework based on an octree data structure to manage space. The approach is oriented towards probabilistic occupancy grid maps and provides a considerable reduction in memory usage with an acceptable computational efficiency.

The problem of mapping increases in complexity when the poses of the robot are not known. Without having access to an external ground truth, the autonomous robot has to localize itself in the environment while building the very same map it uses for localization. This problem is known as **simultaneous localization and mapping** (SLAM). When faced with the challenge of geometric SLAM there are several issues to consider: the size of the environment, the uncertainty in measurements, the ambiguity in perception and the existence of loops in the environment. Several SLAM solution have been proposed over the years to address the aforementioned challenges, but not necessarily all of them because not all environments are made equal. For environments with small loops and reduced uncertainty, a SLAM solution based in scan matching can suffice (e.g. Kohlbrecher et al. (2011)), with the additional advantage of a lower computational cost. A more complex environment also requires a more complex solution. A good example is the grid-based Rao-Backwellized particle filter (RBPF) SLAM solution popularized by Grisetti et al. (2005), that is capable of handing environment of large size and uncertainty with several loops. To improve its efficiency it uses scan matching to obtain better samples and reduce the number of particles. An adaptive resampling technique is employed to maintain a diverse set of particles. Each particle contains its own map and mapping is handled by a probabilistic occupancy grid map. Despite the advances in efficiency introduced by the above RBPF SLAM solution, it can still nowadays be computationally heavy.

1.1 Thesis Statement and Objectives

In the current state of autonomous mobile robots, localization and mapping are still basic requirements to provide true autonomous capabilities. Being part of the base, their quality is reflected in all systems that depend on them, such as path planning, object recognition or location based tasks. The current state-of-the-art in localization (e.g. Monte Carlo), mapping (e.g. OctoMap) and SLAM (e.g. RBPF) are the main contributors for that quality. In this thesis we define quality in terms of accuracy, computational efficiency and space efficiency. The statement of this thesis is that the quality of localization, mapping and SLAM can be further improved to reduce their share of computation resources while offering higher accuracy. Consequently, the purpose of this thesis is to provide improved algorithms and tools

for localization and mapping that offers better efficiency than the current state-of-the-art while providing similar or better accuracy.

With respect to mobile robot localization, the objective of this thesis is to research scan matching algorithms with improved optimization, capable of handling higher uncertainty while retaining (or even improving) their low computational complexity. From a mapping point-of-view, the objective of this thesis is to explore a method to manage data allocation and access of a volumetric grid map that offers a fast data access similar to the regular grid map and the compact data allocation of a sparse data structure. Furthermore, it should be able to support concurrent access to data. A final objective is to use the algorithms and tools developed for localization and mapping as part of SLAM solutions with improved efficiency that can take advantage of the ubiquitous existence of computational systems with multiple threads.

1.2 Contributions

Mobile robot localization and mapping still presents several challenges in terms of accuracy and efficiency. The objective of this thesis is to advance the state-of-the-art by introducing improved algorithms and tools to increase the accuracy and efficiency of mobile robot localization and mapping. Those contributions are:

1. **Accurate and Lightweight Scan Matching Localization:** Our localization algorithm, based on scan matching, offers an accurate solution with minimal computational effort. It is built on top of maximum likelihood pose estimation that, with the introduction of a likelihood field as measurement model, provides a formulation that can be solved with non-linear least squares optimization on a manifold. The manifold is used to further linearize our formulation and simplify the calculation of the Jacobians. To improve its accuracy, the discrete nature of the likelihood field is minimized by interpolating the value of each grid cell with its neighbors. It actively handles outliers in measurements by using a loss function that controls their impact during optimization.

   (a) **Global Localization:** Localization algorithms based on scan matching are local search methods that are not capable of solving the global localization problem. To address this issue we propose a global localization procedure based on uniform sampling to find the best global pose when such is unknown.

2. **Sparse-Dense Approach for Grid Mapping:** To improve the efficiency of any mapping procedure, this thesis proposes a space management method for grid mapping that is based on sparse division of space combined with dense subdivision. Based on the concept of local density perception in a sparse environment, it provides the advantage of sparse space management (including automatic grid size growth) with the benefits of fast local data access times.

   (a) **Online Data Compression:** The local density of data provides the opportunity to increase space efficiency by exploiting statistical redundancy present in the data. A lossless data compression algorithm is used seamlessly and transparently during mapping to proactively reduce the data size.

   (b) **Implicit Data Sharing:** Grid maps are efficiently duplicated by using a Copy-On-Write method that defers the copy of data until a write operation is performed.

(c) **Mapping Framework with Multithreading Support:** The implementation of the sparse-dense approach, in the form of a software library, offers a mapping framework with the proposed space management, online data compression and implicit data sharing with multithreading support. All features are automatically available to any model that is implemented on top of this framework due to the architectural abstract layer that separates the model from space management.

3. **Real-time *online* SLAM:** Built on top of our scan matching based localization algorithm and grid mapping with the sparse-dense approach, we take *online* SLAM a step further in computational efficiency, while retaining good accuracy. Our solution is capable of mapping the environment at higher rate than most modern range sensors (i.e. LIDAR) can provide measurements.

4. **Improved Rao-Blackwellized Particle Filter SLAM:** For environments where a *online* SLAM solution is not adequate an improved Rao-Blackwellized particle filter SLAM is proposed. By using our scan matching and sparse-dense space management we reduced the number of necessary particles to obtain a topologically correct map while significantly improving the computational efficiency. When compared with the current state-of-the-art, we were able to obtain speedups that range from 6 to 25 times faster.

   (a) **Multithreaded RBPF SLAM:** By employing a *Thread Pool* model we were able to further improve the computational efficiency of our solution with multithreading. Thanks to the multithreading support of the sparse-dense framework both the scan matching and mapping procedure can leverage the multiple execution threads.

1.3 Publications

From this work the following publication have been produced:

- **Eurico Pedrosa**, Pereira, A., and Lau, N. (April 2018). A Sparse-Dense Approach for Efficient Grid Mapping. In *Autonomous Robot Systems and Competitions (ICARSC), 2018 International Conference on*, Torres Vedras, Portugal. IEEE

- **Eurico Pedrosa**, Pereira, A., and Lau, N. (2017). A Non-Linear Least Squares Approach to SLAM using a Dynamic Likelihood Field. *Journal of Intelligent & Robotic Systems*

- **Pedrosa**, **Eurico**., Pereira, A., and Lau, N. (April 2017). Efficient Localization based on Scan Matching with a Continuous Likelihood Field. In *Autonomous Robot Systems and Competitions (ICARSC), 2017 International Conference on*, Coimbra, Portugal. IEEE

- **Pedrosa**, **Eurico**., Pereira, A., and Lau, N. (May 2016). A Scan Matching Approach to SLAM with a Dynamic Likelihood Field. In *Autonomous Robot Systems and Competitions (ICARSC), 2016 International Conference on*, pages 35–40, Bragança, Portugal. IEEE

- **Eurico Pedrosa**, Lau, N., Pereira, A., and Cunha, B. (2015). A skill-based architecture for pick and place manipulation tasks. In Pereira, F., Machado, P., Costa, E., and Cardoso, A., editors, *Progress in Artificial Intelligence*, volume 9273 of *Lecture Notes in Computer Science*, pages 457–468. Springer International Publishing

- **Eurico Pedrosa**, Lau, N., and Pereira, A. (2013). Online SLAM Based on a Fast Scan-Matching Algorithm. In Correia, L., Reis, L., and Cascalho, J., editors, *Progress in Artificial Intelligence*, volume 8154 of *Lecture Notes in Computer Science*, pages 295–306. Springer Berlin Heidelberg

1.4 Thesis Structure

This thesis is organized in six chapters that describes the presented contributions. The current chapter, **Chapter 1**, introduced to the reader the motivation that guided the developed work and consequent contributions to the state-of-the-art. The chapter that follows, **Chapter 2**, presents the main concepts and solutions for mobile robot localization and mapping and the challenging problem of solving both concepts simultaneously, commonly known as simultaneous localization and mapping.

Our first contribution is presented in **Chapter 3**. It proposes the development of a localization algorithm for mobile robots based on scan matching. The localization problem is presented as maximum likelihood pose estimation that, with the introduction of the likelihood field as a measurement model, can be solved as a scan matching problem with non-linear least squares optimization on a manifold. The presentation of our contributions continues in **Chapter 4** with the proposal of a sparse-dense framework for efficient mapping. It is a hybrid solution that uses sparse division of space combined with a dense subdivision of space that is capable of covering any mapped area. Additionally, it provides several tools to further improve space efficiency such as online data compression and implicit data sharing. The combined knowledge of the previous two chapters is explored in **Chapter 5** with the development of two grid-based SLAM solutions. One solution tackles the *online* SLAM problem with the already proposed scan matching algorithm for localization that is adapted for incremental mapping. The other solution addresses the *full* SLAM problem with a Rao-Blackwellized particle filter SLAM that leverages on our scan matching algorithm and sparse-dense framework to improve its computational and space efficiency, which includes multithreading. Finally, **Chapter 6** draws the main conclusions of this thesis and discusses open questions and directions for future work.

Chapter 2

Localization and Mapping

In the majority of robotic application the decision of what to do is facilitated by knowing certain *quantities*. For example, it is simpler for a robot to move in the environment when it knows its exact location and the exact position of known objects. However, these variables are not directly measurable, instead, the robot has to rely on information extracted from sensor data. But sensors are a major bottleneck in state estimation, as the information they carry is usually incomplete about those quantities, and their measurements are contaminated by noise. Nonetheless, state estimation tries to recover state variables using the available sensor data and probabilities are used in state estimation algorithms to compute a belief distribution over possible states. Localization and mapping are two problems in robotics that can be solved using probabilistic state estimation algorithms. The objective of this chapter is to present the concepts of robotic localization and mapping in a probabilistic framework with explanatory examples that demonstrate how they can be solved. Additional state-of-the art will be presented in the remaining chapters when deemed necessary.

2.1 Bayesian State Estimation

A *state* is what characterizes an environment and is the collection of all aspects of the robot and environment that can influence the future. Some state variables are prone to change over time, such as the location and orientation of the robot robot, while others will remain unchanged (static), like the location of walls in a building. State that changes over time is named *dynamic state* and its counterpart is called *static state*. Let state be denoted by $\mathbf{x}$, with the understanding that the variables contained in $\mathbf{x}$ will depend on the context. The state at time t is denoted by $\mathbf{x}_t$. Time is assumed to be discrete, therefore all events take place at discrete time steps $t = 0, 1, 2, \ldots, n$. The point in time where the robot starts corresponds to $t = 0$. The number of state variables are potentially endless and, as already mentioned, it depend on the application.

State $\mathbf{x}_t$ follows the Markov property of a stochastic process. It means that the knowledge of past states, measurements, or controls provide *no* additional information that would help estimate future states. It does not require the future to be a deterministic function of the state, it may be stochastic but with the restriction that no variable prior to $\mathbf{x}_t$ may influence the stochastic evolution of future states, unless mediated through the state $\mathbf{x}_t$. Temporal processes where this restrictions apply are usually known as *Markov chains*. It is true that in practice it is not possible to specify a complete state for any reasonable robot system. A complete

state includes all aspects of the environment that may have an impact in the future, the robot itself with its limited computer memory and processing power, the surrounding people, etc., therefore, practical implementations only select a relevant subset of all state variables. Such state is designated *incomplete state*.

For most robotic applications, the state $\mathbf{x}_t$ is continuous. The robot pose, that is, the location and orientation of the robot relative to the global coordinate frame is a good example of a continuous state. But a state can also be discrete, as for example the binary state variable that models whether or not a door is open. State spaces that comprise both continuous and discrete variables are designated *hybrid* state spaces.

2.1.1 Environment Interaction

There are two types of interaction between a robot and its environment: it can move itself and objects in the environment and it can collect information about the environment state with its sensors. Both interactions can happen simultaneously, but to simplify their description they are presented separately.

Control The motion of the robot and manipulation of objects are examples of kinematics as stochastic systems. They change the state of the environment, even if the robot does not performs any action itself the state may change. Hence, it is assumed that the robot *always* performs an action, even when any type of motion is not existent. The data of a control action contains information about the change of the state in the environment. Velocity is an example of control data in mobile robotics. Applying a velocity of $1m$ per second during 2 seconds suggests that the pose of the robot, after executing the control action, is around $2m$ ahead of its pose before the execution of the command. Because of that, control data carry information about the change of state. Another source of control data are *odometers*. This sensor, the odometer, can measure the revolution of the motion wheels installed in the robot. Hence, it holds information about the change of the state. It is true that an odometer is a sensor, and as such its data can be seen as measurement data. But because it measures the effect of a control action it can be used as control data as well. Let us denote u as the control data. Variable u_t denotes the change of state in the time interval $(t-1;t]$. Expression

$$u_{t_1:t_2} = u_{t_1}, u_{t_1+1}, u_{t_1+2}, \ldots, u_{t_2} \tag{2.1}$$

denotes the sequence of all control actions applied between time t_1 and t_2, for $t_1 \leq t_2$. A change in the state of the environment is not restricted to the actions of the robot. The fact that time passes is by itself a control information. Therefore, it is assumed that at every time step t there is exactly one control data item that includes the action "do-nothing" as a valid action.

Measurement The robot senses the state of the environment through a perception process. This process updates the robot's internal state of the environment by extracting information from the data acquired with the robot sensors. For example, the robot can take a camera image or a range scan to collect information about the environment state. The result of this perceptual interaction is called *measurement* or *observation*, being both terms used interchangeably. The acquisition of sensor measurements is not done without delay, and thus, they provide information about the state a few moments ago.

The data carried by a measurement provides information about the environment state at a specific point in time. Measurement data can include camera images, depth, range scans, magnetic north, and so on. Small temporary differences in the acquisition of all data in a measurement (e.g. a laser sensor scans the environment sequentially at very high speed) can be ignored. Let us denote z the measurement data, and z_t the measurement data at time t. Although it is possible for a robot to acquired more than one measurement within a single time step, for the sake of notation only one measurement at a time is assumed to take place. Like before

$$z_{t_1:t_2} = z_{t_1}, z_{t_1+1}, z_{t_1+2}, \ldots, z_{t_2} \tag{2.2}$$

denotes the sequence of all measurements acquired between time t_1 and t_2, for $t_1 \leq t_2$.

It is important to maintain a distinction between control and measurement as they can play fundamental different roles, as for example, in mobile robot localization. Motion (or any control action) tends to exert a loss of information about the state of the environment due to the noisy nature of the robot actuators and the randomness of the robot environment. Perception, on the other hand, adds knowledge about the state of the environment. It is also important to clarify that actions and perceptions are *not* separated in time, but rather concurrent.

2.1.2 Probabilistic Evolution of Controls, States and Measurements

The evolution of controls, states and measurements is controlled by probabilistic laws. State $\mathbf{x}_t$ is generated stochastically from the state $\mathbf{x}_{t-1}$, therefore it is adequate to define the probability distribution from which state $\mathbf{x}_t$ is generated. Taking into account that the emergence of state $\mathbf{x}_t$ might be affected on all past states, measurements and controls, the probabilistic law that represents the evolution of state can be defined by the following probability distribution:

$$p(\mathbf{x}_t|\mathbf{x}_{1:t-1}, z_{1:t-1}, u_{1:t})\,. \tag{2.3}$$

Assuming that the state $\mathbf{x}$ is complete, then, the state $\mathbf{x}_{t-1}$ is set to be a sufficient statistic of all previous controls and measurements up to that point in time, that is $u_{1:t-1}$ and $z_{1:t-1}$. From expression (2.3) only the variable u_t matters if the state $\mathbf{x}_{t-1}$ is known. Hence, the *conditional independence* property can be used to achieve the following equality:

$$p(\mathbf{x}_t|\mathbf{x}_{1:t-1}, z_{1:t-1}, u_{1:t}) = p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)\,. \tag{2.4}$$

A similar reasoning can be applied to the process that models the generation of measurements. Assuming that the state $\mathbf{x}_t$ is complete, the following *conditional independence* can be obtained:

$$p(z_t|\mathbf{x}_{1:t-1}, z_{1:t-1}, u_{1:t}) = p(z_t|\mathbf{x}_t)\,. \tag{2.5}$$

This implies that the state $\mathbf{x}_t$ is sufficient to predict a (noisy) measurement z_t because the information from past measurements, controls and even states are irrelevant if $\mathbf{x}_t$ is complete.

The probability $p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)$ is called the *state transition probability*. It models how the environment state evolves over time as a function of robot controls u_t. The stochastic nature of robot environments is the reason why $p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)$ is a probability distribution and not a deterministic function. The probability $p(z_t|\mathbf{x}_t)$ is called the *measurement probability*. It

models the process by which measurements z_t are generated from the environment state $\mathbf{x}_t$ using a probability distribution. Measurements can also be seen as noisy projections of the state. The controls, states and measurements evolve in the following way: at time t the state is stochastically dependent on the state at time $t-1$ and control u_t; the measurements z_t is stochastically dependent on the state at time t. This probabilistic temporal generative model, depicted in Figure 2.1, is known as *dynamic Bayes network* (DBN).

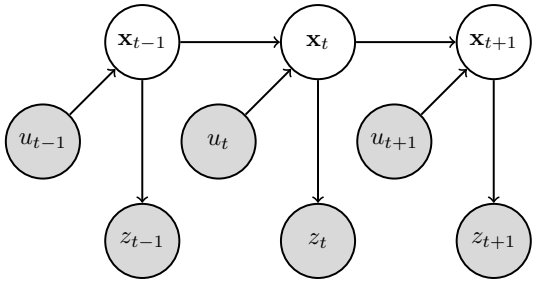


Figure 2.1: Evolution of controls, states and measurements characterized by a dynamic Bayes network.

2.1.3 Bayes Filter

The internal knowledge of the robot about the state of the environment is represented by a *belief*. As already discussed, it is not possible to directly sense the state. For example, the pose of the robot must be inferred from data, because even with sensors like a Global Positioning System (GPS) it is not possible to have the value of the true pose. Therefore, the internal *belief* is used to distinguish the internal knowledge from the true state.

A belief can be represented through conditional probability distributions. For each possible hypothesis with regards to the true state a probability (or density value) is assigned by the belief distribution. Let us denote a belief over a state variable $\mathbf{x}_t$ by $bel(\mathbf{x}_t)$ which is an abbreviation of

$$bel(\mathbf{x}_t) = p(\mathbf{x}_t | z_{1:t}, u_{1:t}) \,, \tag{2.6}$$

the posterior of the probability distribution over the state $\mathbf{x}_t$ constrained on all past measurements $z_{1:t}$ and all past controls $u_{1:t}$. Sometimes it is useful to calculate the posterior *before* integrating z_t, right after executing the control u_t. The resulting posterior is represented by

$$\overline{bel}(\mathbf{x}_t) = p(\mathbf{x}_t | z_{1:t-1}, u_{1:t}) \,. \tag{2.7}$$

In the context of probabilistic filtering, this posterior is also referred to as *prediction*, in the sense that $\overline{bel}(\mathbf{x}_t)$ predicts the state $\mathbf{x}_t$ based on the previous state posterior $bel(\mathbf{x}_{t-1})$ right before integrating the measurements z_t. Calculating $bel(\mathbf{x}_t)$ from $\overline{bel}(\mathbf{x}_t)$ is called *measurement update*.

The *Bayes filter* algorithm presents the most general method for calculating beliefs. It calculates the belief distribution *bel* from control data and measurement. The pseudo-algorithm of the Bayes filter is presented in algorithm 1. It receives as input the belief *bel* at time $t-1$, along with the most recent control data u_t and measurement z_t, and outputs the belief $bel(\mathbf{x}_t)$ at time t. The belief $bel(\mathbf{x}_t)$ at time t is calculated from the belief $bel(\mathbf{x}_{t-1})$ at time $t-1$, making the Bayes filter a recursive function.

The Bayes filter, at its core, has two essential steps. The first step is called *prediction* (line 3 of the algorithm). In this step the control u_t is integrated. It calculates the belief

Algorithm 1: General algorithm for Bayes filtering (Thrun et al., 2005).

1: **Bayes Filter(** $bel(\mathbf{x}_{t-1}), u_t, z_t$ **):**
2: **for all x do**
3: $\overline{bel}(\mathbf{x}_t) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)\ bel(\mathbf{x}_{t-1})\ d\mathbf{x}_{t-1}$
4: $bel(\mathbf{x}_t) = \eta\ p(z_t|\mathbf{x}_t)\ \overline{bel}(\mathbf{x}_t)$
5: **end**
6: **return** $bel(\mathbf{x}_t)$

over state $\mathbf{x}_t$ based on the prior belief over state $\mathbf{x}_{t-1}$ and the control u_t. The predicted state $\overline{bel}(\mathbf{x}_t)$ is calculated by the integral (sum) of the product of the prior assigned to $\mathbf{x}_{t-1}$ and the probability of the state evolution from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$ influenced by the control u_t.

The second step is called *measurement update* (line 4 of the algorithm). The belief $\overline{bel}(\mathbf{x}_t)$ calculated in the previous step is multiplied by the probability that the measurement z_t may have been observed for all hypothetical posterior state $\mathbf{x}_t$. The resulting product is usually not a probability (it may not integrate to one). Hence, the existence of the constant η that normalizes the final belief $bel(\mathbf{x}_t)$.

The recursive computation of the posterior belief requires an initial belief $bel(\mathbf{x}_1)$ as boundary condition. If the value of $\mathbf{x}_1$ is known, $bel(\mathbf{x}_1)$ should be initialized with a mass point distribution with all probability mass centered around the correct value of $\mathbf{x}_1$, and zero probability elsewhere. If the value $\mathbf{x}_1$ is not known, an uniform distribution over the domain of $\mathbf{x}_1$ can be assigned to the initial belief $bel(\mathbf{x}_1)$. Partial knowledge about the value of $\mathbf{x}_1$ can be modeled using non-uniform distributions, although, in practice, it is common to assume complete knowledge or complete ignorance.

2.2 Mobile Robot Localization

Mobile robot localization is the problem of estimating the pose of a robot relative to a given representation (or map) of the environment. Cox (1991), considered it to be the fundamental problem to provide a mobile robot with autonomous capabilities and it holds true up to our days. From a different point of view, mobile robot localization can also be seen as a problem of coordinate transformation (Thrun et al., 2005). The map of the environment is described in a global coordinate system that are independent of the pose of the robot. The localization procedure is responsible for determining the association between the map coordinate system and the local coordinate system of the robot. This association allows the robot to apply the necessary coordinate transformation to express the location of known objects within its own coordinates frame. Nowadays, most of autonomous mobile robotic tasks requires knowledge about the pose of the robot and the position of the objects to be handled.

Unfortunately, the pose of the robot can *not* be sensed directly. Most robots, if not all, do not possess error free measuring sensors, therefore, the pose of the robot has to be integrated over time considering all sources of information (sensors and actions). In a probabilistic framework, the localization problem can be defined by the following probability distribution (Thrun et al., 2005):

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, z_t, u_t, m) \tag{2.8}$$

where $\mathbf{x}_t$ is the robot's pose, z_t the available measurements, u_t the control data (usually odometry information) and m is the map of the environment. This formulation can be further

expanded into:

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, z_t, u_t, m) \propto p(z_t \mid \mathbf{x}_t, m)\ p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, u_t) \tag{2.9}$$

Here, the factor $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, u_t)$ models how the robot's pose evolves overs time as a function of robot controls u_t and $p(z_t \mid \mathbf{x}_t, m)$ models the process by which the measurements z_t are generated from the environment with respect to the pose of the robot. These models are known as *motion model* and *measurement model*, respectively.

2.2.1 Motion Models

The motion model $p(\mathbf{x}_t \mid u_t, \mathbf{x}_{t-1})$ plays a fundamental role in predicting the evolution of the robot's pose. It describes the posterior distribution of the robot's pose $\mathbf{x}_t$ after executing the control u_t at $\mathbf{x}_{t-1}$, that is, a probabilistic generalization of the robot *kinematics* (Cox and Wilfong, 1990). The effect of the control actions on the robot's pose is described by the calculation of its kinematics. The pose (or state) of a rigid robot is usually described by six variables, three for its Cartesian coordinates (x, y and z), and three for its Euler angles (pitch, roll and yaw). In terms of localization, this thesis is mostly focuses on mobile robots operating in a planar environment, therefore the state of the robot is characterized by a pose with three variables: two for its Cartesian coordinates (x and y) and one for its heading direction (yaw).

Let $\mathbf{x}_t = (x\ y\ \theta)^T$ denote the robot pose at time t. The orientation of the robot θ, sometimes called *bearing* or *heading direction*, is assumed to have orientation equal to 0 rad pointing into the direction of its $x-$axis and orientation equal to $\pi/2$ rad when pointing into the direction of its $y-$axis. The pose without orientation is usually referred to as *location*. It is a two-dimensional vector which refers to the $x-y$ coordinate of an object. The probabilistic kinematic model, or *motion model*, is the state transition model in mobile robotics, and is modeled by the conditional probability $p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)$. Here, $\mathbf{x}_t$ and $\mathbf{x}_{t-1}$ are both robot poses and u_t is the control action. This model describes the posterior distribution of the robot pose $\mathbf{x}_t$ after executing the control u_t at $\mathbf{x}_{t-1}$. Note that in actual implementations of the model, the control data is usually provided by the odometry of the robot.

There are two motion model to describe the effect of a control action. They somewhat complement the type of motion information that is being processed. The *velocity motion model* assumes that the control action u_t specifies the velocity commands sent to the motor of the robot. The *odometry motion model* assumes that one has access to odometry information, and the integration of such information is to some extend different from the velocity model. In practice, the odometry model tends to be more accurate than velocity models, with the simple justification that most robots do not execute velocity commands with the same level of reliability that can be obtained by measuring the revolution of the wheels of the robot. However, odometry is only available after executing a motion command. Therefore, it is best suitable for state estimation whereas the velocity model is best used for probabilistic motion planning. For the purpose of mobile robot pose estimation only the odometry motion model (Borenstein et al., 1996) is described.

Odometry Motion Model

Odometry is usually attained by integrating the information of wheel encoders (e.g. every tenth of a second). In reality, odometric information are sensor measurements, not controls, and to model it as measurements the resulting estimates would have to include the actual velocity as state variables. However, it would increase the dimension of the state space,

therefore, to maintain a small state space, the odometry information is assumed to be data from control signals. The true pose of the robot, at time t, is modeled by the random variable $\mathbf{x}_t$. The robot odometry is used to estimate this pose, however, due to the existence of drift and slippage it is not possible to get a consistent coordinate transformation between the coordinates of the robot's odometry and the physical world coordinates. If it was possible, the robot localization problem would be solved by simply using dead reckoning.

Let $\bar{\mathbf{x}}_t = (\bar{x}, \bar{y}, \bar{\theta})^T$ denote the odometry measurement fixed in an internal coordinate of the robot whose relation to the global world coordinates is unknown. In the time interval $(t-1, t]$ that the robot advances from pose $\mathbf{x}_{t-1}$ to pose $\mathbf{x}_t$, the odometry reports back the difference in motion from $\bar{\mathbf{x}}_{t-1}$ to $\bar{\mathbf{x}}_t$. To use this information in pose estimation one must notice that the relative difference between $\bar{x}_{t-1}, \bar{x}_t$, under an appropriate definition of "difference", is a good estimator for the difference of the true poses $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$. Therefore, the control u_t is calculated from the pair $\langle \mathbf{x}_t - 1, \mathbf{x}_t \rangle$

The control u_t is transformed into a sequence of three steps to extract the relative odometry information: an initial rotation δ_{rot1}, followed by a translation δ_{trans}, and finished by another rotation δ_{rot2}. Each pair of positions $(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$ has a unique parameter vector $(\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}})^T$ that is sufficient to reconstruct the relative motion between $\bar{s}$ and $\bar{s}'$. Therefore, the parameters $\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}}$ form a sufficient statistics of the relative motion encoded by the odometry. The probabilistic motion models assume that these parameters are polluted by independent noise. The algorithm 2 describes the method for computing $p(\mathbf{x}_t | \mathbf{x}_{t-1}, u_t)$ from odometry. It accepts as input an initial pose $\mathbf{x}_{t-1}$, a pair of poses $u_t = (\bar{\mathbf{x}}_{t-1}, \bar{\mathbf{x}}_t)^T$ obtained from the odometry of the robot, and a possible final pose $\mathbf{x}_t$. The output is the numeric value of the probability $p(\mathbf{x}_t | \mathbf{x}_{t-1}, u_t)$. The odometry relative motion parameters $(\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}})^T$ are calculated in lines 2 to 4. The associated relative motion parameters $(\hat{\delta}_{\text{rot1}}, \hat{\delta}_{\text{trans}}, \hat{\delta}_{\text{rot2}})^T$ for the given poses $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ are calculated in lines 5 to 7. The individual probabilities for individual motion parameters are calculated through 8 to 10. The function $\mathbf{prob}(a, b^2)$ implements an error distribution over a with mean 0 and variance b^2. Finally the output of the algorithm is calculated in line 11 and it is the combined probability error probability of the individual error probabilities p_1, p_2 and p_3. The variables α_1 through α_4 are robot-specific parameters that can be used to specify the noise in robot motion.

Algorithm 2: Odometry motion model algorithm (Thrun et al., 2005).

1: **Odometry Motion Model(** $\mathbf{x}_t, u_t, \mathbf{x}_{t-1}$ **):**
2: $\quad \delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
3: $\quad \delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
4: $\quad \delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$
5: $\quad \hat{\delta}_{\text{rot1}} = \text{atan2}(y' - y, x' - x) - \theta$
6: $\quad \hat{\delta}_{\text{trans}} = \sqrt{(x - x')^2 + (y - y')^2}$
7: $\quad \hat{\delta}_{\text{rot2}} = \theta' - \theta - \hat{\delta}_{\text{rot1}}$
8: $\quad p_1 = \mathbf{prob}(\delta_{\text{rot1}} - \hat{\delta}_{\text{rot1}}, \alpha_1 \hat{\delta}^2_{\text{rot1}} + \alpha_2 \hat{\delta}^2_{\text{trans}})$
9: $\quad p_2 = \mathbf{prob}(\delta_{\text{trans}} - \hat{\delta}_{\text{trans}}, \alpha_3 \hat{\delta}^2_{\text{trans}} + \alpha_4 \hat{\delta}^2_{\text{rot1}} + \alpha_4 \hat{\delta}^2_{\text{rot2}})$
10: $\quad p_3 = \mathbf{prob}(\delta_{\text{rot2}} - \hat{\delta}_{\text{rot2}}, \alpha_1 \hat{\delta}^2_{\text{rot2}} + \alpha_2 \hat{\delta}^2_{\text{trans}})$
11: $\quad$ **return** $p_1 \cdot p_2 \cdot p_3$

An alternative way to apply the motion model is used in particle filters algorithms. The use of particle filters, require an algorithm for *sampling* from $p(\mathbf{x}_t | \mathbf{x}_{t-1}, u_t)$ rather than a

closed-form expression for computing $p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)$ for any $\mathbf{x}_t, \mathbf{x}_{t-1}$ and u_t. The algorithm for sampling from the odometry motion model is presented in algorithm 3. Its input is an initial pose $\mathbf{x}_{t-1}$ and odometry data u_t, and its output is a random $\mathbf{x}_t$ distributed according to $p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)$. The difference to the motion model algorithm is that it randomly computes $\mathbf{x}_t$ (lines 5 to 7) instead of computing the probability of a given $\mathbf{x}_t$.

Algorithm 3: Odometry sample motion model algorithm (Thrun et al., 2005).

1: **Odometry Sample Motion Model(** $u_t, \mathbf{x}_{t-1}$ **):**
2: $\quad \delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
3: $\quad \delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
4: $\quad \delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$
5: $\quad \hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1\delta^2_{\text{rot1}} + \alpha_2\delta^2_{\text{trans}})$
6: $\quad \hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3\delta^2_{\text{trans}} + \alpha_4\delta^2_{\text{rot1}} + \alpha_4\delta^2_{\text{rot2}})$
7: $\quad \hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1\delta^2_{\text{rot2}} + \alpha_2\delta^2_{\text{trans}})$
8: $\quad x' = x + \hat{\delta}_{\text{rot1}} \cos(\theta + \hat{\delta}_{\text{rot1}})$
9: $\quad y' = y + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}})$
10: $\quad \theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$
11: $\quad$ **return** $\mathbf{x}_t = (x', y', \theta')^T$

2.2.2 Measurement Models

The measurement model $p(z_t|\mathbf{x}_t, m)$ plays a fundamental role in the measurement update step of the Bayes filter. It physically models the sensors of the robot. Assuming that the robot pose $\mathbf{x}_t$ and the map m are known, the measurements model specifies the probability that z_t is measured. It is common, for sensors equipped in robots, to generates more than one measurement at time t:

$$z_t = \{z_t^1, z_t^2, \ldots, z_t^K\} \tag{2.10}$$

Thus, the resulting calculation is the collection of probabilities $p(z_t^k|\mathbf{x}_t, m)$, where z_t^k is the kth measurement, and K the total number of such measurements. Assuming conditional independence between the measurements, the resulting probability is given by:

$$p(z_t|\mathbf{x}_t, m) = \prod_i^K p(z_t^i|\mathbf{x}_t, m) \ . \tag{2.11}$$

Note that the assumption of conditional independence is only true on ideal cases. Dependencies in noise may exist due to adjacent measurements corrupted by people, errors in the model m, approximations in the posterior, etc. However, if these violations are ignored the assumption of conditional independence maintains.

Beam Models of Range Finders

Range finder is a common sensor in robotics and it measure the range to nearby objects. Ranges may be measure along a beam (i.e. laser range finders) or within a cone (i.e. ultrasonic sensors). The model that approximates the physical model of range finders is known as *beam model* (Moravec, 1989; Fox et al., 1999b; Thrun et al., 2005). The beam model can consider

four types of errors: local measurement noise, errors due to unexpected objects, errors due to failure to detect objects, and random unexplained noise. The expected model $p(z_t|\mathbf{x}_t, m)$ can be a mixture of four probabilities, each of which corresponds to a specific type of error.

Measurement noise: Let z_t^{k*} denote the "true" range of the object measured by z_t^k. In location-based maps, a *ray casting* function is used to obtained the range z_t^{k*}. In feature-based maps, the range z_t^{k*} is usually obtained by searching for the closest feature within a measurement cone. However, even if the measurement is correct its value is subject to error. This *measurement noise* is usually modeled by a Gaussian, denoted by p_{hit}, with mean z_t^{k*} and standard deviation σ_{hit}. The values measured by the range sensor are limited to the interval $[0; z_{\max}]$, where $z_{\max}$ is the maximum sensor range. Therefore, the measurement probability is specified by:

$$p_{\text{hit}}(z_t^k|\mathbf{x}_t, m) = \begin{cases} \eta\, \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2) & \textbf{if } 0 \leq z_t^k \leq z_{\max} \\ 0 & \textbf{otherwise} \end{cases}, \tag{2.12}$$

where z_t^{k*} is obtained from $\mathbf{x}_t$ and m via ray casting, and $\mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2)$ a univariante Gaussian with mean z_t^{k*} and standard deviation σ_{hit}. The normalizer η is calculated by

$$\eta = \left(\int_0^{z_{\max}} \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{\text{hit}}^2)\, dz_t^k \right)^{-1}. \tag{2.13}$$

Unexpected objects: The environments in which robots operate are dynamic, whereas maps m are static. As consequence, objects not represented in the map can cause range finders to produce surprisingly short ranges. Instead of treating dynamic objects as part of the state vector (to estimate their location), treat the objects as noise. This way, not modeled objects have the property that they cause ranges to be shorter than z_t^{k*}. The likelihood of sensing unexplained objects decrease with range. Such situation is described by an *exponential distribution*. Let λ_{short} denote the parameter of the exponential distribution, an intrinsic parameter of the measurement model. The probability $p_{\text{short}}(z_t|\mathbf{x}_t, m)$ takes the following form:

$$p_{\text{short}}(z_t^k|\mathbf{x}_t, m) = \begin{cases} \eta\, \lambda_{\text{short}} e^{-\lambda_{\text{short}} z_t^k} & \textbf{if } 0 \leq z_t^k \leq z_{\max} \\ 0 & \textbf{otherwise} \end{cases}, \tag{2.14}$$

Like in the previous case, the normalizer η is needed since the exponential is limited to the interval $[0; z_{\max}]$:

$$\eta = \left(\int_0^{z_{\max}} \lambda_{\text{short}} e^{-\lambda_{\text{short}} z_t^k} dz_t^k \right)^{-1} = \frac{1}{1 - e^{-\lambda_{\text{short}} z_t^{k*}}} \tag{2.15}$$

Failures: Occasionally, objects are just not detected by range finders. For example, for sonar sensor this happen very often as result of specular reflection, and laser range finders fail when sensing black, light-observing objects. A *sensor failure* can be manifested by a max-range measurement, that is, the sensor return its maximum possible value $z_{\max}$. This situation is modeled by a point-mass distribution centered at $z_{\max}$:

$$p_{\max}(z_t^k|\mathbf{x}_t, m) = \begin{cases} 1 & \textbf{if } z = z_{\max} \\ 0 & \textbf{otherwise} \end{cases}. \tag{2.16}$$

Random measurements: There is also the possibility of entirely *unexplainable measurements* from range finders. Sonars, for example, often generate phantom readings when they bounce off walls. Similar situation can also happen with laser range finders. Such measurements are modeled by an uniform distribution spread over the entire sensor measurement range $[0; z_{\max}]$:

$$p_{\text{rand}}(z_t^k|\mathbf{x}_t, m) = \begin{cases} \frac{1}{z_{\max}} & \textbf{if } 0 \leq z_t^k \leq z_{\max} \\ 0 & \textbf{otherwise} \end{cases} . \tag{2.17}$$

These four different probabilities are mixed by a weighted average defined by the parameters $w_{\text{hit}}, w_{\text{short}}, w_{\max}$ and w_{rand} with $w_{\text{hit}} + w_{\text{short}} + w_{\max} + w_{\text{rand}} = 1$.

$$p(z_t^k|\mathbf{x}_t, m) = \begin{pmatrix} w_{\text{hit}} \\ w_{\text{short}} \\ w_{\max} \\ w_{\text{rand}} \end{pmatrix}^T \cdot \begin{pmatrix} p_{\text{hit}}(z_t^k|\mathbf{x}_t, m) \\ p_{\text{short}}(z_t^k|\mathbf{x}_t, m) \\ p_{\max}(z_t^k|\mathbf{x}_t, m) \\ p_{\text{rand}}(z_t^k|\mathbf{x}_t, m) \end{pmatrix} \tag{2.18}$$

An implementation of the beam model is presented in algorithm 4. It takes as input a range scan z_t, a robot pose $\mathbf{x}_t$ and a map m. The loop (lines 2 and 7) multiplies the likelihood of individual sensor beams z_t^k, following Equation 2.11. A ray casting function (in line 4) is used to calculate the noise-free range z_t^{k*} for a particular measurement. In line 5, the likelihood of each individual range measurement k_t^k is computed following Equation 2.18. Finally, the algorithm returns the desired probability $p(z_t|\mathbf{x}_t, m)$.

Algorithm 4: Beam model algorithm (Thrun et al., 2005).

```
1: Beam Model( z_t, x_t, m ):
2:     q = 1
3:     for k = 1 to K do
4:         compute z_t^{k*} for the measurement z_t^k using ray casting
5:         p = w_hit · p_hit(z_t^k|x_t, m) + w_short · p_shor(z_t^k|x_t, m) +
               w_max · p_max(z_t^k|x_t, m) + w_rand · p_rand(z_t^k|x_t, m)
6:         q = q · p
7:     end
8:     return q
```

Likelihood Field for Range Finders

While the *beam model* applies a *ray casting* function to find the "true" z_t^{k*} range of the object measured by z_t^k, the *likelihood field* (Thrun, 2001; Thrun et al., 2005) projects the endpoint of z_t^k into global coordinates of the map m to find the distance to the nearest object. Let $\mathbf{x}_t = (x\ y\ \theta)^T$ denote the robot pose at time t, $(x_{k,\text{sens}}\ y_{k,\text{sens}})^T$ the relative position of the sensor on frame of the robot, and $\theta_{k,\text{sens}}$ the angular orientation of the sensor relative to the heading of the robot. The endpoint of the measurement z_t^k, in global coordinates, is given by the transformation:

$$\begin{pmatrix} x_{z_t^k} \\ y_{z_t^k} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \left[\begin{pmatrix} x_{k,\text{sens}} \\ y_{k,\text{sens}} \end{pmatrix} + z_t^k \begin{pmatrix} \cos\theta_{k,\text{sens}} \\ \sin\theta_{k,\text{sens}} \end{pmatrix} \right] \tag{2.19}$$

These coordinates are only valid when the sensor actually detects an obstacle. Therefore, whenever the range measurement takes on its maximum value $z_t^k = z_{\max}$ it is simply discarded by the likelihood field model. Similar to the beam model, three types of errors are assumed.

Measurement noise. The noise of the process is modeled by a probability density that requires finding the nearest object in the map. Let Δ denote the Euclidean distance between $(x_{z_t^k}\ y_{z_t^k})^T$ and the nearest object in the map m. The value of Δ_i is computed by a search function defined by

$$\mathcal{D}(x_{z_t^k}, y_{z_t^k}) = \min_{x',y'} \left(\sqrt{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2} \mid \langle x', y' \rangle \text{occupied in } m \right) . \tag{2.20}$$

Then, the measurement noise can be modeled by a zero-centered Gaussian:

$$p_{\text{hit}}(z_t^k | \mathbf{x}_t, m) = \epsilon_{\sigma_{\text{hit}}}(\Delta) . \tag{2.21}$$

The density p_{hit} is obtained by intersecting (and normalizing) the likelihood field by the sensor axis.

Failures. Like for the beam model, it is assumed that max-range readings have a large likelihood modeled by a point-mass distribution $p_{\max}$.

Random measurements. To finalize, a uniform distribution p_{rand} is used to model random noise in perception.

Again, the desired probability $p(z_t^k|\mathbf{x}_t, m)$ integrates all three distributions: $z_{\text{hit}} \cdot p_{\text{hit}} + z_{\text{rand}} \cdot p_{\text{rand}} + z_{\max} \cdot p_{\max}$ using the known mixing weights $z_{\text{rand}}, z_{\text{hit}}$ and $z_{\max}$. An implementation of the likelihood field model is presented in algorithm 5. The outer loop, as already discussed in the beam model, multiplies the individual values of $p(z_t^k|\mathbf{x}_t, m)$. The value of the range reading is validated in line 4, in case of a max range reading the measurement z_t^k is discarded. The measurement end-point is calculated in line 5 and 6. The calculation of the nearest obstacle in $x-y-$space is executed (in line 7), and the resulting likelihood is obtained in line 8 by mixing a normal and uniform distribution. The function $\textbf{prob}(\Delta, \sigma_{\text{hit}})$ calculates the probability of Δ under a Gaussian of mean zero and standard deviation σ_{hit}.

Algorithm 5: Likelihood field model (Thrun et al., 2005).

1: **Likelihood Field Model(** $z_t, \mathbf{x}_t, m$ **):**
2: $q = 1$
3: **for all** k **do**
4: **if** $z_t^k \neq z_{max}$ **then**
5: $x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos\theta + \theta_{k,\text{sens}}$
6: $y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin\theta + \theta_{k,\text{sens}}$
7: $\Delta = \mathcal{D}(x_{z_t^k}, y_{z_t^k})$
8: $q = q \cdot (z_{\text{hit}} \cdot \textbf{prob}(\Delta, \sigma_{\text{hit}}) + \frac{z_{\text{rand}}}{z_{\max}})$
9: **end**
10: **return** q

2.2.3 A Taxonomy of Localization Problems

Not all localization problems have the same degree of complexity. In this section, the taxonomy of localization problems are discussed to provide a better understanding of the difficulty of those problems. The localization problems are divided by this taxonomy along several important dimensions according to the nature of the environment and initial knowledge that a robot may hold relative to the localization problem in question (Thrun et al., 2005).

Local Versus Global Localization The localization problem can be divided into three types of localization problems with an increase of complexity. These problems are characterized by the knowledge that is available at the beginning and at run-time.

- *Position tracking* or *local localization* assumes that the *initial* robot pose is known (Schiele and Crowley, 1994; Weiss et al., 1994; Borenstein et al., 1996). The pose of the robot can be attained by handling the effect of the noise in robot motion, which is usually small. This is a *local* problem because the uncertainty is restricted to a region near the true pose of the robot. The uncertainty of the pose is commonly modeled by a unimodal distribution (e.g. Gaussian).

- The *global localization* makes no assumptions about the *initial* pose of the robot (Burgard et al., 1998; Roumeliotis and Bekey, 2000; Jensfelt and Kristensen, 2001). The robot is initially placed anywhere in its environment without any knowledge about its location. Bounding errors of the pose can not be assumed, and unimodal probabilistic distributions are usually ill-suited. The *global localization* problem is more difficult to solve than *position tracking*.

- The *kidnapped robot* is a variant of *global localization* that is even more difficult to solve (Engelson and McDermott, 1992; Fox et al., 1999b; Lima et al., 2011). During operation a robot can be kidnapped and teleported to any other location. The *kidnapped robot* is more difficult to solve than the *global localization* problem because the robot can have a wrong belief about its location in the environment. In *global localization* the robot knows that it does not know where it is. Although, during operation, a robot is rarely kidnapped this problem has a practical importance, that is the ability to recover from localization failures.

Static Versus Dynamic Environments The environment is a dimension with a profound impact on the difficulty of localization. They can be static or dynamic. In a *static environment* the pose of the robot is the only existing state (or variable quantity). That is, in a static environment only the robot moves while other object remain in the same location, forever. In a *dynamic environment* existing objects, other than the robot, may change their location or configuration over time. Changes that persist over time with an impact in more than a single sensor reading, can affect the outcome of the localization procedure. Untraceable changes are of course of no relevance to localization, and those that affect a single measurement are better handled as noise. In practice, most real environment are dynamic, with states changes happening at different rates. Example of continuing changes are: people, doors, movable furniture or even other mobile robots.

Passive Versus Active Approaches A third dimension that differentiates localization problems is the fact of whether or not the localization algorithms has control over the motion of the robot. There are two cases, *passive* and *active* localization. In *passive* localization algorithms, the localization module is only an *observer* of the operations of the robots. The motion of the robot is controlled by other modules that pay no attention about facilitating localization. In *active* localization algorithms, the localization module is considered in the control of the robot as to minimize localization errors and possible problems of moving a deficiently localized robot into a risky place. Active approaches have in general better localization results than passive ones. However, they require control over the robot motion, which in practice tends to be insufficient: the robot requires to be able to localize itself even when executing some other tasks than localization.

Single-Robot Versus Multi-Robot A fourth dimension of the localization problem has to do with the number of robots involved. The most common approach to localization is *single-robot localization*. It handles only one robot. It is the most convenient form for localization algorithms as all data is collected at a single robot platform and there is no communication issues. The *multi-robot localization* (Fox et al., 2000; Rekleitis et al., 2002; Martinelli et al., 2005) problem come into existence in a team of robots. If each robot can localize itself individually, the multi-robot localization can be solved using single-robot localization. However, if a robot can observe other robots, there is an opportunity to improve localization. If the relative position of both robots are known, or other type of information about the relative positions, then the belief of one robot can be use to bias the belief of the other. The multi-robot localization approach raises the issue of non-trivial representation of beliefs and the nature of communication between them.

2.2.4 Localization Approaches

As already stated, localization is the problem of determining the pose of the robot relative to the given map of the environment, by extracting information from actions and measurements (Figure 2.2). Localization approaches have been developed over the years for a large set of maps, such as *feature-based* and *geometric-based* — maps will be discussed in the next chapter. Localization assumes the existence of an accurate map of the environment. In this section several algorithms for mobile localization are presented.

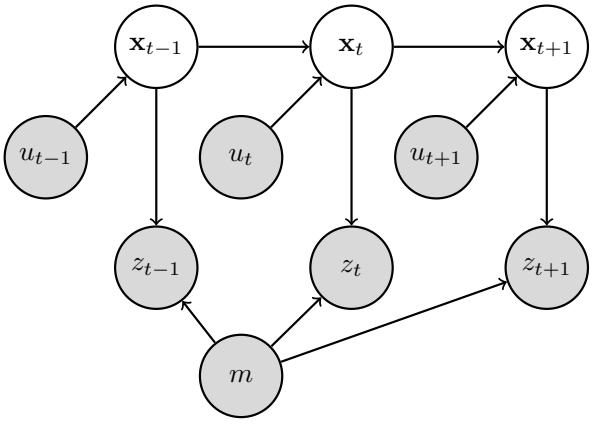


Figure 2.2: Graphic model of mobile robot localization.

Markov Localization

The *Markov localization* (Koenig and Simmons, 1998; Fox et al., 1999b) is the direct application of the Bayes filter stated in algorithm 1. The algorithm 6 shows the basic Markov localization algorithm. The Markov localization only requires an additional input, a map m. The map m directly influences the measurement model $p(z_t|\mathbf{x}_t, m)$ (in line 4). Although not always necessary, it is also incorporated in the motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t, m)$ (in line 3). Just like in the Bayes filter, the probabilistic belief at time $t-1$ is transformed into a belief at time t. The Markov localization approach handles the global localization problem, the position tracking problem and the kidnapped robot problem in static environment.

The initial belief $bel(\mathbf{x}_1)$, that reflects the initial knowledge of the pose of the robot, is set differently depending on the type of localization problem.

- **Position tracking.** When the initial pose is known the initial belief $bel(\mathbf{x}_1)$ is set to a point-mass distribution

$$bel(\mathbf{x}_1) = \begin{cases} 1 & \text{if } \mathbf{x}_1 = \bar{\mathbf{x}}_1 \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

  where $\bar{\mathbf{x}}_1$ denotes the known initial pose. However, because point-mass distributions are discrete and without a density, it is replaced by an approximation, usually a narrow Gaussian distribution centered in $\bar{\mathbf{x}}_1$

$$bel(\mathbf{x}_1) =\sim \mathcal{N}(\mathbf{x}_1; \bar{\mathbf{x}}_1, \Sigma)\,, \tag{2.23}$$

  with Σ as the covariance of the initial pose uncertainty.

- **Global localization.** In the case of an unknown initial pose, the belief $bel(\mathbf{x}_1)$ is initialized by an uniform distribution over the valid space of all valid poses in the map:

$$bel(\mathbf{x}_1) = \frac{1}{|X|}\,, \tag{2.24}$$

  where $|X|$ is the volume of the space of all valid poses in the map.

- **Other.** In the case of partial knowledge about the initial pose of the robot, the initial distribution can be easily manipulated. For example, if the robot is next to a door, the belief $bel(\mathbf{x}_1)$ can be initialized with a density that is zero for all places except for regions near a door.

Algorithm 6: Markov Localization (Thrun et al., 2005).

```
1: Markov localization( bel(x_{t-1}), u_t, z_t, m ):
2:     for all x_t do
3:         bel̄(x_t) = ∫ p(x_t|x_{t-1}, u_t, m) bel(x_{t-1}) dx
4:         bel(x_t) = η p(z_t|x_t, m) bel̄(x_t)
5:     end
6:     return bel(x_t)
```

EKF Localization

The EKF localization algorithm (Dickmanns and Graefe, 1988) is a special case of Markov localization, it represents beliefs $bel(\mathbf{x}_t)$ by their mean μ_t and covariance Σ_t. The EKF localization algorithm in question assumes the use of a feature-based map. At any point in time t, a vector of ranges r_t and bearings ϕ_t to nearby features $z_t = \{z_t^1, z_t^2, \ldots\}$ is obtained. Let us assume that all features are uniquely identifiable and the correspondence is known. The identification of a feature is expressed by a set of *correspondence variables*, denoted c_t^i, one for each feature vector z_t^i.

The EFK localization algorithm is presented in algorithm 7. It requires as input the Gaussian estimate of the robot pose at time $t-1$, with mean μ_{t-1} and covariance Σ_{t-1}, the control u_t, a map of known features m, and set of features z_t obtained at time t, together with the correspondence variables c_t. The output of the algorithm is a revised estimate u_t, Σ_t. The additions of the EKF localization to the EKF are in lines 4 to 13, a measurement update adapted for localization using features. It loops through all features i measured at time t. The correspondence of the i-th feature in the measurement vector is assigned to j (in line 5). Then it calculates a predicted measurement $\bar{z}_t^i$ (in line 5). The Jacobian H_t^i is then used to calculate the uncertainty S_t^i (in line 7) that corresponds to the predicted measurement z_t^i. The estimate is updated (in line 12 and 13), once for each feature.

The presented EKF localization is only applicable when landmark correspondences can be determined with complete certainty, which in practice is hardly ever true. Therefore, most of localization implementations try to identify the landmarks during the localization process. The *maximum likelihood correspondence* is a simple method to cope with the correspondence problem. First it finds the most likely correspondence variable, and then assumes this value as true. The maximum likelihood technique is not without problems. If there are many equally likely hypotheses for the correspondence variable the outcome may be simply wrong. This problem of false data association may be reduced by selecting landmarks that are sufficiently unique and sufficiently far apart that confusing them with each other is unlikely, but it may

Algorithm 7: EKF localization with known correspondences (Thrun et al., 2005).

1: **EKF localization with known correspondences(** $\mu_{t-1}, \Sigma_{t-1}, u_t, m, z_t, c_t$ **):**
2: $\quad \bar{\mu}_t = g(\mu_{t-1}, u_t)$
3: $\quad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4: $\quad$ **for all** *observed features* $z_t^i = (r_t^i, \phi_t^i)^T$ **do**
5: $\quad\quad j = c_t^i$
6: $\quad\quad \bar{z}_t^i = h(\bar{\mu}_t, m_j)$
7: $\quad\quad S_t^i = H_t^i \bar{\Sigma}_t [H_t^i]^T + Q_t$
8: $\quad\quad K_t^i = \overline{\Sigma}_t [H_t^i]^T [S_t^i]^{-1}$
9: $\quad\quad \bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \bar{z}_t^i$
10: $\quad\quad \bar{\Sigma}_t = (I - K_t^i H_t^i) \bar{\Sigma}_t$
11: $\quad$ **end**
12: $\quad u_t = \bar{\mu}_t$
13: $\quad \Sigma = \bar{\Sigma}_t$
14: $\quad$ **return** μ_t, Σ_t

not be an easy task.

A version of EKF localization with unknown correspondence using maximum likelihood correspondence is presented in algorithm 8 The key differences lies on the measurement update: for each observation, a quantification value for all landmarks k is calculated (in lines 5 to 8). The correspondence variable $j(i)$ is chosen (in line 9) by maximizing the likelihood of the measurement z_t^i given any possible landmark m_k in the map.

Algorithm 8: EKF localization with unknown correspondences (Thrun et al., 2005).

1: **EKF localization(** $\mu_{t-1}, \Sigma_{t-1}, u_t, m, z_t$ **):**
2: $\bar{\mu}_t = g(\mu_{t-1}, u_t)$
3: $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4: **for all** *observed features* $z_t^i = (r_t^i, \phi_t^i)^T$ **do**
5: **for all landmarks** k **in map** m **do**
6: $\bar{z}_t^k = h(\bar{\mu}_t, m_k)$
7: $S_t^k = H_t^k \bar{\Sigma}_t [H_t^k]^T + Q_t$
8: **end**
9: $j(i) = \underset{k}{\text{argmax}} \det(2\pi\Sigma_t^k)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t^i - \bar{z}_t^k)^T [S_t^k]^{-1}(z_t^i - \bar{z}_t^k)\}$
10: $K_t^i = \overline{\Sigma}_t [H_t^{j(i)}]^T [S_t^{j(i)}]^{-1}$
11: $\bar{\mu}_t = \bar{\mu}_t + K_t^i (z_t^i - \bar{z}_t^{j(i)})$
12: $\bar{\Sigma}_t = (I - K_t^i H_t^{j(i)}) \bar{\Sigma}_t$
13: **end**
14: $u_t = \bar{\mu}_t$
15: $\Sigma = \bar{\Sigma}_t$
16: **return** μ_t, Σ_t

Grid Localization

The *grid localization* algorithm utilizes an histogram filter over a grid decomposition of the pose state to approximate the posterior. The histogram filter is characterized by a collection of discrete probability values maintained as posterior

$$bel(\mathbf{x}_t) = \{p_{k,t}\} \tag{2.25}$$

where each grid cell $\mathbf{x}_k$ defines a probability $p_{k,t}$. The set of all cells forms a partition of space of all valid poses:

$$\text{dom}(X_t) = \mathbf{x}_{1,t} \cup \mathbf{x}_{2,t} \cup \ldots \mathbf{x}_{K,t}\,. \tag{2.26}$$

In the basic algorithm of grid localization, the space partition is time-invariant with each grid cell of the same size. The granularity of the space for a mobile robot moving in a plane is, for example, 0.10 meters for the $x-$ and $y-$dimension, and 5 degrees for the rotation dimension. The finer the representation the better are the results, but at the cost of increasing the computational requirements.

The histogram filter, used in the grid localization algorithm, is similar to the discrete version of the Bayes filter but with continuous states, shown in algorithm 9. Its input is the discrete probability values $\{p_{t-1,k}\}$, the control u_t, the measurement z_t and the map m. The

motion update takes place in line 3, and the measurement update in line 4. The initial belief $bel(\mathbf{x}_1)$ is represented by an uniform histogram.

Algorithm 9: Grid Localization with an histogram filter (Thrun et al., 2005).

1: **Grid Localization(** $\{p_{k,t}\}, u_t, z_t$ **):**
2: **for all** k **do**
3: $\bar{p}_{k,t} = \sum_i p_{i,t-1}$ **motion_model**$(\text{mean}(\mathbf{x}_k), u_t, \text{mean}(\mathbf{x}_i))$
4: $p_{k,t} = \eta\, \bar{p}_{k,t}$ **measurement_model**$(z_t, \text{mean}(\mathbf{x}_k), m)$
5: **end**
6: **return** $\{p_{k,t}\}$

Monte Carlo Localization

The *Monte Carlo Localization* (MCL) (Dellaert et al., 1999; Fox et al., 1999a; Thrun et al., 2001; Kwok et al., 2004) is a popular localization algorithm that represents the belief $bel(\mathbf{x}_t)$ by particles. Similar to a grid-based Markov localization, the MCL can be applied to local and global localization. The MCL is one of the most popular localization algorithms in robotics because it s easy to implement and has a tendency to work well across a broad range of localization problems.

The algorithm 10 presents the basic algorithm of MCL. The belief $bel(\mathbf{x}_t)$ is represented by a set of M particles $\mathcal{S}_t = \{\mathbf{x}_t^{[1]}, \mathbf{x}_t^{[2]}, \ldots, \mathbf{x}_t^{[M]}\}$. In line 4, samples are drawn from the motion model using particles of the current belief as starting point. Then, in line 5, the measurement model is used to represent the importance weight of that particle. The initial belief $bel(\mathbf{x}_0)$ can assume a distribution that depends on the knowledge of that same belie. If the initial pose of the robot is known, the initial belief $bel(\mathbf{x}_1)$ is obtained by randomly generating M such particles from the prior distribution $p(\mathbf{x}_1)$ and setting an uniform importance weight M^{-1} to each particle. However, if the initial pose of the robot is unknown, the initial belief $bel(\mathbf{x}_1)$ is obtained by drawing M samples from an uniform distribution within the valid state space. In this case the initial importance weight for each particle is also uniform.

Almost any distribution of practical importance can be approximated by the MCL algorithm. It is not restricted to a limited parametric subset of distributions like, for example, the EKF localization. The accuracy of the localization is increased by the number of particles. The number of particles M is a parameter that can be tuned to trade off the accuracy of the localization and the computational requirements of the MCL. Because the approximations of MCL are of the non-parametric nature, it has the ability to represent complex multi-modal probability distributions, and blend them with focused Gaussian distributions.

The presented MCL form solves the local and global localization problem but it cannot recover from the kidnapped robot or global localization failures. As the particles evolve, they are placed in locations with the most likely pose of the robot. At some point only particles near a single pose are kept, and the algorithm is unable to recover if this pose is incorrect. This problem can be solved with a simple heuristic. Assume that the probability that the robot might get kidnapped is small, therefore it generate a small fraction of random states in the motion model update procedure.

The number of particles M to use is an important parameter for the efficiency of particle

Algorithm 10: Monte Carlo Localization based on particle filters (Thrun et al., 2005).

Monte Carlo Localization($\mathcal{S}_{t-1}, u_t, z_t, m$ **):**
  $\bar{\mathcal{S}}_t = \mathcal{S}_t = \emptyset$
  for $m = 1$ **to** M **do**
    $\mathbf{x}_t^{[m]} =$ **sample_motion_model**$(u_t, \mathbf{x}_{t-1}^{[m]})$
    $w_t^{[m]} =$ **measurement_model**$(z_t, \mathbf{x}_t^{[m]}, m)$
    $\bar{\mathcal{S}}_t = \bar{\mathcal{S}}_t + \langle \mathbf{x}_t^{[m]}, w_t^{[m]} \rangle$
  end
  for $m = 1$ **to** M **do**
    draw i **with probability** $\propto w_t^{[m]}$
    add $\mathbf{x}_t^{[i]}$ **to** $\mathcal{S}_t$
  end
  return $\mathcal{S}_t$

filters. To avoid divergence due to sample depletion in MCL, large sample sets has to be used to allow a mobile robot to handle both global localization and position tracking, which can be a waste of computational resources. *KDL-sampling* (Fox, 2003) is a variant of MCL that adapts the number of particles over time. It measures the difference between two probability distributions to determine the number of samples that, with probability $1-\delta$, the error between the true posterior and the sample-based approximation is less than ϵ.

Scan Matching Localization

The scan matching approach, when used for localization, tries to find the relative distance and rotation between a reference pose and the current pose of the robot, i.e. it addresses the pose tracking problem in localization. This is achieved by comparing one range scan acquired at the reference pose and one range scan taken at the actual pose of the robot. Furthermore, it is assumed that the actual pose is approximately known (i.e. from dead reckoning). This assumption reduces the search space of the scan-matching procedure. Let r_t denote the relation (i.e. relative translation and rotation) between the reference scan and actual scan calculated by the scan-matching algorithm. This relation r_t is then used to estimate the pose of the robot:

$$\bar{\mathbf{x}}_t = \mathbf{x}_{t-1} + r_t \ . \tag{2.27}$$

According to (Gutmann and Fox, 2002), given good inputs, a scan matching algorithm can localize a robot with relative precision, and in the linear case its estimation is optimal. However, it cannot recover from tragic failures caused by bad a bad matching of incorrect model of the environment. Nonetheless, because its search is bounded by small deviations of the sensor scans, it is computationally efficient.

The accuracy of the pose estimation is influenced by how the scan matching algorithm calculates the relation r_t between a scan and its reference and by the model used to represent the environment. Iterative Closest Point (ICP) (Besl and McKay, 1992; Thrun et al., 2003) and Iterative Closest Line (Bosse, 2004; Olson, 2008; Censi, 2008) methods are commonly used in scan matching. In ICP, each point in the current scan is associated with the reference

scan according to a distance metric (e.g. Euclidean distance). The goal is to calculate the rigid-body transformation that best aligns both scans (e.g. Horn, 1987).

Lu and Milios (1997a) describe two approaches for scan matching using the ICP method. The first separates both translation and rotation components. The main idea is to alternately fix one component and optimize the other. Given rotation, translations are optimized using least-squares. The optimization of the rotational components is conducted by the global-section method (Press et al., 1992). The second method, named iterative dual correspondence (IDC), combines two ICP-like methods with different scan-matching heuristic.

ICL is similar to ICP, however instead of matching query points with reference points, the query points are matched to lines extracted from the reference points. The motivation behind this approach is that the sensor samples the environment sparsely, and there may not be reasonable correspondence for all points. The simple ICL variant does a piecewise linear interpolation between each pair of adjacent range scan points. Others use heuristics methods to determine which pair of points are more likely to be part of a connected surface (Fischler and Bolles, 1981; Bosse, 2004; Olson, 2008).

There is a fair amount of scan matching heuristics in the literature. These include using Euclidean distance Lauer et al. (2006), polar coordinates (Diosi and Kleeman, 2007), the Normal Distribution Transformation (NDT) (Biber and Strasser, 2003), feature-based methods (Bosse, 2004; Olson, 2008), Hough transforms (Censi et al., 2005), histograms (Roferi, 2002; Bosse and Roberts, 2007), and correlative scan matching (Olson, 2015).

Most of the described scan matching approaches implies a *scan vs scan* matching. However, there is also the case where *scan vs map* matching is performed. For example, the work presented by Lauer et al. (2006) provide a robust real time self-localization, in a highly dynamic but structured environment, to a mobile robot. Instead of finding a relation r_t between the map and the current scan, it calculates the most likely pose $\mathbf{x}_t$ given the current measurement z_t. This method is also called maximum likelihood pose estimation (Thrun et al., 2001).

The *maximum likelihood* approach for pose estimation, although non-probabilistic, is simpler and easier to calculate than the posterior (2.3). The idea is simple: given a measurement and odometry reading, calculate the most likely pose. Mathematically speaking, the $\mathbf{x}_t$ pose is obtained as the maximum likelihood estimate of

$$\hat{\mathbf{x}}_t = \underset{\mathbf{x}_t}{\operatorname{argmax}}\, p(z_t|\mathbf{x}_t, m)\, p(\mathbf{x}_t|\hat{\mathbf{x}}_{t-1}, u_t) \tag{2.28}$$

To summarize, in time $t-1$ the (non-probabilistic) estimate of $\hat{\mathbf{x}}_{t-1}$ is given to the robot. As u_t is executed and a new z_t is obtained, the most likely pose $\hat{\mathbf{x}}_t$ is calculated by the robot.

In this approach $\hat{\mathbf{x}}_t$ is found using a gradient ascent algorithm in log likelihood space. It is common to maximize the log likelihood instead of the likelihood because it is mathematically easier to handle, and the maximization is justified by the fact that it is a strict monotonic function. Thus, this approach tries to calculate

$$\hat{\mathbf{x}}_t = \underset{\mathbf{x}_t}{\operatorname{argmax}}\, \ln[p(z_t|\mathbf{x}_t, m)\, p(\mathbf{x}_t|\hat{\mathbf{x}}_{t-1}, u_t)]\ . \tag{2.29}$$

Taking advantage of the properties of the logarithm, this expression can be decomposed into additive terms:

$$\hat{\mathbf{x}}_t = \underset{\mathbf{x}_t}{\operatorname{argmax}}\, \ln\, p(z_t|\mathbf{x}_t, m) + \ln\, p(\mathbf{x}_t|\hat{\mathbf{x}}_{t-1}, u_t)\ . \tag{2.30}$$

Obtaining the pose estimate is now a matter of using the proper method to solve the maximization problem. Notable methods are the gradient descent, Gauss-Newton or Levenberg-Marquard.

2.3 Mapping

The creation of models of the environment is a crucial part for many robotic application (Burgard and Hebert, 2008). These models of the environment are used by the robot so that is can adapt its decisions to the current state of the environment. In robotic localization it is assumed that that a model of the environment is available a priori – a model used in robotic localization is often called a *map*. In several real-world applications the existence of a map, usually built by hand, is assumed to be true. However, there are several application domains where a priori map is not available. Furthermore, a handmade map is usually inaccurate, for example, a blueprint of a building does not always correspond to the real structure of the building, and besides it does not consider furniture or any other item that shape the environment.

Having the capability to learn a map from scratch can enable the robot to adapt to changes without human supervision. Thus, taking one more step in the direction of truly autonomous mobile robots. The models are typically created from sensor data as the robot explores the environment. Mapping with mobile robots is a challenging problem due to a number of reasons:

- Maps are defined over a continuous space, therefore the space of all maps has an infinite number of dimensions. Even under discrete approximations that can help reduce the state space, map can easily be described by a huge number of variables. The Bayes filtering approach worked well for localization, but the problem of learning maps makes it unusable due to the high-dimensional space.

- Learning maps is a "chicken-and-egg" problem. First, when the robot moves through its environment it accumulates errors in odometry, gradually raising the uncertainty about its whereabouts in the environment. Localization algorithms are used to correct this errors, although, they required a known map of the environment. Second, constructing a map with known poses of the robot can be easy, however, when an initial map and exact pose information are not available the robot has to do both: learn the map and localize itself relative to this map. The process of learning maps are often referred to as the simultaneous localization and mapping (SLAM).

However, not all mapping problems are equally hard. The difficulty of the mapping problem lies in a collection of several factors, the more relevant are:

- **Size.** The larger the environment relative to the perceptual range of the robot, the more hard it is to build a map.

- **Noise in perception and actuation.** Building a map would be much easier if sensor and actuators were noise-free. The bigger the noise, the harder it is to build a map.

- **Perceptual ambiguity.** Different locations may look alike to the sensors of the robot, making it difficult to create a correspondence between different places traversed at different points in time.

- **Loops.** Loops (or cycles) in the configuration of the environment are specially difficult to map. Odometry errors can be minimized when going up and down a corridor, but cycles make robots return via different paths with a high accumulated odometric errors.

The difficulty of the SLAM problem is avoided by assuming that the exact robot path during mapping is known. This problem is also known as *mapping with known poses* and focuses on integrating sensor measurements into a representation of the environment. Feature-based representations is beyond the scope of the thesis, therefore we will discuss one of the most popular method for geometric mapping, the probabilistic occupancy grid.

2.3.1 Mapping with a Probabilistic Occupancy Grid

An *occupancy grid map* is a popular probabilistic approach to represent the environment (Moravec, 1989; Thrun et al., 2005). Mapping with occupancy grids addresses the problem of generating consistent maps from noisy and uncertain measurement data, assuming that the robot pose is known. The basic idea behind an occupancy grid is to represent the map as a grid of random variables. Each random variable is binary and represents the occupancy of the location it covers, and are used to estimate an approximate posterior estimation. The objective of mapping with an occupancy grid is to calculate the map posterior given the data

$$p(m|\mathbf{x}_{1:t}, z_{1:t}) \ , \tag{2.31}$$

where m is the map, $\mathbf{x}_{1:t}$ the sequence of all poses, and $z_{1:t}$ is the the set of all measurements up to time t. The control u_t is not necessary because the poses are already known.

Occupancy grid maps are fine-grained grids defined over the continuous space of locations. Their most common usage is to represent the two-dimensional (2D) flat surface that most mobile robots use to navigate. Occupancy grids can be extended to three-dimensional representation of the environment but at significant computational expenses. Let us denote m_i as the grid cell with index i. The space is partitioned by an occupancy grid into many grid cells

$$m = \{\mathbf{m}_i\} \ . \tag{2.32}$$

Each m_i corresponds to a binary random variable, which specifies whether a cell is occupied or free. Let $p(\mathbf{m} = \text{occupied})$ or $p(\mathbf{m})$ denote the probability that a grid cell is occupied, and $p(\mathbf{m} = \text{free})$ denote the probability that a grid cell is free. The posterior in Equation 2.31 is difficult to calculate when the number of grid cells is significantly high. For example, a grid with 10,000 cells has 2^{10000} possible maps. The approach of an occupancy grid is to divide the problem of estimating the map into a collection of separate problems, more specifically

$$p(\mathbf{m}_i|\mathbf{x}_{1:t}, z_{1:t}) \tag{2.33}$$

for all grid cell $\mathbf{m}_i$. Now, each estimation is a binary problem with static state. Furthermore, it is assumed that the individual cells of the grid are independent. Note that this is a rather strong assumption. It is a convenient decomposition but it does not represent dependencies among neighboring cells. In practice, there are objects larger than individual grid cells, like cabinets, chairs, etc. Thus, an occupied grid cell raises the probability of its neighboring cells to be occupied. Still, this assumption is adopted for convenience.

Under conditional independence assumption the estimation of the occupancy probability for each grid cell is a binary estimation problem with static static. The occupancy of a grid

cell is, in general, represented using *log odds*:

$$l_{t,i} = \ln \frac{p(\mathbf{m}_i|\mathbf{x}_{1:t}, z_{1:t})}{1 - p(\mathbf{m}_i|\mathbf{x}_{1:t}, z_{1:t})} \ . \tag{2.34}$$

The log odds representation offers numerical stability for probabilities near zero and one. The probabilities are easily recover from the log odds ration:

$$p(\mathbf{m}_i|\mathbf{x}_{1:t}, z_{1:t}) = 1 - \frac{1}{1 + \exp\{l_{t,i}\}} \ . \tag{2.35}$$

The occupancy grid mapping procedure in algorithm 11 iterates over all grid cells i, and updates those that fall into the sensor cone of the measurement z_t. For those where it does, the occupancy is updated by the virtue of occupancy grid mapping using the function **inverse_sensor_model()** in line 4. If not, the occupancy remains the same, as shown in line 6. The constant l_0 is the prior of occupancy in the form of log odds ratio:

$$l_0 = \ln \frac{p(\mathbf{m}_i = \text{occupied})}{p(\mathbf{m}_i = \text{free})} = \ln \frac{p(\mathbf{m}_i)}{1 - p(\mathbf{m}_i)} \ . \tag{2.36}$$

In practice, the value of the prior $p(\mathbf{m}_i)$ is assumed to be 0.5 which results in $l_0 = 0$. The inverse measurement model $p(\mathbf{m}_i|\mathbf{x}_{1:t}, z_{1:t})$ is implemented in its log odds form by the function **inverse_sensor_model()**:

$$\textbf{inverse\_sensor\_model}(\mathbf{m}_i, \mathbf{x}_t, z_t) = \ln \frac{p(\mathbf{m}_i|\mathbf{x}_t, z_t)}{1 - p(\mathbf{m}_i|\mathbf{x}_t, z_t)} \ . \tag{2.37}$$

As an example, the considered model model starts by calculating the beam index k and the range r for the center of mass of the cell $\mathbf{m}_i$ (lines 10 to 13). The pose of the robot is given by $\mathbf{x}_t = (x\ y\ \theta)^T$. It returns the prior for occupancy l_0 (in log odds) if the cell is outside the measurement range of the sensor beam, or is further than $\alpha/2$ the detect range z_t^k (line 14). It return l_{occ} if the range of the cell is within $\pm\alpha/2$ of the detected range z_t^k (line 16).

This model assigns to all cells within the sensor cone and with range near the measurement range an occupancy value of l_{occ} (line 14 and 15). The width of this region is defined by the parameter α, and the beam angle is controlled by the parameter β. It returns l_{free} if the range to the cell is smaller than the measurement range by a more than $\alpha/2$ (line 16).

2.4 Simultaneous Localization And Mapping

The *simultaneous localization and mapping* (SLAM) problem appears when the robot does not possess a map of the environment, nor it knows its own pose. Instead, only measurements $z_{1:t}$ and control $u_{1:t}$ are available to the robot. In SLAM, the robot constructs a map of its environment while simultaneously localizes itself relative to this same map. The SLAM problem is more difficult than the localization and mapping problems. It is hard to localize in a map that is unknown and has to be estimated along the way. And it is also hard to do mapping when the poses are unknown and have to be estimated along the way.

From a probabilistic point of view, the SLAM problem has two main forms: *online* SLAM and *full* SLAM (Thrun et al., 2005). In *online SLAM problem* the posterior is estimated over momentary poses along the map:

$$p(\mathbf{x}_t, m|z_{1:t}, u_{1:t}) \ . \tag{2.38}$$

Algorithm 11: Occupancy grid Mapping (Thrun et al., 2005).

Occupancy Grid Mapping($\{l_{t-1,i}, \mathbf{x}_t, z_t\}$ **):**
  for all cells $\mathbf{m}_i$ **do**
    if $\mathbf{m}_i$ *in perceptual field of* z_t **then**
      $l_{t_i} = l_{t-1,i} +$ **inverse_sensor_model**$(\mathbf{m}_i, \mathbf{x}_t, z_t) - l_0$
    else
      $l_{t_i} = l_{t-1,i}$
  end
  return $\{l_{t,i}\}$
inverse_sensor_model($\mathbf{m}_i, \mathbf{x}_t, z_t$ **):**
  Let x_i, y_i be the center-of-mass of $\mathbf{m}_i$
  $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
  $\phi = \text{atan2}(y_i - y, x_i - x) - \theta$
  $k = \text{argmin}_j |\phi - \theta_{j,sens}|$
  if $r > min(z_{max}, z_t^k + \alpha/2)$ *or* $|\phi - \theta_{k,sens}|$ **then return** l_0
  if $z_t^k < z_{max}$ *and* $|r - z_t^k| < \delta/2$ **then return** l_{occ}
  if $r \leq z_t^k$ **then return** l_{free}

Here $\mathbf{x}_t$ is the pose at time t, m is the map, $z_{1:t}$ the measurements, and $u_{1:t}$ the controls. This problem is designated online SLAM because it only tries to estimate the variables that persist at time t. Typical online SLAM approaches are incremental, past measurements and controls are discarded once they have been processed. In *full SLAM* the posterior is calculated over the entire path $\mathbf{x}_{1:t}$ along with the map, instead of just the current pose $\mathbf{x}_t$:

$$p(\mathbf{x}_{1:t}, m | z_{1:t}, u_{1:t}) \ . \tag{2.39}$$

Any SLAM problem has to deal with continuous and discrete components. The pose of the robot and the location of the objects in the map are part of the continuous estimation problem. Objects are usually landmarks in feature-based maps, and patches detected by range finders in grid-based maps. The discrete nature of a SLAM problem lies on the correspondence problem of detected objects, which is typically discrete: either the object is the same as previously detected one, or it is not.

In practice, the calculation of the full posterior is usually unfeasible, the high dimensionality of the continuous parameters and the large number of discrete correspondence variables are the main causes. In contrast, localization problems calculate posteriors over a three-dimensional continuous space, and in most application the correspondences are unknown making the number of possible assignments to the vector of all correspondences grow exponentially in time t. Therefore, practical SLAM approaches rely on approximations to cope with the correspondence problem.

2.4.1 Feature-based SLAM

The EKF-SLAM is one of the earliest and most influential feature-based SLAM algorithm (Leonard and Durrant-Whyte, 1991; Davison et al., 2007; Castellanos et al., 2007; Jesus and Ventura, 2012). Based on the Extended Kalman Filter (EKF), it is used in SLAM to estimate the pose of the robot and the location of features in the environment. For now, let us assume that all correspondences are known. It is a very strong assumption, but otherwise all potential data association problems had to be handled, increasing the complexity of the problem.

The state vector is a combination of the pose of the robot and landmarks coordinates

$$\mathbf{x}_t = (\underbrace{x, y, \theta}_{\text{robot pose}}, \underbrace{m_{1,x}, m_{1,y}}_{\text{landmark}_1}, \ldots, \underbrace{m_{n,x}, m_{n,y}}_{\text{landmark}_n})^T . \tag{2.40}$$

Here x, y, θ denote the pose of the robot at time t, $m_{n,x}, m_{n,y}$ are the coordinates of the n-th mark. For convenience, let r_t denote the pose of the robot, and m_n the vector of landmark coordinates. For a map with n landmarks the state is represented by a $(3 + 2n)$-dimensional Gaussian with mean μ_t and covariance Σ_t

$$\mu_t = \begin{bmatrix} r_t \\ m_n \end{bmatrix} \qquad \Sigma_t = \begin{bmatrix} \Sigma_{r_t r_t} & \Sigma_{r_t m_n} \\ \Sigma^T_{r_t m_n} & \Sigma_{r_t m_n} \end{bmatrix} \tag{2.41}$$

As shown above, the covariance Σ_t can be decomposed into four covariances. The first, is the robot pose covariance $\Sigma_{r_t r_t}$. The second is the landmarks covariance $\Sigma_{m_n m_n}$. The third is the covariance between the robot pose and the landmarks $\Sigma_{r_t m_n}$, and the fourth covariance is the transpose matrix of $\Sigma_{r_t m_n}$.

This type of solution is well known and inherits many of the same benefits and problems of the basic EKF algorithm for navigation and tracking problems:

Convergence: The convergence of the map is observed in the monotonic convergence of the determinant of the map covariance matrix $\Sigma_{mm,t}$ and all landmark pair sub-matrices toward zero. The variance of individual landmarks converge to a lower bound defined by initial uncertainties in robot pose and observations (Dissanayake et al., 2001).

Computational effort: The measurement update step requires the calculation of the joint covariance matrix and landmarks to be updated every time an measurement is taken, which requires a exponential growth as the number of landmarks increase. Fortunately, there are methods that allow these calculations in real-time (Leonard and Feder, 1999; Guivant and Nebot, 2001).

Data association: The EKF-SLAM is fragile to incorrect associations of measurements to landmarks (Neira and Tardos, 2001) (the *loop closure* is specially difficult in this solution).

Non linearity: The EKF-SLAM relaxes the non-linearity problem by employing a linearization to non-linear motion and measurement models. This can lead to inevitable inconsistencies in solutions (Julier and Uhlmann, 2001).

The FastSLAM algorithm (Montemerlo et al., 2002) is another featured-based solution that combines EKF for landmark estimation with a particle filter for estimating the robot's path. The use of particles filters puts the FastSLAM algorithm in the unusual situation where it solves both the *online SLAM problem* and *full SLAM problem*. The approach is formulated to calculate the full path posterior, however, particle filters estimate on pose at-a-time. Each particle, in FastSLAM, contains an estimated robot pose $\mathbf{x}_t^{[k]}$ and a set of Kalman filters with mean $u_t^{[k]}$ and covariance $\Sigma_t^{[k]}$, one for each feature m_j in the map. The index of the particle is denoted by $[k]$ and the total number of particles by M. The FastSLAM algorithm can be represented by a series of steps:

- Execute the following M times:

- **Retrieval.** Retrieve a pose $\mathbf{x}_{t-1}^{[k]}$ from the particle set $\mathcal{S}_{t-1}$.
- **Prediction.** Sample a new pose $\mathbf{x}_t^{[k]} \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}^{[k]}, u_t)$.
- **Measurement update.** For each feature observed z_t^i find the correspondence j for the measurement z_t^i, then incorporate the measurement z_t^i into the corresponding EKF, by updating the mean $u_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$.
- **Importance weight.** Calculate the importance weight $w_t^{[k]}$.

- **Resampling.** Sample, with replacement, M particles, where each particle is sampled with a probability proportional to $w^{[k]}$.

The high dimensional state space of the SLAM problem makes a straightforward application of particle filters computationally unfeasible. However, sample space can be reduced by applying *Rao-Blackwellization* (RB) to represent the posterior over some variables, along with Gaussians (or other parametric PDF) to represent all other variables. The joint state is partitioned according the product rule

$$p(\mathbf{x}_1, \mathbf{x}_2) = p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1) \ . \tag{2.42}$$

If $p(\mathbf{x}_2|\mathbf{x}_1)$ can be represented analytically, only $p(\mathbf{x}_1)$ need to be sampled $\mathbf{x}_1 \sim p(\mathbf{x}_1)$. As result, the joint distribution is represented by the set $\{\mathbf{x}_1^i, p(\mathbf{x}_2|\mathbf{x}_1)\}_i^N$ and statistics such as the marginal

$$p(\mathbf{x}_2) \approx \frac{1}{N}\sum_i^N p(\mathbf{x}_2|\mathbf{x}_1^i) \ . \tag{2.43}$$

Using the mathematical insight of RB, the full SLAM posterior $p(\mathbf{x}_{1:t}|z_{1:t}, u_{1:t})$ can be written in the factored form

$$p(\mathbf{x}_{1:t}|z_{1:t}, u_{1:t}) = p(\mathbf{x}_{1:t}|z_{1:t}, u_{1:t}, c_{1:t}) \prod_{i=1}^{N} p(m_i|\mathbf{x}_{1:t}, z_{1:t}, c_{1:t}) \ . \tag{2.44}$$

This *factorization* allows the calculation of the posterior over path and maps to be decomposed into $N+1$ probabilities. The features estimator is now conditioned on the robot path, which means that there is a separate copy of each feature estimator, one for each particle. This type of particle filters are commonly known as *Rao-Blackwellize particle filters* (RBPF).

The presented algorithm is known by the literature as FastSLAM 1.0. The later version, FastSLAM 2.0 (Montemerlo et al., 2003), differs from the former only in terms of the form of their proposal distribution, and consequently the importance weight. The proposal distribution in FastSLAM 1.0 is modeled by

$$\mathbf{x}_t^{[k]} \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}^{[k]}, u_k) \ , \tag{2.45}$$

and the importance weight is calculated by

$$w_t^{[k]} = w_{t-1}^{[k]} p(z_t|\mathbf{x}_{1:t}^{[k]}, z_{1:t}, u_t) \ . \tag{2.46}$$

For FastSLAM 2.0, the proposal distribution includes the current measurement

$$\mathbf{x}_t^{[k]} \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) \ , \tag{2.47}$$

where

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}^{[k]}, u_{1:t}, z_{1:t}, c_{1:t}) = \eta^{[k]} p(z_t|\mathbf{x}_t, \mathbf{x}_{1:t-1}^{[k]}, z_{1:t-1}, u_{1:t}, c_{1:t}) \\ p(\mathbf{x}_t|\mathbf{x}_{1:t-1}^{[k]}, z_{1:t-1}, u_{1:t}, c_{1:t}) \tag{2.48}$$

and $\eta^{[k]}$ is a normalizer. The importance weight is $w_t^{[k]} = w_{t-1}\frac{1}{\eta^{[k]}}$. As consequence, the FastSLAM 2.0 is locally optimal (Doucet, 1998)

2.4.2 Grid-based SLAM

Grid-based maps have the advantage of representing the environment of the robot without any predefined definition of landmarks, and arbitrary types of environments can be modeled. The work of Lu and Milios (1997b) introduces a scan matching method for geometric mapping. It considers the translation and rotation as two separate components, alternatively fixing one and optimizing the other and then swapping the role of components. The translation is optimized using least squares and the rotation by global-section method. This approach has sufficient accuracy with small computational effort but it is generally affected by an increasing error, visible whenever known areas are revisited. The Normal Distribution Transform (NDT) (Biber and Strasser, 2003) assigns to each cell of the grid map a normal distribution that models the probability of measuring a point. It performs scan matching by building the NDT of the previous scan and then it finds the rigid transformation with the best score using the current scan. The optimization is performed using Newton's algorithm. Furthermore, it uses four overlapped grids to reduce the effect of discretization. Holz and Behnke (2010) propose a fast ICP-based method for SLAM based on incremental scan registration and *Sparse Point Maps*. To improve registration and maintain scalability, they present several heuristics for rejecting bad correspondence pairs and duplicated points in the sparse map. However, this solution has a tendency to increase in computational complexity as the sparse map grows bigger due to the need of having to search for correspondences. Kohlbrecher et al. (2011) propose a SLAM system based on scan matching with full 3D estimation. It is based on the optimal alignment of a scan with the map learned so far. The optimization is solved using the Gauss-Newton method. To mitigate the problem of getting stuck in a local minimum they pursue a multi-resolution map representation. Furthermore, they introduce a bilinear filtering in their maps to reduce the side effects of discretization. The Critical Rays Scan Match SLAM (Tsardoulias and Petrou, 2013) is a scan matching approach with hill climbing optimization. To reduce the number of rays used during optimization a set of heuristics is to select only rays that are considered to be critical to the scan matching between the current scan and the map. The work of Hess et al. (2016) proposes a grid-base SLAM solution based on scan matching combined with graph optimization of sub-maps for loop closure. The scan matching procedure is similar to Kohlbrecher et al. (2011) but only uses a single occupancy map.

The Rao-Blackwellized Particle Filter (RBPF) for SLAM Grisetti et al. (2005, 2007a) is an application of the FastSLAM algorithm to occupancy grid mapping. It estimates the posterior over trajectories and maps using a particle filter and each particle represents a map m^i and an estimate of the robot pose x_t^i on that map. To improve the proposal distribution, they perform scan matching to determine the mode of the meaningful area of the observation likelihood function.

2.4.3 Graph-based SLAM

Solving the SLAM problem using a graph based formulation was proposed by Lu and Milios (1997a). At that time solving error minimization problems using the available methods was deemed complex. It took several years until advances in the field of sparse linear algebra to resulted in efficient approaches to the optimization problem at hand. Consequently, graph-based SLAM approaches have experienced a rise in popularity (Dellaert and Kaess, 2006; Kaess et al., 2007; Grisetti et al., 2009, 2010b,b; Konolige et al., 2010; Kaess et al., 2012) and have become a state-of-the-art techniques with respect to speed and accuracy.

The graph-based SLAM solves the *full SLAM problem* by constructing a simplified estimation problem by creating an abstraction of the raw sensor measurements, i.e. "virtual measurements". An edge between two nodes is labeled with a distribution over the relative positions of the two poses, conditioned to currents observations. Normally, the measurement model $p(z_t|\mathbf{x}_t, m)$ is multi-modal, thus the Gaussian assumption falls through. This means that a measurement z_t might result in several possible edges connecting different poses in the graph and the graph connectivity needs to be modeled by a probability distribution.

Instead of directly dealing with this multi-modality in the estimation process, which would result in an combinatorial increase of the complexity, the estimate is restricted to the most likely topology. Therefore, the most likely constraint resulting from a measurement needs to be calculated. This calculation depends on the probability distribution over the poses of the robot. This problem is known as data association. To find the correct data association an estimate of the conditional prior over the robot path $p(\mathbf{x}_{1:t}|z_{1:t}, u_{1:t})$ is required. Methods to efficiently solve the data association problems will not be discussed. Such methods deal associations by means of spectral clustering (Olson et al., 2006), joint compatibility branch and bound (Neira and Tardos, 2001), or backtracking (Hähnel et al., 2003).

If the data association is known and the local measurements are affected by (locally) Gaussian noise, then the goal of a graph-based mapping method is to compute a Gaussian approximation of the posterior over the robot path. This process involves calculating the mean of this Gaussian as the configuration of the nodes that maximizes the likelihood of the measurements. This task is characterized by finding this maximum likelihood as a constraint optimization problem.

Let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_t)^T$ denote the vector of all poses, where $\mathbf{x}_i$ describes the pose of node i. Let z_{ij} and Ω_{ij} denote respectively the mean and information matrix of a virtual measurement between the nodes i and j. This virtual measurement represents a transformation that maximizes the overlapping of measurements acquired at node i with measurements acquired at node j. Let $\hat{z}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ denote the prediction of a virtual measurement given a configuration of the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. Generally this prediction is the relative transformation between the two nodes. Therefore, the log likelihood l_{ij} of a measurement is

$$l_{ij} \propto [z_{ij} - \hat{z}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]^T \Omega_{ij} [z_{ij} - \hat{z}_{ij}(\mathbf{x}_i, \mathbf{x}_j)] \; . \tag{2.49}$$

Let $e(\mathbf{x}_i, \mathbf{x}_j, z_{ij})$ denote a function that calculates the difference between the expected observation $\hat{z}_{ij}$ and the real observation z_{ij} acquired by the robot. To simplify the notation, the indices of the measurements are encoded in the indices of the error function

$$e_{ij}(\mathbf{x}_i, \mathbf{x}_j) = z_{ij} - \hat{z}_{ij}(\mathbf{x}_i, \mathbf{x}_j) \; . \tag{2.50}$$

Let $\mathcal{C}$ denote the set of pair of indices for which a constraint (measurement) z exists. The objective of the maximum likelihood approach is to find the configuration of nodes $\mathbf{x}^\star$ that

minimizes the negative log likelihood $F(\mathbf{x})$ of all observations

$$F(x) = \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{e_{ij}^T \Omega_{ij} e_{ij}}_{F_{ij}} , \tag{2.51}$$

therefore, it tries to solve the following equation:

$$\mathbf{x}^\star = \underset{\mathbf{x}}{\operatorname{argmin}} F(\mathbf{x}) . \tag{2.52}$$

Solving the equation above is a matter of applying adequate optimization methods. For an example please refer to Grisetti et al. (2010a).

Chapter 3

Scan Matching Approach to Localization with a Likelihood Field

The application of scan matching to mobile robot localization relies on matching a set of points $\{\mathbf{p}_i\}$ against a reference surface $\mathcal{S}^{\text{ref}}$, in order to find the roto-translation $\mathbf{q} = (\theta, t)$ that minimizes a function of the distances of the roto-translated points $\{\mathbf{q} \oplus \mathbf{p}_i\}$ to their projection on $\mathcal{S}^{\text{ref}}$. Here, $\oplus$ is the roto-translation operator, such that

$$(\theta, t) \oplus \mathbf{p}_i \triangleq R(\theta)\mathbf{p}_i + t \ , \tag{3.1}$$

where $R(\theta)$ is the rotation matrix. Using an Euclidean projector on $\mathcal{S}^{\text{ref}}$, denoted $\Pi(\mathcal{S}^{\text{ref}}, \cdot)$, the minimization problem can be defined by

$$\min_{\mathbf{q}} \sum_i \left\| \mathbf{q} \oplus \mathbf{p}_i - \Pi(\mathcal{S}^{\text{ref}}, \mathbf{q} \oplus \mathbf{p}_i) \right\|^2 \ . \tag{3.2}$$

To solve this minimization problem, several Iterative Closest Point (ICP) variants have been proposed (Lu and Milios, 1997a; Pfister et al., 2002; Montesano et al., 2005; Censi, 2008). But, as noted by Burguera et al. (2009), they may suffer from two important drawbacks: incorrect correspondences from the projector $\Pi(\mathcal{S}^{\text{ref}}, \cdot)$ and poor convergence as a side effect of inconsistent correspondences.

In this chapter, we propose a *fast* scan matching approach with a *likelihood field* as measurement model. The likelihood field has a central role in this work: it addresses the drawbacks of the ICP algorithm, by avoiding the direct establishment of correspondences (Burguera et al., 2009); and it is the element that connects scan matching to mobile robot localization. Once the scan matching formulation is in place we will solve it by using *non-linear least squares*.

The *likelihood field* (LF) (Thrun, 2001) was designed as a measurement model that provides smoother results and is computationally more efficient than the *beam model* (Thrun, 2001). Efficiency is achieved by caching the likelihood values of the reference surface $\mathcal{S}^{\text{ref}}$ in a discrete volumetric grid. Access to the values is provided by a function

$$l : \mathbb{R}^n \to \mathbb{R} \tag{3.3}$$

such that $l(\mathbf{p}_i)$ returns the likelihood of the point $\mathbf{p}_i$, where $\mathbf{p}_i$ can either be defined in 2D or 3D space depending on the application.

3.1 Maximum Likelihood Pose Estimation

Let $\mathbf{x}_t$ be the robot state and z_t the measurements at time t. The control u_t at time t determines the change of state from $t-1$ to t. The distribution that models the robot pose estimate is given by

$$p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, z_t, u_t, m) \propto p(z_t \mid \mathbf{x}_t, m)\; p(x_t \mid \mathbf{x}_{t-1}, u_t) \tag{3.4}$$

The first term, $p(z_t|\mathbf{x}_t, m)$, is the *measurement model* that expresses how likely is a particular measurement if the robot's pose is known. The term $p(x_t|\mathbf{x}_{t-1}, u_t)$ is the *motion model* that describes the probability of the pose $\mathbf{x}_t$ after executing the control u_t.

Our proposal to solve the localization problem is based on the calculation of the maximum likelihood of $p(z_t|\mathbf{x}_t, m)$. The motion model is used as an initial hint for $\mathbf{x}_t$, and treated as a constant. This assumption is valid because we are considering a local search approach, and the accurate measurements from a LIDAR sensor cause the product of (3.4) to be dominated by the observation likelihood (Grisetti et al., 2007a). Thus, the pose estimation can be reduced to

$$\hat{x}_t = \underset{\mathbf{x}_t}{\operatorname{argmax}}\, p(z_t \mid \mathbf{x}_t, m)\ . \tag{3.5}$$

The measurement z_t is defined by a set of points $\{\mathbf{p}_t^i\}$, thus, the resulting calculation is the collection of probabilities $p(\mathbf{p}_t^i|\mathbf{x}_t, m)$, where $\mathbf{p}_t^i$ is the ith measurement. Assuming conditional independence between measurements, the measurement model probability is given by:

$$p(z_t \mid \mathbf{x}_t, m) = \prod_i p(\mathbf{p}_t^i \mid \mathbf{x}_t, m)\ . \tag{3.6}$$

Introducing (3.6) into (3.5) the maximum likelihood estimate can be rewritten as

$$\hat{\mathbf{x}}_t = \underset{\mathbf{x}_t}{\operatorname{argmax}} \prod_i p(\mathbf{p}_t^i \mid \mathbf{x}_t, m)\ . \tag{3.7}$$

As it can be numerically unstable to solve (3.7), a usual approach is to use the log-likelihood. The problem can now be formulated as a minimization of the negative log-likelihood of our measurement model:

$$\hat{\mathbf{x}}_t = \underset{\mathbf{x}_t}{\operatorname{argmin}} \sum_i -\ln\left[p(\mathbf{p}_t^i \mid \mathbf{x}_t, m)\right]\ . \tag{3.8}$$

The next step is to select a proper measurement model for the pose estimation, and our choice is the likelihood field.

3.2 Likelihood Field as Measurement Model

Because we are dealing with measurements taken from a LIDAR sensor, the calculation of the *probability density function* (PDF) of each sample z_t should, in principle, consider which surface of the map is visible from the pose $\mathbf{x}_t$. The *beam model* (Thrun, 2001) does this by applying an expensive ray-casting operation. To avoid this expensive operation we adopted the LF model that, although neglecting visibility and occlusion effects, if constructed properly, can give smoother results at a lower computational cost (Thrun, 2001). The LF is defined as a function of x–y-coordinates expressing the likelihood of object detection (Thrun,

2001). Although not a proper PDF, we will demonstrate its use in pose estimation with scan matching, more specifically, how it can be used as the measurement model in localization.

Let us define the LF function $l(\mathbf{p})$ as:

$$l(\mathbf{p}) = \exp\left(-\|\mathbf{p} - \Pi(m, \mathbf{p})\|^2 \sigma^{-2}\right) \tag{3.9}$$

where $\mathbf{p}$ is a point in the map, $\Pi(m, \cdot)$ is the Euclidean projector on the map m to the closest object and σ is a scaling factor that guarantees the exponent is dimensionless. To be a proper LF, $l(\mathbf{p})$ must have the following property: the closer a point $\mathbf{p}$ is to its projection $\Pi(m, \mathbf{p})$ the more likely it is to be measured. This property is fulfilled by the fact that the value of $l(\mathbf{p})$ is the exponent of the negative squared Euclidean distance between $\mathbf{p}$ and the closest object in m, which increases in value as the Euclidean distance decreases.

We can realize that (3.9) corresponds to the non normalized PDF of a Gaussian distribution with respect to to the Euclidean distance. Turning the LF function into a PDF is a matter of introducing a normalization factor η:

$$\begin{aligned} f(\mathbf{p}) &= \eta l(\mathbf{p}) \\ &= \eta \exp(-\varepsilon^2(\mathbf{p})) \end{aligned} \tag{3.10}$$

with

$$\varepsilon(\mathbf{p}) = \|\mathbf{p} - \Pi(m, \mathbf{p})\| \sigma^{-1} \tag{3.11}$$

as the Euclidean distance to the nearest neighbor. This new obtained PDF (3.10) can now be used as our measurement model. Assuming that all LIDAR scans are independent, the measurement model for our maximum likelihood pose estimation is given by

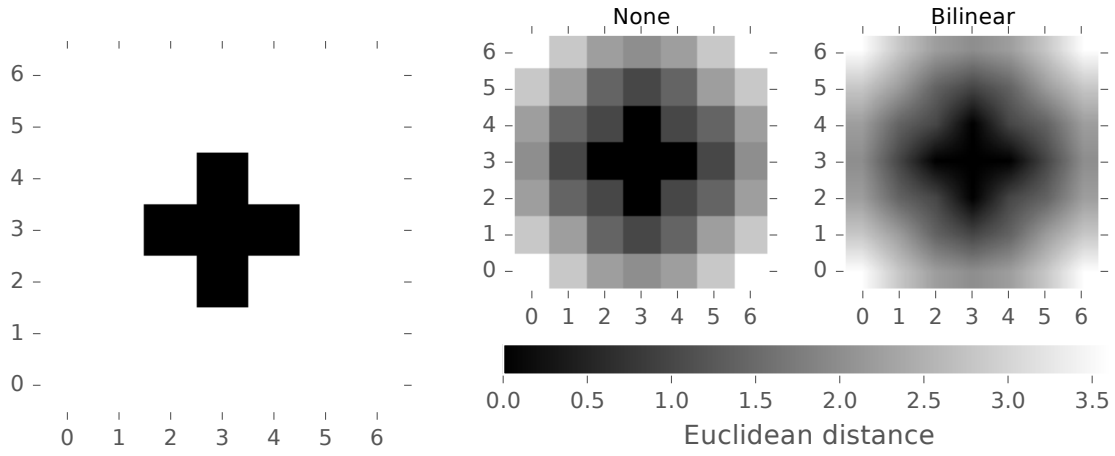
$$p(z_t \mid \mathbf{x}_t, m) = \prod_i \eta \exp\left(-\varepsilon^2(\mathbf{x}_t \oplus \mathbf{p}_t^i)\right) \ . \tag{3.12}$$

3.2.1 Continuous Likelihood Field

The low computational cost of the likelihood field results from the discretization of the environment in the form of a grid where each cell contains pre-computed likelihood of a measurement. Kohlbrecher et al. (2011) noted that discrete representations limit the precision that can be achieved. As an improvement, they propose the use of an interpolation scheme that allows sub-cell accuracy through bilinear filtering. The same interpolation scheme is applied to the Euclidean distance grid $\hat{\varepsilon}$, the discrete version of the Euclidean distance ε that is part of the likelihood field. Given a continuous coordinate $\mathbf{p}_\varepsilon = [x, y]$ in grid coordinates with $x, y \in [0, 1)$, the value of the Euclidean distance is computed from the four closest integer coordinates $\mathbf{p}_{00..11}$. It should be noted that the cells of the grid are organized in a regular grid with spacing equal to 1 in grid coordinates. The effective formula for value interpolation yields

$$\hat{\varepsilon}(x, y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} \varepsilon(\mathbf{p}_{00}) & \varepsilon(\mathbf{p}_{01}) \\ \varepsilon(\mathbf{p}_{10}) & \varepsilon(\mathbf{p}_{11}) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix} \ . \tag{3.13}$$

Inherent to this scheme is that values from the grid cells are now samples from a continuous function. Now, instead of pre-computing the likelihood for each cell, only the Euclidean distance grid is pre-computed and further used to calculate the likelihood as a continuous function, an example being depicted in Figure 3.1. The necessary gradients used for optimization



(a) (b)

Figure 3.1: Interpolation scheme for a continuous likelihood field. a) Example of an environment occupancy map that models free space and obstacles. b) Corresponding distance map, here two schemes are employed, one without interpolation that represents the typical discrete Euclidean distance map, the second scheme employs a bilinear interpolation on top of the discrete distance map.

are calculated using numerical differentiation whilst considering the values of the coordinates $\mathbf{p}_{00..11}$, which are the coordinates of its neighbors. The Jacobian of the Euclidean distance, denoted $\mathbf{J}_\varepsilon$, is calculated as follow:

$$\mathbf{J}_\varepsilon = -\frac{1}{r}\begin{bmatrix}(1-y)(\varepsilon(\mathbf{p}_{00})-\varepsilon(\mathbf{p}_{10}))+y(\varepsilon(\mathbf{p}_{01})-\varepsilon(\mathbf{p}_{11}))\\(1-x)(\varepsilon(\mathbf{p}_{00})-\varepsilon(\mathbf{p}_{01}))+x(\varepsilon(\mathbf{p}_{10})-\varepsilon(\mathbf{p}_{11}))\end{bmatrix}^T \tag{3.14}$$

where r is the resolution of the Euclidean distance grid.

3.3 Non-Linear Least Squares Optimization

The use of *non-linear least squares* to solve the minimization problem introduced by the formal definition of scan matching in (3.2) is a common approach, as, by definition, *least squares* tries to minimize the sum of the squared errors of our objective function w.r.t its parameters. To obtain the estimation of $\mathbf{x}_t$ we need to solve the minimization (3.8). However, it lacks the least squares formulation found in the scan matching definition of (3.2), and subsequently it can not be solved using least squares optimization. Therefore, minimization (3.8) needs to be converted to an equivalent least squares formulation. This can be achieved by replacing the measurement model in (3.8) with the measurement model based on the LF:

$$\begin{aligned}\hat{\mathbf{x}}_t &= \underset{\mathbf{x}_t}{\operatorname{argmin}} \sum_i -\ln\left[\eta\exp(-\varepsilon^2(\mathbf{x}_t\oplus\mathbf{p}_t^i))\right]\\ &= \underset{\mathbf{x}_t}{\operatorname{argmin}} -n\ln\eta+\sum_i\varepsilon^2(\mathbf{x}_t\oplus\mathbf{p}_t^i)\\ &= \underset{\mathbf{x}_t}{\operatorname{argmin}} \sum_i\varepsilon^2(\mathbf{x}_t\oplus\mathbf{p}_t^i)\ .\end{aligned} \tag{3.15}$$

The terms $-n \ln \eta$ can be safely ignored because it is a constant and thus it does not influence the minimization. Up until now, in this chapter, localization and scan matching had been treated as two different concepts. The introduction of the LF into the localization problem has the consequence of converting the localization problem into a scan matching problem and, as a result, it can now be solved using least squares. Our approach tries to find the pose $\hat{\mathbf{x}}_t$ that best aligns the endpoints $\mathbf{p}_t$ of the scan with the local surface of the map m. Due to the use of the LF with an distance map as the Euclidean projector, there is no need to find correspondences for the endpoints. Also, because scans are aligned with the map m, there is an implicit matching with all previous scans.

A common approach to solve a *non-linear least squares* optimization is to use the Gauss-Newton (GN) method (Madsen et al., 2004). The method consists in iteratively updating the pose estimate with the rule

$$\hat{\mathbf{x}}_t^{i+1} = \hat{\mathbf{x}}_t^i + \Delta\mathbf{x} \,, \tag{3.16}$$

where at each iteration the update step $\Delta\mathbf{x}$ is obtained by solving the *normal equation*:

$$(\mathbf{J}^T\mathbf{J})\Delta\mathbf{x} = -\mathbf{J}^T\varepsilon(\hat{x}_t \oplus \mathbf{p}_t) \tag{3.17}$$

with

$$\mathbf{J} = \frac{\partial\varepsilon(\hat{\mathbf{x}}_t \oplus \mathbf{p}_t)}{\partial\hat{\mathbf{x}}_t \oplus \mathbf{p}_t}\frac{\partial\hat{\mathbf{x}}_t \oplus \mathbf{p}_t}{\partial\hat{\mathbf{x}}_t}$$

The method stops when convergence is found or when a maximum number of iterations is reached. The initial hint of $\hat{\mathbf{x}}_t^{i=1}$ is provided by the previous estimate $\hat{\mathbf{x}}_{t-1}$ plus the displacement of the odometry between $t-1$ and t. To prevent unnecessary estimates, e.g. when there is no motion, the estimate of the pose is only carried when a certain amount of motion exists, usually provided by the odometry u_t. The motion is divided into its translation ρ and rotational ϑ components. The pose estimation is only triggered when a threshold is reached in any of the components. A variant of GN, called Levenberg-Marquardt[1] (LM) (Madsen et al., 2004), is also used as an alternative method to obtain the estimates. It changes the normal equation as follows:

$$(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\Delta\mathbf{x} = -\mathbf{J}^T\varepsilon(\hat{\mathbf{x}}_t \oplus \mathbf{p}_t) \,, \tag{3.18}$$

where μ is a *damping* parameter that affects the optimization. For all $\mu > 0$, it is guaranteed that $\Delta\mathbf{x}$ is in the direction of the steepest descent. If $\mu \to 0$, the method falls back into GN, whereas if $\mu \to \infty$ the gradient descent is used. Furthermore, the update $\mathbf{x}_t + \Delta\mathbf{x}$ is only performed if there is a significant residual error reduction. Upon success, μ is decreased, otherwise μ is increased. The reasoning behind this heuristic is that GN has quadratic convergence near the minimum, but gradient descent has a better behavior when far from the minimum.

3.4 Least Squares on a Manifold

In localization, the configuration space (i.e. the robot's pose $\hat{\mathbf{x}}$) is not Euclidean. The translation component of $\hat{\mathbf{x}}$ forms an Euclidean space, however the rotation component spans over the non-Euclidean rotation group. For example, the two-dimensional rotation is represented by an angle θ bounded by $\theta \in [-\pi, \pi)$. Adding small changes to θ, which happens in

[1]The precise name of the method presented in Equation 3.18 is *Levenberg*, however we follow the nomenclature present in Madsen et al. (2004) where the method is named *Levenberg-Marquardt*.

every iteration of the optimization, is not an issue as the angle can be normalized each time it is changed. However, the comparison between two angles near their limits $-\pi$ and π, results in a difference of almost 2π although their difference is close to zero.

A common approach to numerically handle non-Euclidean spaces is to perform the optimization on a manifold. A manifold is a mathematical space that is not necessarily Euclidean from a global perspective, but can be seen as Euclidean locally (Lee, 2003; Hall, 2015). Any rigid body transformation in $\mathbb{R}^n$ can be expressed as a $(n+1) \times (n+1)$ homogeneous matrix

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \quad \text{with} \quad \mathbf{R} \in SO(n), \quad \mathbf{t} \in \mathbb{R}^n \,, \tag{3.19}$$

where $SO(n)$ is the Lie group of rotation matrices, the *Special Orthogonal* group. The rigid body transformation is also a Lie group, the *Special Euclidean* group $SE(n)$, that forms a smooth manifold in $\mathbb{R}^n$. To avoid singularities when dealing with angles, these spaces are usually expressed in an over parameterized way, for example the groups $SO(2)$ and $SO(3)$ are represented by a unit complex and a normalized quaternion, respectively. Increasing the number of parameters to optimize is not desirable, therefore, the optimization is performed in the corresponding Lie algebra $\mathfrak{se}(n)$, which is the minimal representation of $SE(n)$ and also its tangent space at the identity. The Lie algebra $\mathfrak{se}(n)$ can also be understood as the linearization of the Lie group $SE(n)$.

Since the optimization is now carried on $\mathfrak{se}(n)$ it is necessary to introduce an operator $\boxplus$ that maps a local variation $\Delta\mathbf{x}$ in the Euclidean space onto a pose in the manifold space, $\boxplus : SE(n) \times \mathfrak{se}(n) \to SE(n)$. This mapping leads to the modified versions of (3.17) and (3.16) in the least squares definition:

$$(\breve{\mathbf{J}}^T \breve{\mathbf{J}}) \Delta\mathbf{x} = -\breve{\mathbf{J}}^T \varepsilon(\hat{\mathbf{x}}_t^i \oplus \mathbf{p}_t) \tag{3.20}$$

$$\hat{\mathbf{x}}_t^{i+1} = \hat{\mathbf{x}}_t^i \boxplus \Delta\mathbf{x} \,, \tag{3.21}$$

with

$$\breve{\mathbf{J}} = \frac{\partial \varepsilon((\hat{\mathbf{x}}_t \boxplus \Delta\mathbf{x}) \oplus \mathbf{p}_t)}{\partial \Delta\mathbf{x}} \Big|_{\Delta\mathbf{x}=0} \tag{3.22}$$

Now, because the optimization is performed in the origin frame of the tangent space, the Jacobian of the robot-translation is the Jacobian of a rigid body transformation at the origin, which simplifies its calculation. In the context of two-dimensional localization, the Jacobian $\breve{\mathbf{J}}$ is a stack matrix formulated as

$$\breve{\mathbf{J}}^i = \mathbf{J}_\epsilon^i \begin{bmatrix} 1 & 0 & -y^i \\ 0 & 1 & x^i \end{bmatrix} \tag{3.23}$$

where $\langle x^i, y^i \rangle$ are the Cartesian coordinates of the measurement $\mathbf{p}_t^i$ and $\mathbf{J}_\epsilon^i$ is the Jacobian of the Euclidean distance used by the likelihood field (see 3.13).

3.5 Handling Outliers

The incorrect modelling of the measurement likelihood will likely create disturbances in the maximum likelihood estimate (Triggs et al., 2000), such as the maximum likelihood pose

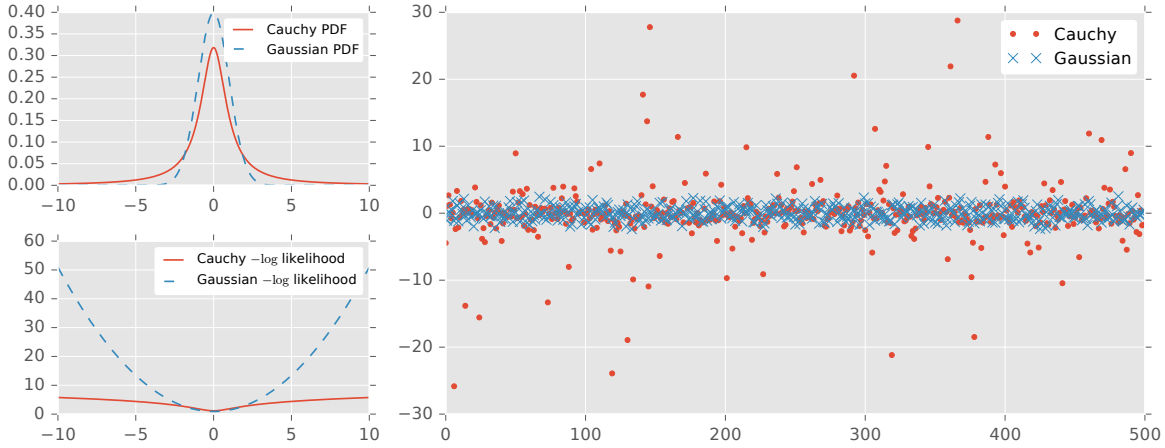


Figure 3.2: Not all bell-shaped measurement distributions are Gaussians (Triggs et al., 2000). The Cauchy distribution, although with a narrower peak and broader tail, is not very different from the Gaussian distribution (*top left*) but their negative log likelihood are very different (*bottom left*), in fact the Cauchy distribution accounts for larger deviations (outliers) than the Gaussian one (*right*).

estimate. The common cause for this type of mismodelling is the failure to take into account the existence of *outliers* (e.g. abnormal data values caused by unexpected features). An example of such incorrect modelling, is the use of the Gaussian distribution to model a measurement likelihood that does not account for outliers (Figure 3.2).

In mobile robot localization outliers are naturally present in range scans. To address this problem we can use a robustified version of the least squares problem presented in (3.15):

$$\hat{\mathbf{x}}_t = \underset{\mathbf{x}_t}{\operatorname{argmin}} \sum_i \rho_i \left(\varepsilon^2(\mathbf{x}_t \oplus \mathbf{p}_t^i) \right) \ , \tag{3.24}$$

where ρ is a *loss function* used to reduce the influence of outliers. The advantage of this formulation is that we can maintain the original measurement model, i.e. the likelihood field. Following Triggs et al. (2000) proposal, we use the *Cauchy* loss function ρ and its respective weight function w:

$$\rho(x^2) = \frac{c^2}{2} \ln \left(1 + \frac{x^2}{c^2} \right) \tag{3.25}$$

$$w(x) = \frac{1}{x} \frac{d\rho}{dx} = \frac{1}{1 + \frac{x^2}{c^2}} \ , \tag{3.26}$$

where c is the factor that controls the behavior of the loss function. The minimization process follows the same steps as previously discussed but with the following substitutions:

$$\begin{aligned} \overline{\mathbf{J}}_i = \sqrt{w_i}\, \breve{\mathbf{J}}_i \quad \wedge \quad \overline{\varepsilon}_i = \sqrt{w_i}\, \varepsilon(\hat{\mathbf{x}}_t \oplus \mathbf{p}_t^i) \\ (\overline{\mathbf{J}}^T \overline{\mathbf{J}}) \Delta \mathbf{x} = -\overline{\mathbf{J}}^T \overline{\varepsilon} \end{aligned} \tag{3.27}$$

This addition improves the estimation of the robot pose $\hat{\mathbf{x}}_t$ and makes the approach more tolerant to outliers (Triggs et al., 2000).

3.6 Global Localization

In localization taxonomy, the scan matching approach to localization falls only into the category of *local localization* (or *pose tracking*), and because its search space is restricted to a localized area it cannot solve the global localization problem. When a good initial pose is given to the localization process, this is not an issue. However, there are some situations where an initial pose is not available, an example being the scenario of an autonomous robot that has to find its pose in the environment (i.e. after powering up) without human intervention or external devices.

To solve the global localization problem, we propose a solution akin to particle filters. The probability $p(\mathbf{x}_t|z_{1:t}, u_{1:t})$ is represented by a set of N weighted samples distributed according to a uniform distribution:

$$\mathcal{U}(\mathbf{a}, \mathbf{b}) \approx \left\{\mathbf{x}^{[i]}\right\}_{i=1,\dots,N} , \tag{3.28}$$

where $\mathbf{x}^{[i]}$ is a hypothetical pose (i.e. a sample of $\mathbf{x}$) and $\mathbf{a}$ and $\mathbf{b}$ are the minimum and maximum coordinates that define the bounding box of the known space, respectively. A uniform distribution is used to represent the fact that the true pose can be anywhere in the environment with equal probability. To reduce the search space for random sampling we add the constraint that a sample must be in free space:

$$\mathbf{x}^{[i]} \sim \mathcal{U}(\mathbf{a}, \mathbf{b}) : m(\mathbf{x}^{[i]}) = \text{free} . \tag{3.29}$$

The weight of each sample is defined as the sum of the squared error of the current measurement:

$$w^{[i]} = \sum_k \varepsilon^2(\mathbf{x}^{[i]} \oplus \mathbf{p}_t^k) . \tag{3.30}$$

After sampling, the sample with the lowest weight is chosen to represent the true pose of the robot:

$$\mathbf{x} = \min_{w^{[i]}} \left\{\mathbf{x}^{[i]}, w^{[i]}\right\}_{i=1,\dots,N} . \tag{3.31}$$

Using this sampling method it is not guaranteed to find the true pose. The number of sample N used may not be enough to explore the full free space. Instead of increasing the number of samples N, the sampling process is repeated until the weight of the sample with the lowest weight is below a threshold α. The pseudo-code of the sampling global localization is presented in algorithm 12.

3.7 Experimental Results and Evaluation

We seek to evaluate the quality of the pose estimates generated by our localization solution as well as its computational efficiency. For that purpose, we used a set of publicly available datasets, and ran them through our algorithm. The datasets in question are the ACES Building, the Intel Research Lab, the MIT CSAIL Building, the Freiburg Indoor Building 079 and the MIT Killian Court [2]. For reference, all experiments were run in a computer with an Intel Core i7 2.8GHz and 16GiB of RAM, and all data was used without pre-processing.

To better evaluate our result we need to define a comparison baseline. For that purpose we chose the Adaptive Monte Carlo Localization (AMCL) (Fox, 2003), a localization algorithm

[2] `http://cres.usc.edu/radishrepository/view-all.php` (last access: September 2017)

Algorithm 12: Sampling global localization

global localization(z_t, m, α, N **):**
  $\mathcal{S} = \emptyset$
  repeat
    for $i = 1$ **to** N **do**
      $\mathbf{x}^{[i]} \sim \mathcal{U}(\mathbf{a}, \mathbf{b}) : m(\mathbf{x}^{[i]}) = \text{free}$
      $w_t^{[i]} = \sum_k \varepsilon^2(\mathbf{x}^{[i]} \oplus \mathbf{p}_t^k)$
      $\mathcal{S} = \mathcal{S} + \langle \mathbf{x}^{[i]}, w_t^{[i]} \rangle$
    end
    $\langle \mathbf{x}, w_{\min} \rangle = \min_{w^{[i]}} (\mathcal{S})$
  until $w_{\min} < \alpha$
  return x

based on particle filter with an adaptive number of samples. This decision is based on the fact that AMCL is a popular state-of-the-art localization algorithm widely used in mobile robotics. To distinguish our solution from AMCL, we will refer to it as Scan Matching or SM. Additionally, for SM, results for different optimization strategies, i.e. Gauss-Newton (GN) and Levenberg-Marquardt (LM), will also be presented and evaluated.

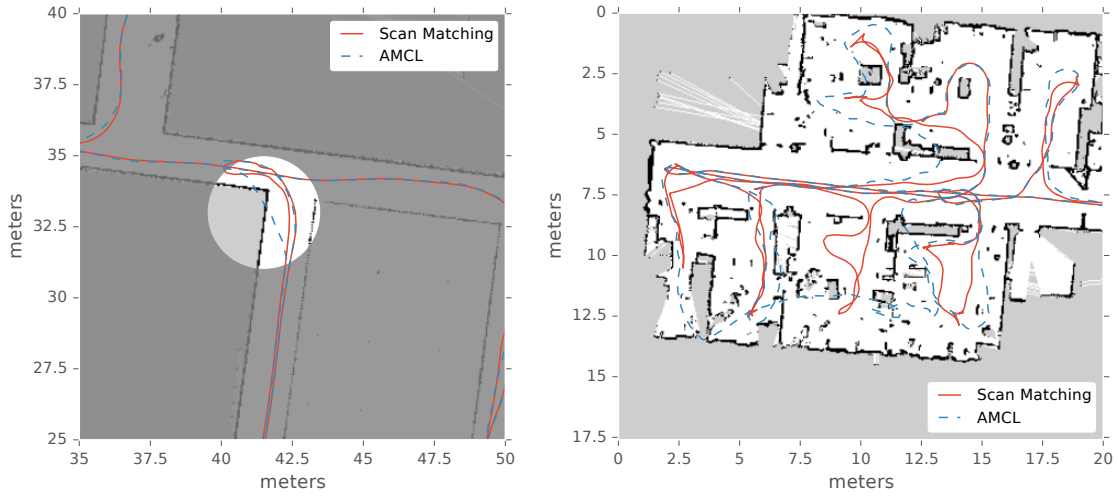
For both solutions (AMCL and SM), a localization update only occurs after accumulating $\rho = 0.2m$ translation motion or after accumulating $\vartheta = 0.5rad$ rotational motion. In AMCL's particle filter a minimum of 500 and a maximum of 5000 particles are used. Both solutions use a likelihood field measurement model, but with the difference that AMCL only utilizes 30 scan rays for evaluation while our proposal uses the complete set of scan rays.

3.7.1 Trajectory Validation

The set of all pose estimates results in the trajectory traveled by the mobile robot and any viable robot localization algorithm should obtain a trajectory closer as possible to the real one. The best method to evaluate the obtained trajectories is to compare them with a ground-truth. Unfortunately a ground-truth for the selected datasets is not available, however it is still possible to assess the validity of a trajectory. A trajectory is considered to be fully (or partially) valid when: for the same dataset the trajectories outputted by two different algorithms are similar; and the trajectory only crosses free space.

In all datasets the trajectories from our solution and AMCL are very similar, Figures 3.6(a), 3.7(a), 3.8(a), 3.9(a) and 3.10(a), but with a localized and a broader exceptions. The localized exception is found in the CSAIL dataset. Here, on a contained area of the map, the AMCL algorithm fails to maintain the trajectory inside free space while our solution maintains a valid trajectory, as depicted in Figure 3.3(a).

The broader exception if found in the Fr079 dataset. There are considerable differences between our proposal's trajectory and AMCL's at certain areas of the map, e.g. Figure 3.3(b). Nonetheless, the trajectory of our solution is always contained in free space while AMCL's is not. The odometry provided by the dataset contains considerable errors that frequently lead to erroneous pose estimations in most localization algorithms. By increasing the pose estimation frequency our solution was able handle the error in the odometry. The same increase was used in the AMCL algorithm, but it still failed to handle the errors in the odometry.



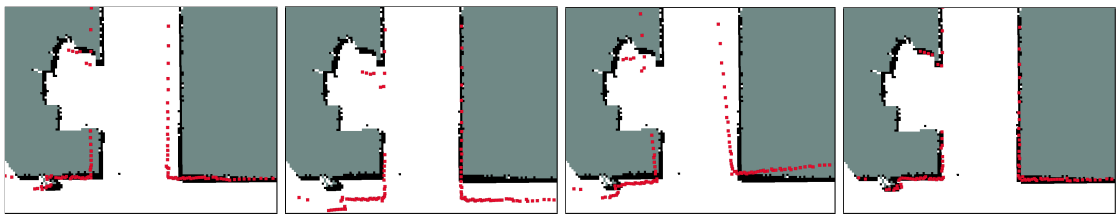
(a) CSAIL (b) Fr079

Figure 3.3: Erroneous deviation of AMCL's trajectories. (a) Localized trajectory deviation in the CSAIL dataset. (b) Considerable amount of trajectory deviations in the Fr079 dataset due to errors in the odometry.

3.7.2 Accuracy of Pose Estimates

As already stated, the best method to evaluate the accuracy of the pose estimates would be to compare them with a ground-truth, unfortunately one is not available. Instead of directly evaluate the pose estimate, we propose to evaluate how well the current scan matches the map at the estimated pose. We reason that the scan matching error correlates with the accuracy of the pose estimate, the lower the scan matching error the higher the accuracy of the estimate. An example is showed in Figure 3.4. The metric used to (indirectly) assess the accuracy of each pose estimate is the root mean square error (RMSE) of the scan matching defined as

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}\,, \tag{3.32}$$



(a) (b) (c) (d)

Figure 3.4: Pose estimate accuracy inferred from scan-to-map matching errors. (a) Pose estimate with errors in the Y axis. (b) Pose estimate with errors in the X axis. (c) Pose estimate with angular errors. (d) Pose estimate with the smallest scan matching errors.

where $\hat{y}_i$ is the estimate of y_i. In our scan-to-map matching, the error is given by the Euclidean distance of a measurement end-point and the closest object in the map, which is already given by (3.11). Hence our metric to evaluate the accuracy of pose estimate is given by

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \varepsilon^2(x_t \oplus \mathbf{p}_t^i)} \; . \tag{3.33}$$

An overview of the obtained errors for each algorithm, group by dataset, is presented in Figure 3.5, and showed with more detail in Figures 3.6(b), 3.7(b) 3.8(b) 3.9(b) 3.10(b). The 95% percentile of the error distribution indicates the value below which the error observations fall and allow us to compare the error between algorithms. Our scan matching solution has a lower error than AMCL for most datasets, and when comparing with Gauss-Newton strategy, our proposal has constantly lower error than AMCL.

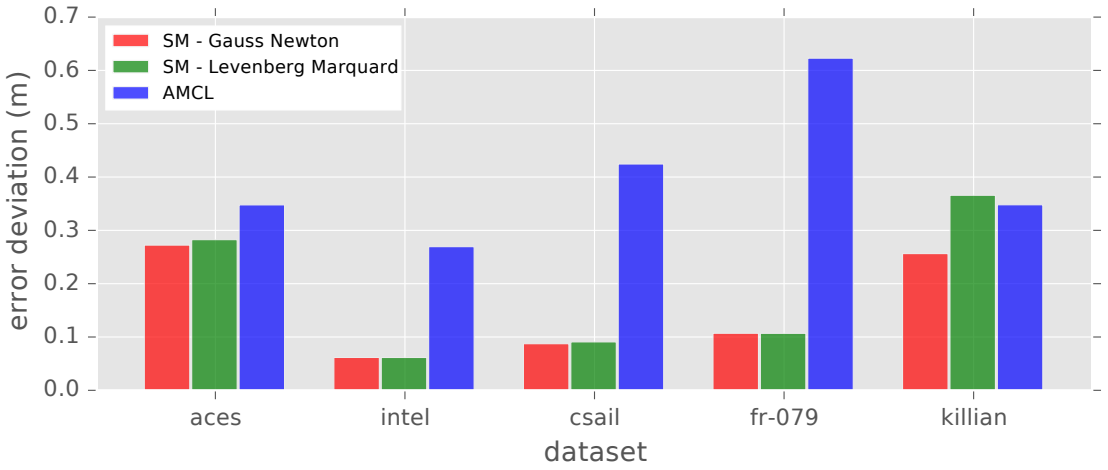


Figure 3.5: Root mean square error overview. For each algorithm, grouped by dataset, the 95% percentile value is showed (a lower value is better).

3.7.3 Computational Efficiency Analysis

We are also interested in analyzing the computational efficiency of our localization solution. More specifically, the mean processing time (or update time) between each scan s_{proc} and the mean number of iterations needed during optimization.

The execution times of our solution and AMCL's are summarized in Table 3.1, and showed with more detail in Figures 3.6(c), 3.7(c), 3.8(c), 3.9(c) and 3.10(c). The scan matching solution offers some interesting results in terms of computational efficiency. Not only it is, on average, faster than AMCL but the gamut of execution times is considerable shorter. The Levenberg-Marquardt optimization strategy has an execution time that is regularly slightly above Gauss-Newton strategy. When using the Levenberg-Marquardt optimization strategy with the Killian dataset the optimizer, in an isolated event, has poor convergence which results in abnormal maximum execution time when compared with other datasets.

All datasets have a data rate of approximately 6Hz (a 166 milliseconds period) and that means that our scan matching solution and AMCL are always capable of estimating the pose

Table 3.1: Mean execution times and respective standard deviation, and min-max values for all datasets, grouped by algorithm.

	Scan Matching - GN		Scan Matching - LM		AMCL	
	$s_{\text{proc}}\ ms$	min-max	$s_{\text{proc}}\ ms$	min-max	$s_{\text{proc}}\ ms$	min-max
aces	0.67 ± 0.28	$0.19 - 2.00$	0.96 ± 0.34	$0.33 - 3.02$	3.98 ± 4.78	$2.62 - 57.42$
intel	0.56 ± 0.18	$0.24 - 1.95$	0.78 ± 0.24	$0.33 - 2.64$	13.5 ± 11.9	$7.68 - 144.0$
csail	1.40 ± 0.61	$0.35 - 5.90$	1.88 ± 0.75	$0.65 - 7.50$	5.14 ± 7.76	$2.55 - 65.47$
fr079	0.99 ± 0.37	$0.28 - 4.53$	1.40 ± 0.48	$0.45 - 5.81$	5.56 ± 7.85	$2.60 - 85.75$
killian	0.66 ± 0.32	$0.18 - 5.75$	1.16 ± 0.80	$0.17 - 41.5$	2.91 ± 0.51	$2.56 - 34.11$

in real-time. However, nowadays LIDAR sensors have data rates of 25Hz (a 40 milliseconds period) or even 50Hz (a 20 milliseconds period) which may hinder AMCL's capability to process data in real-time, but our solution still maintains the real-time property.

In AMCL, the number of particles in the set has a direct influence in its computational efficiency, the higher the number of particles the higher the processing time. Because this number adapts to the variance in the particle set, areas in the map with higher uncertainty (or error) triggers an increase in the number of particles that results in higher processing times, such phenomenon can be observed in Figure 3.6(c) where different plateaus of executions times are visible. This observation tell us that, for most datasets, AMCL needs to continuously trade computational efficiency for better pose estimates.

In our scan matching solution, the number of iterations per optimization has a clear impact in its computational efficiency - the lower the number of iterations, the higher the computational efficiency. A summary of the number of iterations of our solution, per strategy, is shown in Table 3.2. On average, our solution has a reduced number of iterations during optimization, specially when using the Gauss-Newton strategy, and as result it has a high computational efficiency. The reduced number of iterations is also an indication that the emplaced non-linear least squares optimization has good convergence. As previously mentioned, the Levenberg-Marquardt strategy has a one-time event in the Killian dataset where its convergence is poor.

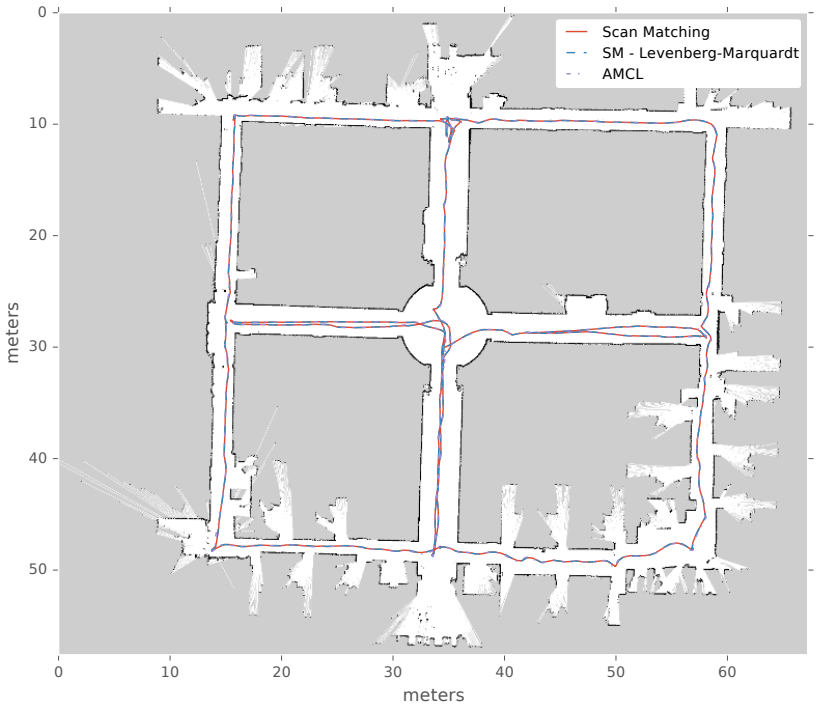
Table 3.2: Mean number of optimization iterations and respective standard deviation, and min-max values for all datasets, grouped by algorithm.

	Scan Matching - GN		Scan Matching - LM	
	# iterations	min-max	# iterations	min-max
aces	6.93 ± 3.34	$1 - 24$	12.30 ± 4.60	$3 - 37$
intel	5.70 ± 2.14	$1 - 22$	9.88 ± 3.63	$3 - 32$
csail	8.10 ± 4.03	$1 - 40$	13.37 ± 5.86	$3 - 58$
fr079	5.47 ± 2.24	$1 - 25$	9.65 ± 3.52	$2 - 34$
killian	5.99 ± 3.40	$1 - 44$	13.84 ± 9.26	$1 - 488$

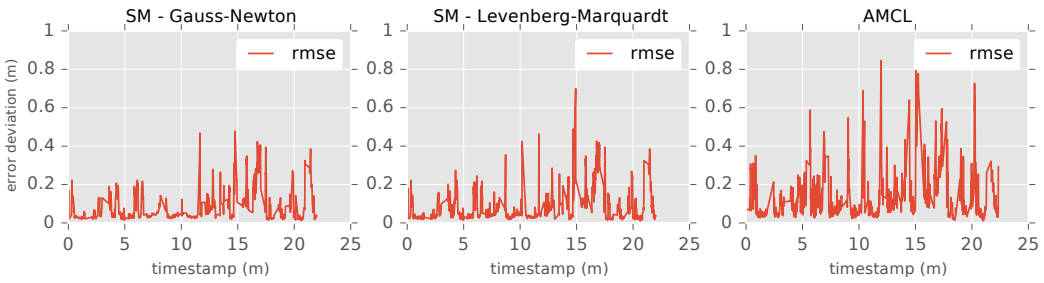
3.8 Conclusion

This chapters presents a localization algorithm for mobile robots based on scan matching supported by a likelihood field. The localization problem is first introduced as a maximum likelihood pose estimation, and by introducing a continuous likelihood field as the measurement

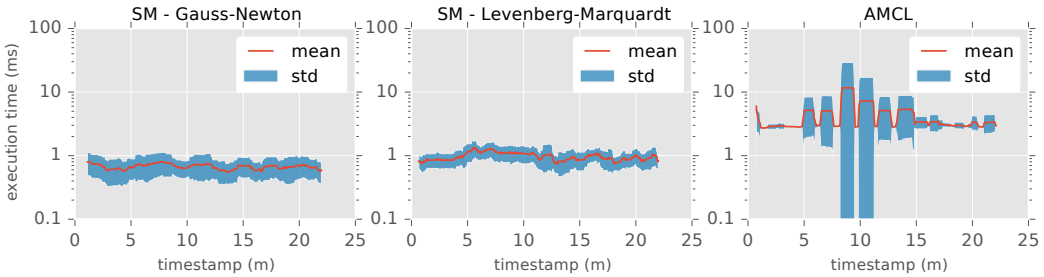
model the localization problem is converted into an scan matching problem solved by non-linear least squares on a manifold. In the carried experiments, our scan matching proposal has a high computational efficiency that results from good convergence of the optimization process. Furthermore, when compared with AMCL, a state-of-the-art localization algorithm, our solution not only provides pose estimates with better accuracy for all experiments but also trajectories that never cross occupied space, like in some experiments with AMCL. The end result is an accurate localization algorithm for mobile robots with a high computational efficiency that makes it suitable for tasks with real-time requirements.



(a) Robot trajectory



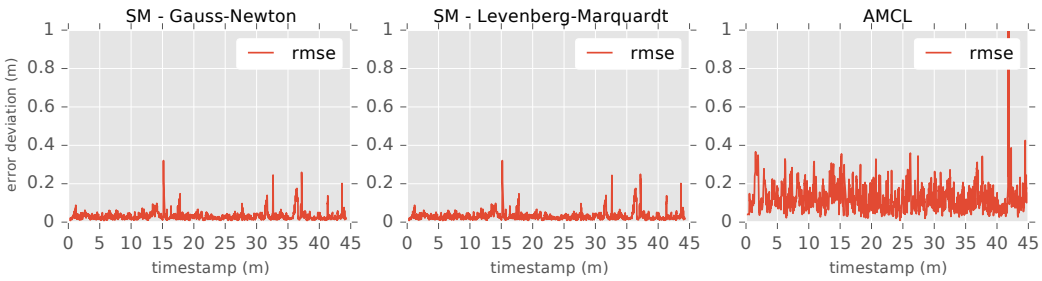
(b) Root mean square error at each pose estimate



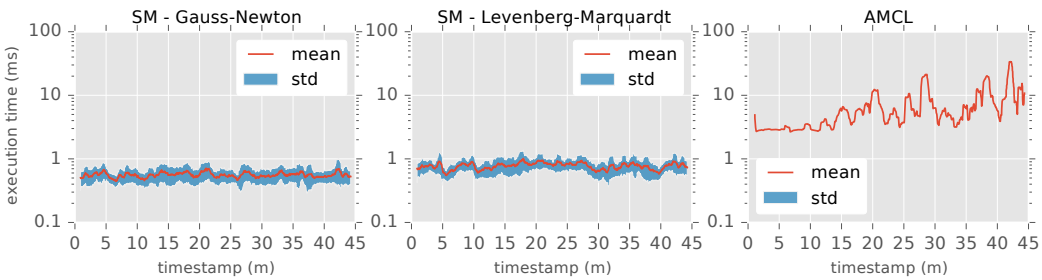
(c) Execution times

Figure 3.6: Localization experimental results for the ACES dataset.

(a) Robot trajectory

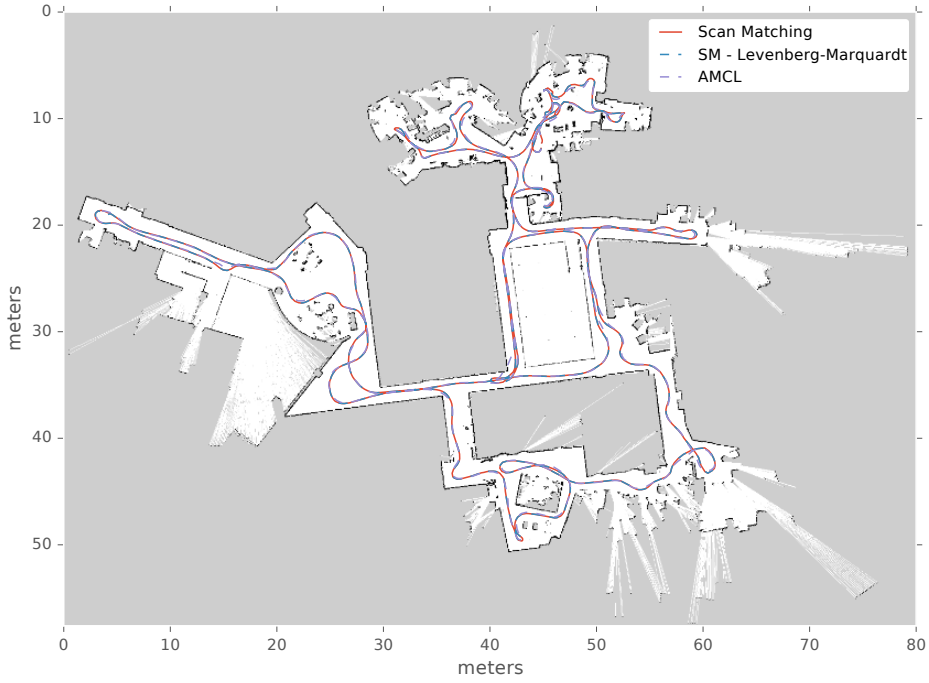


(b) Root mean square error at each pose estimate

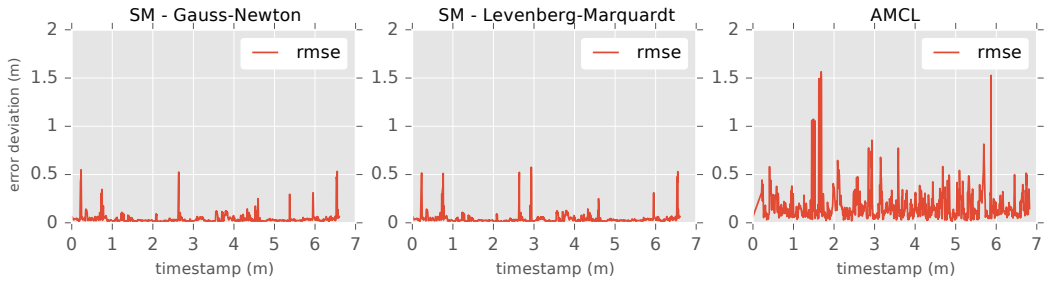


(c) Execution times

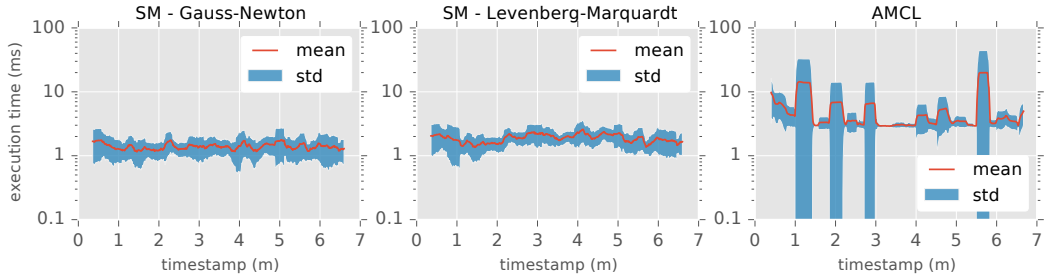
Figure 3.7: Localization experimental results for the Intel dataset.



(a) Robot trajectory



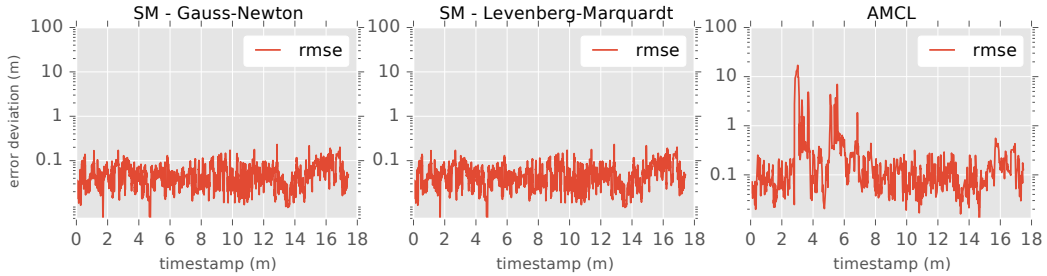
(b) Root mean square error at each pose estimate



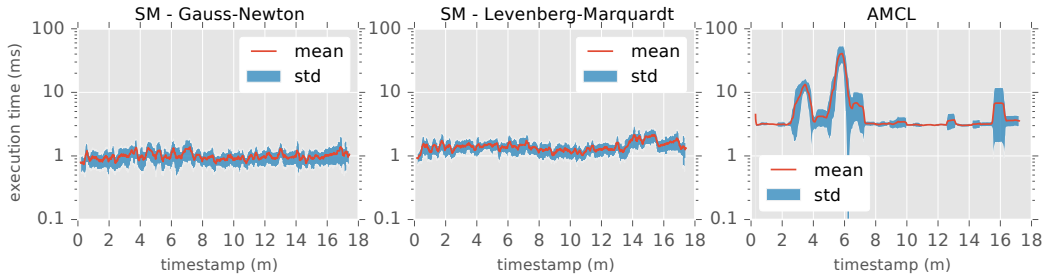
(c) Execution times

Figure 3.8: Localization experimental results for the ACES dataset.

(a) Robot trajectory

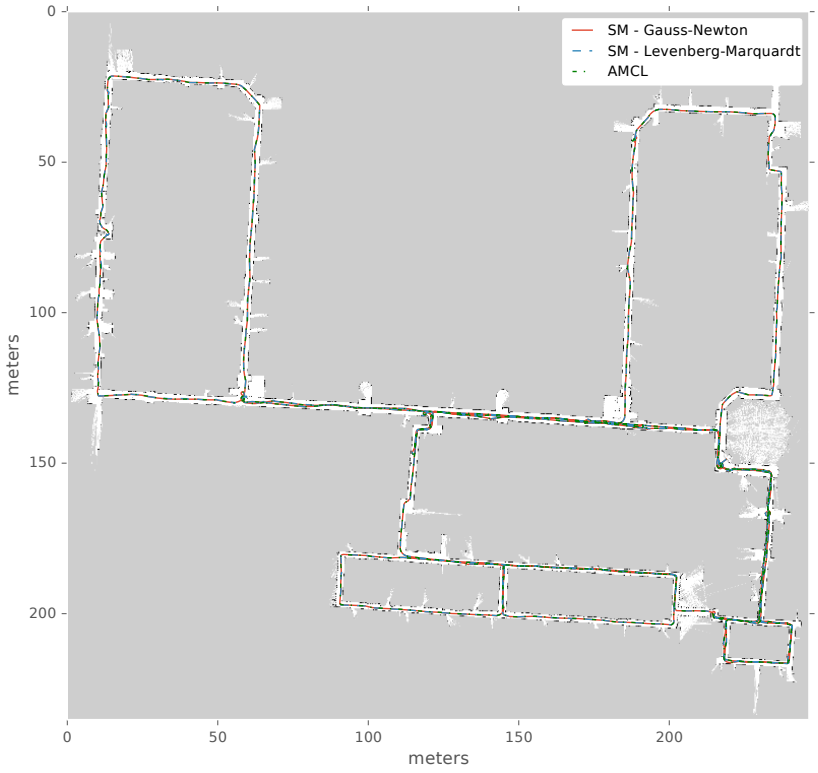


(b) Root mean square error at each pose estimate

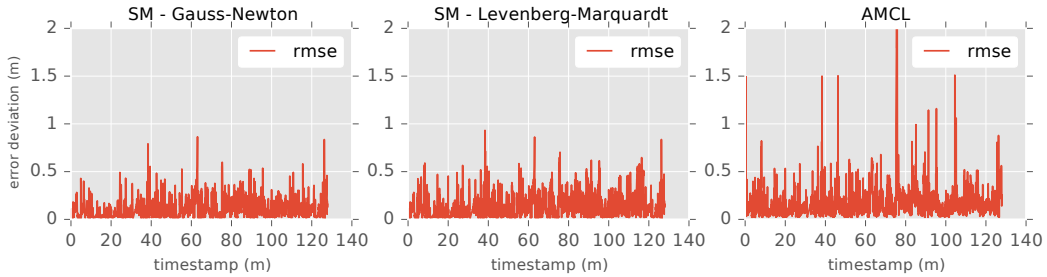


(c) Execution times

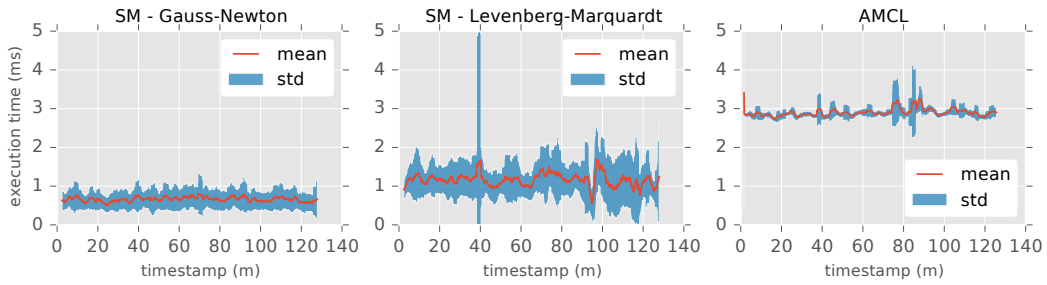
Figure 3.9: Localization experimental results for the Fr079 dataset.



(a) Robot trajectory



(b) Root mean square error at each pose estimate



(c) Execution times

Figure 3.10: Localization experimental results for the Killian dataset.

Chapter 4

A Sparse-Dense Approach for Efficient Grid Mapping

The creation of models of the environment is an important part for many robotic applications (Burgard and Hebert, 2008). One of such models is the spacial representation of the environment which is used in a variety of applications such as robotic localization, path planning, manipulation, and object recognition. This type of applications requires the capability to provide a probabilistic representation of space capable of modeling free, occupied and unknown areas with efficient runtime and memory usage.

One of the earliest and popular approaches to model a three-dimensional environment uses a regular volumetric grid that discretizes the space into cubic volumes of equal size, usually known as *voxels*, with early approaches authored by Roth-Tabak and Jain (1989); Moravec (1996). Implementing a regular volumetric grid is straightforward, hence its popularity. Let g be a linear array that contains all cells of a grid map and $\hat{c}_{ijk}$ a cell located at the coordinates $(i, j, k) \in {\mathbb{N}_0}^3$. For a grid map of size $(w \times h \times d) \in \mathbb{N}$, any grid cell can be referenced by

$$\hat{c}_{ijk} = g[k + d(j + ih)] \quad \text{with} \quad i < w, j < h, k < d \,. \tag{4.1}$$

There is also the matter of converting world coordinates to the discrete grid coordinates. If r is the resolution of the volumetric grid and (x, y, z) coordinates in the world frame, then the transformation to grid coordinates is given by

$$(i, j, k) = \left\lfloor (x, y, z)\frac{1}{r} \right\rceil + \vec{o} \,, \tag{4.2}$$

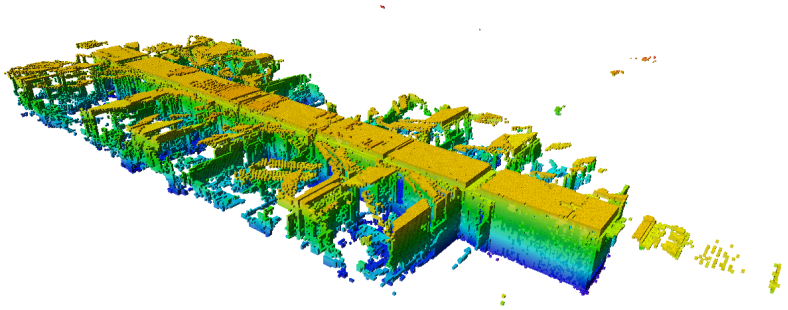
where $\vec{o}$ is the offset that defines the origin of the volumetric grid, i.e. it maps a world coordinate to the grid's origin coordinates $(i, j, k) = (0, 0, 0)$. Note that a two-dimensional version of the volumetric grid is a special case of the three-dimensional version where k and d are fixed to the values $k = 0$ and $d = 1$.

Despite its popularity, the regular volumetric grid representation has one major drawback in its implementation: it requires a large amount of memory. In its basic implementation, a grid must be instantiated with a size at least as big as the bounding box of the space that was mapped, independently of the grid cell distribution that was actually used. In large environments, such as outdoor, or even is scenes where a high granularity is required, the required memory can actually be a technical obstacle that prevents its usage. But there is another drawback that hinders its efficiency: it requires beforehand the extent of the area

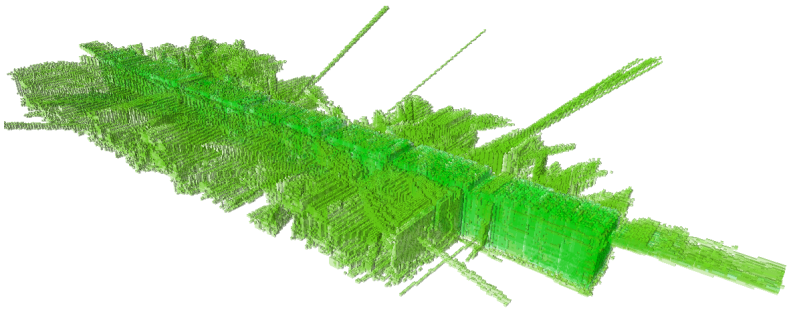
being mapped, or, if size readjustment is in place, it may require a time consuming copy operation every time the map area expands.

Other models have been proposed to addressed the aforementioned issues of the volumetric grid representation such as: elevation map (Herbert et al., 1989) that is essentially a 2D grid where each cell contains its measured height; multi-level surface map (Triebel et al., 2006) that is similar to the elevation map but it recognizes the existence of more than one surface per cell; and point sparse maps that instead of discretizing the environment it saves the measurements directly, like the point cloud approach proposed by Cole and Newman (2006) for 3D SLAM. But these models still have shortcomings: the first two fail to model unknown mapped areas and are unable to represent arbitrary environment; the last one does not handle memory efficiently, as more range data are acquired more memory is needed to store them.

Hornung et al. (2013) proposed a probabilistic 3D mapping framework, called OctoMap, that implements a volumetric grid using an octree with a focus on probabilistic occupancy maps. An octree is a hierarchical tree data structure that recursively subdivides a volume into eight sub-volumes until a minimum voxel size is achieved. Additionally, it does not require prior knowledge about the extent of the mapped area, grid cells are only initialized when a measurements needs to be integrated. This results in an efficient space management that considerably reduces the necessary memory to represent a volume grid, and by pruning the tree it is possible to further reduce memory consumption. OctoMap is at the moment the most popular and widely used volumetric grid implementation in the field of robotics. An example of its usage is shown in Figure 4.1.



(a) Representation of the occupied volumes with color coded height.



(b) Representation of the free volumes.

Figure 4.1: 3D Occupancy map of the FR-079 corridor dataset, constructed offline with OctoMap at a resolution of 0.05 meters.

4.1 Motivation

The work presented in Chapter 3 provides a fast and accurate localization algorithm and it is our intention to explore its applicability in two-dimensional SLAM with occupancy grids, specifically for *online* SLAM and *full* SLAM. The use of our localization proposal in *online* SLAM requires two grid maps: a probabilistic occupancy map that is built by integrating measurement and a likelihood field that is a reflection of the occupancy map. For the *full* SLAM problem we decided to pursue a solution based on a Rao-Blackwellized particle filter, that although it calculates a full path posterior, each particle estimates one pose at-a-time (Thrun et al., 2005) following an *online* approach. This realization opens the possibility to use our *online* solution as part of a *full* solution. Generating and using maps is ubiquitous in SLAM and with our approach pose estimation requires two maps, and scaling to a particle filter adds additional pressure on the issue of having an efficient volumetric grid implementation, as each particle represents a pose estimation that contains its own map.

To fulfill our requirements we need a volumetric grid implementation with efficient space (or memory) management without compromising runtime. The 3D mapping framework OctoMap is a volumetric grid implementation that offers such efficiency. It defines three essential requirements for a reliable implementation: probabilistic representation, modeling of unknown areas and efficient runtime and memory usage. Let's discuss them and their applicability to our application.

Probabilistic representation To create a 3D model of the environment a mobile robot senses the space by taking 3D range measurements. These measurements are bound to be tainted by uncertainty, as erroneous measurements can be caused by the limitations of the sensor but also by reflections and dynamic objects. To create an accurate model we need to address the fact that noisy measurements are part of the system, and therefore, we should model this underlying uncertainties using probabilities. A probabilistic representation provides a method to fuse multiple uncertain measurements into a robust estimate of the true state of the environment. Additionally, in probabilistic sensor fusion the integration of multiple measurements is not restricted to a single sensor or even to a single robot.

A probabilistic occupancy grid map is a text-book example of a viable probabilistic representation of the environment. It is widely used in applications such as SLAM, motion planning or even scene representation. However, the "probabilistic representation" is coupled with the semantics of probabilistic occupancy mapping, that forgoes the fact that not all grid maps need to represent probabilities. Such an example is the likelihood field that we used as a measurement model for localization using scan matching (described in section 3.2). It also needs the modeling of unknown areas, efficient access times and a proper memory management but it does not require a built-in probabilistic representation for the state of each grid cell. We argue that a volumetric grid implementation, at its lower level, should not be bounded by the semantics of the model it is supporting. This is not the case with OctoMap, since it is not possible to implement different models beyond occupancy grids to take advantage of the octree data structure use for memory management.

Modeling of unknown areas As an example, in autonomous mobile navigation, the knowledge about occupied and free space is essential for planing free-collision paths, and un-

known areas should be avoided, therefore it is also important to know the *unknown*. This also allows new possibilities such as autonomous exploration, where unknown areas are the points of interest. In the octree data structure, it is possible to model the *unknown* as every node that is not allocated is unknown space. However, once a node is allocated it cannot revert to an unknown state.

Efficiency There are two metrics to consider when referring to the efficiency of an implementation: **runtime** and **memory**. When mapping the environment, specially in 3D, the amount of data to integrate can be overwhelming and the system can ultimately lag behind and provide a representation that does not correspond to the current state of the environment. In the same situation, memory consumption is a major drawback. Unless a compact memory management is used, a large environment may not fit in the memory of the robot, rendering the system to a halt. These problems can be mitigated by sub-sampling or down-sampling (or both), but we are trading accuracy for better run-times and lower memory consumption.

Accessing data within the octree data structure requires an additional overhead when compared with the fixed-size volumetric grid. A random single update, on a tree with n nodes with depth d, has an access complexity of $\mathcal{O}(d) = \mathcal{O}(\log n)$, with additional memory allocation overheads when traversing unknown areas. In practice, the octree has a depth limited to the constant $d_{\max}$ that results in a complexity of $\mathcal{O}(d_{\max})$. In general, the octree data access is runtime efficient but, when dealing with 3D measurement, the high number of data accesses can undermine the runtime efficiency of the application, specially if more than one map is used at the same time. Reducing $d_{\max}$ would improve runtime efficiency, but in exchange the area that the octree is capable to represent would decrease significantly.

By delaying the initialization of a grid cell until it is necessary for integration, the octree is capable of reducing the necessary amount of memory to represent the known areas of the environment. Redundant information can be reduced by pruning the tree, which further reduces memory usage, but it requires all eight children of a node to have the same occupancy value, meaning that a single child can prevent pruning even though there is redundant information. This situation is further aggravated in two-dimensional applications, where pruning will never happen because at most only four children per leaf node will be used. For better compactness all grid cells can be brought to their maximum likelihood (either free or occupied), but this is a lossy technique that discards information for future updates. A better approach to increase compactness would be to evaluate information redundancy of grid cell clusters with a higher number of members without compromising runtime and memory efficiency. As a final observation, OctoMap does not provide a memory efficient method to duplicate a tree, as it will always do a deep copy, which is not optimal for application such as SLAM particle filter.

Although OctoMap is a successful implementation of a volumetric grid, proved by its extensive use in several robotic applications, we found that it is not suitable for our SLAM application. Not only it is bounded to a single model, the occupancy grid, but it is also inefficient for two-dimensional representations and does not provide an efficient duplication of data required by a SLAM particle filter. Hence, in this chapter we will introduce a volumetric grid implementation suitable for our SLAM application but that is general enough to be used in other applications, for example as a replacement for OctoMap.

4.2 Mapping Framework

The approach proposed in this chapter is an efficient hybrid solution that uses sparse division of space combined with a dense subdivision of space that is virtually capable of covering any mapped area at any defined resolution. To further reduce memory consumption a compression scheme is also in place. Additionally, it provides an abstraction layer that separates model implementation from space management, meaning that any model transparently inherits the underlying space management.

4.2.1 Sparse-Dense Volumetric Subdivision

An environment of unknown area size can, conceptually, be of infinite size. From this global perspective any perception of the environment is rather sparse. A sparse representation of the environment has the advantage of a lower memory requirement than its dense counterpart. The trade-off is a reduction in access time efficiency, but it gains the flexibility to represent both known and unknown space without additional cost. However, from a local perspective, when perceiving the environment with range data the knowledge about the environment is rather dense. Assuming locality, consecutive perceptions of the environment are related to its immediate surroundings. In this situation a dense representation of the environment is more advantageous, as access time is constant and it requires less initializations.

The sparse and dense representations are not mutually exclusive. Both can be combined to create an efficient representation of the environment. The proposal is to divide the volumetric grid into smaller dense volumetric grids of equal size that are referenced by a sparse structure. In summary, space is represented by a sparse grid of dense volumetric grids (Figure 4.2).

A dense volumetric grid is ruled by the parameter $L \in \mathbb{N}$ that defines the length of its edge measured in grid cells. Therefore, a dense volumetric volume of length L represents a sub-volumetric grid with L^3 grid cells, named *patch*. The *patch* is the actual container of grid cells that in turn contains the data used by the model, therefore, access to a grid cell goes through one level of indirection. Given the coordinates (i, j, k) of grid cell $\hat{c}_{ijk}$, the first step is to find the *patch* that contains such cell and then to obtain its reference within the *patch*.

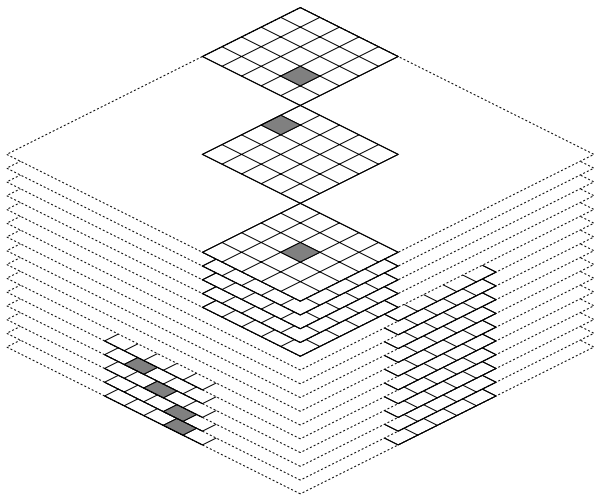


Figure 4.2: Sliced view of the volumetric grid with volume subdivision. The area is divided into sub-volumes of equal sides when data in integrated.

Let p be a *patch* and P a sparse structure (e.g. a dictionary) that holds all *patches*. Each *patch* is identified by a unique identifier given by the function $h : \mathbb{N}_0^3 \to \mathbb{N}_0$ such that

$$h(\hat{i}, \hat{j}, \hat{k}) = \hat{k} + \Lambda(\hat{j} + \hat{i}\Lambda) \quad \text{with} \quad \Lambda \in \mathbb{N} : \Lambda^3 \leq 2^b \,, \tag{4.3}$$

where Λ is the size limit of the volumetric grid imposed by the finite nature of all computational systems and b is the bits resolution that governs the imposed size limit. The access to the correct *patch* for the coordinates (i, j, k) is now given by

$$p_h = P\left[h(\frac{i}{L}, \frac{j}{L}, \frac{k}{L})\right] \quad \text{with} \quad i, j, k < \Lambda L \,. \tag{4.4}$$

If the *patch* is not found in P it should be created and initialized. Before obtaining the grid cell, it is now necessary to adjust the coordinates to the current *patch* p_h, which is achieved with the following:

$$(\bar{i}, \bar{j}, \bar{k}) = (i \bmod L, j \bmod L, k \bmod L) \,. \tag{4.5}$$

Finally, the grid cell $\hat{c}_{ijk}$ is obtained by

$$\hat{c}_{ijk} = p_h[\bar{k} + L(\bar{j} + \bar{i}L)] \,. \tag{4.6}$$

Note that p_h is a multi-dimensional array with a row-major flat memory layout where data is stored in a continuous memory segment that provides faster access times. The conversion of world coordinates to discrete grid coordinates is the same as (4.2), but with its origin $\vec{o}$ set to the geometric center of the volumetric grid, that is

$$\vec{o} = \frac{\Lambda}{2}(1, 1, 1) \,. \tag{4.7}$$

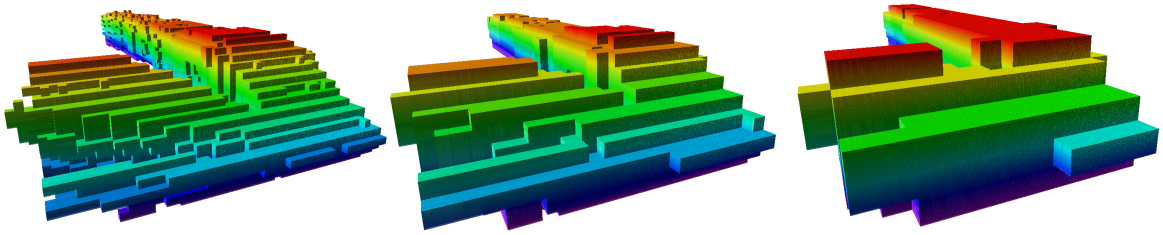
Despite the indirect reference to a grid cell introduced by the use of *patches*, from a user perspective any grid cell is accessible by providing its world coordinates (x, y, z), similar to the classic dense volumetric grid, making the use of *patches* a transparent process. Anecdotally, our volumetric grid with resolution r can map an environment of size

$$(rL\Lambda)^3 \text{ m}^3 \,, \tag{4.8}$$

which can be large as a room, a building, a city block, a city, a region or even a country, depending on the parameters that are used.

4.2.2 Space Efficiency

Space (or memory) efficiency of our proposed volumetric grid is heavily influenced by the value of L, i.e. by the number of grid cells contained in each *patch*. A single used grid cell in a *patch* requires the allocation and initialization of L^3 grid cells therefore, it is desirable to have a low value for L to avoid unnecessary allocations. However, the sparse structure that holds all *patches* trades access time for lower space requirements, hence by (4.4) a low value of L can introduce significant access time overheads and hinder time efficiency. The size of the *patch*, defined by L, is therefore a trade-off slide between space and time efficiency: a lower value provide better space efficiency but worse access times, and a higher value results in lower access times but with a considerable space overhead that results from cells that are allocated but not touched during integration. As more data is integrated into the volumetric grid the space overhead is attenuated, which is a visible effect of the local density assumption (see Figure 4.3). But, in some applications, the space overhead is too big to be neglected. To address this issue we propose three additions to our volumetric grid: two-dimensional awareness, data compression, and data sharing with reduced overhead.



(a) $L = 4$, $\approx 60\%$ space overhead (b) $L = 8$, $\approx 72\%$ space overhead (c) $L = 16$, $\approx 81\%$ space overhead

Figure 4.3: Visualization of sparse-dense volumetric subdivision for different values of L with respective space overhead. In this example only one scan is integrated. The space overhead corresponds to the percentage of cells that are allocated but not touched during integration. As more data is integrated the space overhead reduces, the final space overhead is approximately 32%, 48% and 64% for (a), (b) and (c) respectively.

Dimensional Awareness

Not all applications require a three-dimensional representation of the environment. A practical example is the two-dimensional mapping of an environment. In this situation, the allocation of L^3 grid cells for each *patch* has a guaranteed space overhead of

$$1 - \frac{1}{L} \geq 50\% \quad \forall L \in \mathbb{N} : L > 1 \,, \tag{4.9}$$

that will never be used. Such degree of space overhead is clearly not desirable. To avoid this shortcoming, our volumetric grid is aware of the environment dimensionality, that is, the dimensionality of the environment is explicitly defined as either two or three dimensional. To accommodate this awareness several changes to the volumetric grid are required. The first change is the number of grid cells allocated upon initialization of a *patch*. When the environment is two-dimensional, only L^2 grid cells are allocated instead of L^3. Furthermore, the unique identification of a *patch* (4.3) has the following modification:

$$h(i,j,k) = \begin{cases} k + \Lambda(j + i\Lambda) & \text{if } D = 3 \\ j + i\Lambda & \text{if } D = 2 \end{cases} \tag{4.10}$$

where D is the number of dimensions to consider. The access to an individual grid cell $\hat{c}_{ijk}$ also needs to be changed to in order to handle two-dimensional and three-dimensional linearized grids:

$$\hat{c}_{ijk} = \begin{cases} p_h[\bar{k} + L(\bar{j} + \bar{i}L)] & \text{if } D = 3 \\ p_h[\bar{j} + \bar{i}L] & \text{if } D = 2 \end{cases} \tag{4.11}$$

The remaining equations (i.e. (4.4), (4.6) and (4.7)) do not need to change, because for any value of k they remain valid.

Online Data Compression

As previously discussed, the use of *patches* comes from the local density assumption of measurements, however it does not guarantee that all grid cells are used to integrate measurements – a disadvantage when compared to an octree structure that only allocates a grid cell

when it is needed. But this trade-off in space provides some time efficiency, and depending on the application the space overhead may be negligible. Nonetheless, for situations where the overhead is not negligible we propose to take advantage of the *patch* flat memory layout to compress the data and reduce its size during operations.

In situations where most grid cells in a *patch* are "untouched", lossless data compression algorithms can be efficiently used to exploit the statistical redundancy contained in the data. This also has the advantage that data compression can be carried independently of the model implemented on top of the volumetric grid. Although data compression is a good mechanism to reduce the memory footprint of a *patch*, read and write operations require the data to be uncompressed. This raises the issue of when to compress and when to decompress data. A naive approach would be to decompress the data before every read/write operation and soon afterwards, compress it again. But such approach is not feasible, as data compression and decompression are computationally expensive operations and any space gain is overshadowed by the overall time inefficiency.

Our proposal for data compression relies on the use of a caching mechanism to maintain the recently accessed *patches*, with uncompressed data, so they can be accessed at a later time without incurring in a compress-decompress time penalty. The *patches* that are not referenced by the cache list have their data compressed. Cache replacement follows a least recently used (LRU) strategy. When cache replacement occurs, the data of the *patch* that enters the cache is decompressed, while the data of the *patch* that is removed from the cache is compressed (see Figure 4.4).

The viability of this cache mechanism relies, once more, in the local density assumption and in the size of the cache list. For example, concerning local density, in the integration of a three-dimensional laser scan, consecutive range rays are close in space and likely to "touch" the same *patch*. The optimal size of the cache list is an open issue.

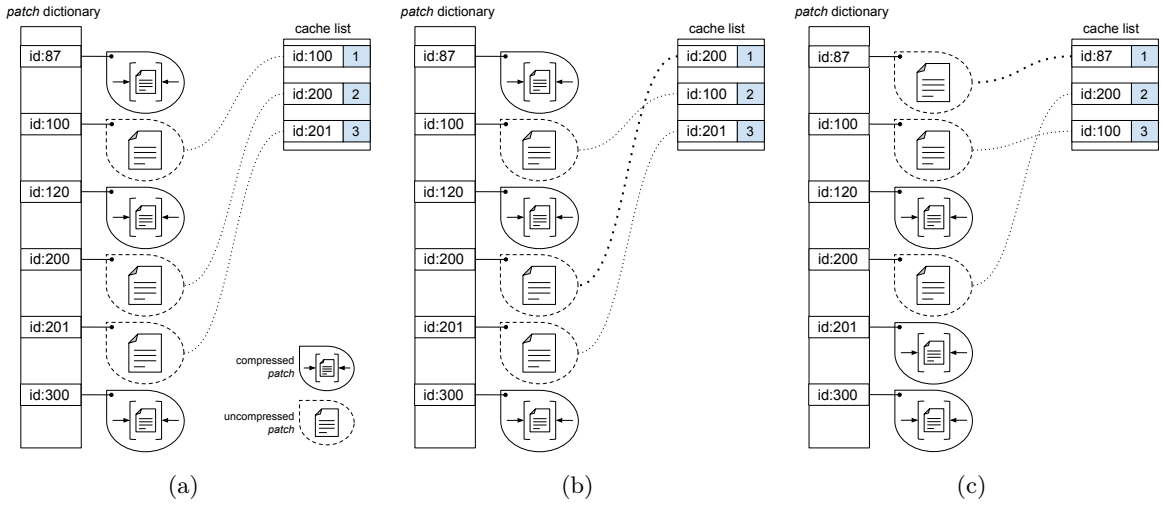


Figure 4.4: Cache mechanism for online data compression with an LRU cache replacement strategy. Scenario (a) depicts the current state of the cache, only the *patches* referenced by the cache list have their data uncompressed. In (b), *patch*-200 in the cache list is referenced, as result it is moved to the top of the cache list, no other operations being required. Next, in (c), *patch*-87 **not** in the cache list is referenced, as result, and because the cache list is full, the *patch*-201 referenced at the bottom of the cache list is removed from the list and its data is compressed, then the reference *patch*-87 is added to the cache list and its data in uncompressed.

Implicit Data Sharing

The modeling of the environment is not restricted to a single volumetric grid. The environment can be represented by different grids that capture distinct characteristics, such as occupancy, traversal cost or distance to closest object. There are also modeling techniques that reconstruct the environment by branching from the same model state with different parameters and later prune the model tree, by removing the model instances that worse represent the actual environment. The grid-based SLAM with Rao-Blackwellized particles (Grisetti et al., 2007a) is a perfect example of model branching. Each particle contains its own representation of the environment that is inherited from its ancestral particle during re-sampling. Such inheritance implies a full copy of the grid for each particle that is spawn. In a reasonable sized environment, combined with the number of particles, the space requirements can grow to an unmanageable amount. To address this issue, Eliazar and Parr (2003) propose the use of a single grid (instead of multiple) where each grid cell contains an ancestry tree that associates changes to particles. However, this is a specialized solution that only works for particle filters and breaks our architecture of a volumetric grid that is agnostic to any model.

Fairfield et al. (2007) introduced an efficient method to share an octree structure by wrapping each node of the octree with a copy-on-write (CoW) structure that employs a strategy where data duplication is deferred until the data is changed. Duplicated CoW structures keep a count reference to the same resource, but when a caller tries to modified the data a private copy is created and the previous reference is release, unless it has a single reference, which indicates that data duplication is not needed. The advantage of this strategy is that if the resource is never modified it will will never be duplicated.

Considering our volumetric grid solution, efficient data sharing must operate at the *patch* level. Thus, by importing the CoW strategy to our volumetric grid, each *patch* is wrapped by a CoW structure to optimize data sharing. Now, the duplication of a volumetric grid only implies a copy of the sparse structure that contain CoW references to the *patches*, and the duplication of a *patch* only happens when it is accessed for a write operation (see Figure 4.5).

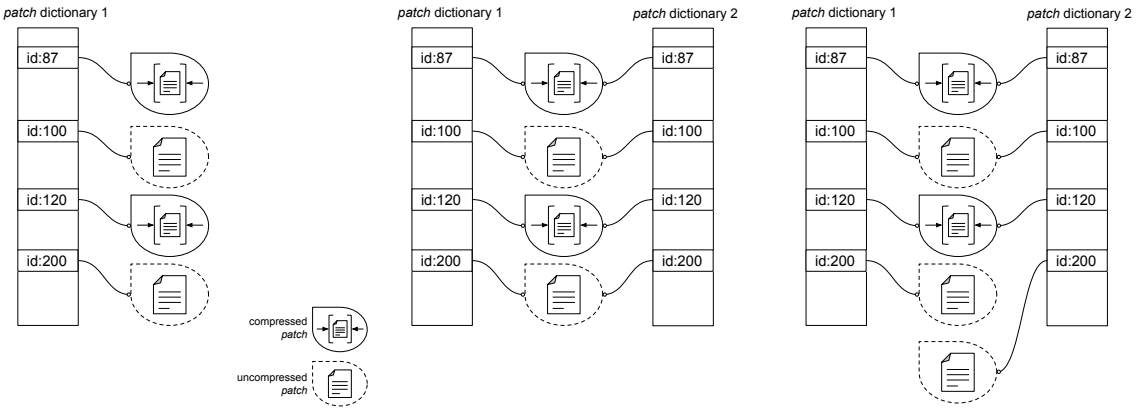


Figure 4.5: Copy-on-write (CoW) strategy for implicit data sharing. Scenario (a) depicts the current state of a volumetric grid. In (b) the initial volumetric grid is duplicated, and due to CoW, the *patches* belong to both volumetric grids without the need to duplicate. Next, in (c), *patch*-200 in one of the grids is accessed for writing, following the CoW strategy the *patch* is duplicated and the reference to the previous is released.

4.3 Implementation Details

This section contains the description of the most relevant details of our implementation of the proposed sparse-dense volumetric grid. The software was developed as a C++ library and tested in the Linux and MacOS platforms.

4.3.1 Software Architecture Overview

One of the most important requirements of our mapping framework is the separation of the volumetric grid space management from model implementation and that is achieved with the software architecture depicted in Figure 4.6. As result, space is provided to the model as a discrete dense division of the environment (i.e. grid cells) without forcing or requesting size limits. The advantage of this abstraction is that data compression and implicit data sharing is transparently available for any model, independently of its logic. The exception is the dimensionality awareness, where the model has to declare its preference. Nonetheless, if a model does not declare its preference the volumetric grid will operate at three-dimensions, which is more than capable of handling a model with a two-dimensional logic – thus increasing the abstraction.

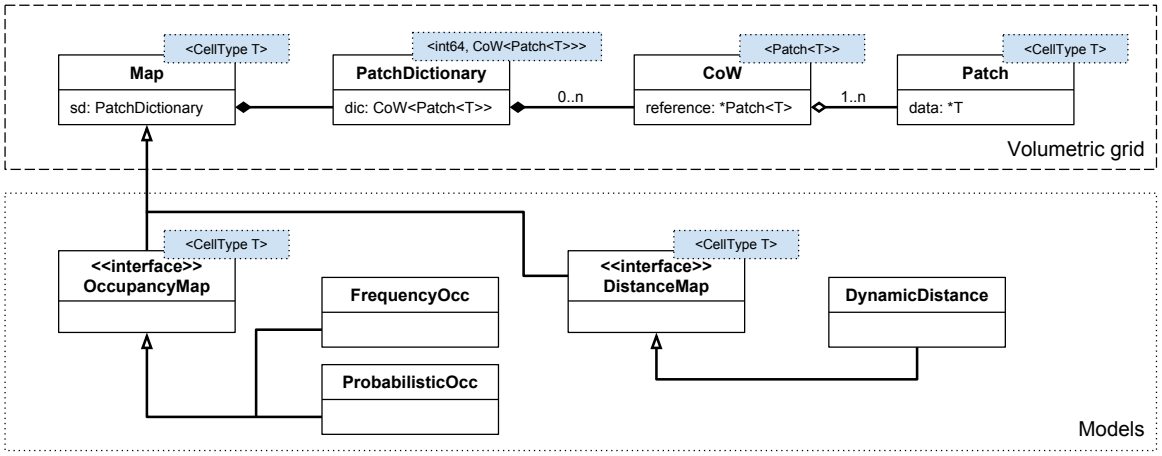


Figure 4.6: Simplified UML diagram of our software architecture.

In our architecture, the class *Map* is responsible for handling everything related with the volumetric grid. It includes methods to access individual grid cells, transform coordinates to and from map coordinates, control the compression mechanism and obtain internal state information such as memory usage and metric bounds of the map. To take advantage of the sparse-dense volumetric subdivision, every model has to adhere to the following convention: read-only operations must call the **immutable** version of the grid cell getter that returns a null reference when the requested coordinates does not map to an existing *patch*; and write operations must call the **mutable** version of the grid cell getter that dynamically allocates new *patches* when the requested coordinates does not map into an already allocated *patch*. Note that the grid cell getter is the implementation of (4.6). This convention also makes the CoW strategy possible. The differentiation between read and write operation is an obligatory functionality for an effective use of the strategy.

The implementation of a model starts by deriving from the class *Map*. An extra level of

abstraction can be added by defining an interface for different types of models. For example, occupancy maps represens the occupancy state of a grid cell, and require methods to update and query that state. However, there are different ways of updating the occupancy state, such as by using probabilities or by ternary states (unknown, free or occupied). They require different structures to represent the environment, but conceptually they are both occupancy maps.

4.3.2 Sparse-Dense Structure

All *patch* references are kept in a *map* container (i.e. a dictionary) that is implemented with a self-balancing binary search tree (e.g. red-black tree). The binary tree provides an average (and worst case) time complexity of $O(\ln n)$ for searching, insertion and deletion. Furthermore, accessing an individual grid cell in a *patch* is an $O(1)$ operation due to the use of the flat layout memory, or array linearization.

The maximum metric size of the volumetric grid is imposed by Λ that is defined by a bit resolution b (see (4.3)). For our implementation we chose a bit resolution of $b = 64$ that results in

$$\Lambda \in \mathbb{N} : \Lambda^3 \leq 2^{64} \implies \Lambda \leq \sqrt[3]{2^{64}}\,. \tag{4.12}$$

To maximize the area covered by the map, $\Lambda = \sqrt[3]{2^{64}}$ is used. In practice, we are defining the hash of a *patch* to be an unsigned 64 bits integer, where Λ is the maximum number where an integer overflow will not happen in the hash calculation.

4.3.3 Lossless Data Compression

As already discussed, we take advantage of the *patch*'s flat memory layout to compress and decompress the data when needed. Concerning the implementation, a crucial decision is which algorithm to use. Three aspects must be considered when selecting a lossless compression algorithm: compression ratio, compression time and decompression time. The amount of lossless compression algorithms available is considerable and each one of them tackles different aspects of compression. Some provide good compression and decompression speeds but sacrifice compression ratio, and others offer the opposite, good compression ratio at the cost of lower compression and decompression speed. It is also common, within each algorithm, to sacrifice compression speed for better compression ratios while maintaining similar decompression speeds. Shared among all algorithms is the asymmetry between compression and decompression speeds, where decompression is always faster.

The main concern for our application is to have a balanced trade-off between compression speeds and ratio. Using the caching mechanism, the number of compressions will always be equal to number of decompressions. At first glance, the compression speed of the algorithm can be a bottleneck, as already mentioned it is usual to trade compression speed for better compression ratio. The common algorithms for lossless data compression are LZ77/LZSS/LZMA (Ziv and Lempel, 1977; Salomon and Motta, 2010) and its derivatives, and popular choices are lz4[1], snappy[2], zlib[3] and zstd[4]. The algorithms lz4 and snappy have high compression and

[1] `http://lz4.github.io/lz4/`
[2] `http://google.github.io/snappy/`
[3] `http://www.zlib.net/`
[4] `http://facebook.github.io/zstd/`

Table 4.1: In-memory benchmark of lossless compression algorithms. The results were obtained with data blocks of 16KB using 1 core i7-2640M CPU @ 2.80GHz. The baseline is the system memory copy function.

name	**compression**	**decompression**	**ratio**
memcpy	5766 MB/s	5766 MB/s	1.00
lz4	469 MB/s	1723 MB/s	1.72
snappy	367 MB/s	1118 MB/s	1.77
zlib	65 MB/s	231 MB/s	2.49
zstd	164 MB/s	434 MB/s	2.55

decompression speeds with reasonable compression ratio, while zlib and zstd have higher compression ratio with reasonable speeds, albeit considerably lower. The performance of these algorithm is supported by an in-memory benchmark of lossless LZ77/LZSS/LZMA derivative compression algorithms, available online at `http://github.com/inikep/lzbench`. Their benchmark results are shown in Table 4.1.

Despite the benchmark, the actual choice and respective performance of an algorithm depends on the model, i.e. the integration rate, data redundancy, data density, etc. Therefore, several algorithms were implemented, including the already mentioned. This provides a higher flexibility to the user that can choose between speed and compression ratio.

During cache substitution there is a decompression followed by a compression, executed in sequential order. Decompression imposes a hard dependency: because data needs to be decompressed to be read and/or written the execution flow has to wait until the operation is finished to continue. On the other hand, compression does not impose any dependency to the main execution flow and can be performed in parallel – we name it *asynchronous compression*. This way, the increase in execution time by compression can be reduced, which attenuates the asymmetry between compression and decompression speeds. The parallelism and concurrency is achieved through multi-threading.

4.3.4 Multi-Threading Support

Nowadays, multi-core systems with several execution threads are ubiquitous. Such resources are an opportunity to increase the time efficiency of our volumetric grid implementation. To support multi-threading we need to identify the shared resources and respective critical sections. There are two levels of multi-threading support: thread-safe CoW mechanism and a thread-safe online data compression. The former assumes that each execution thread contains a branched version of the same volumetric grid, and the latter allows several execution threads to perform data compression and decompression to the same *patch* independently of belonging to multiple volumetric grids.

Considering the CoW mechanism, the only critical section is the duplication of a *patch*. The action is only performed if other CoW structures point to the same *patch* and that verification and possible duplication needs to be mutually excluded. Failing to do so can result in data corruption and unnecessary data duplications. Read-only operations does not impose a critical section because CoW guarantees that write operation will never change the data from other execution thread. Enclosing the verification and duplication in a critical section will inevitably introduce an execution time overhead, and considering that this procedure is called very frequently the system looses efficiency. Fortunately this can be avoided by performing an initial verification about the number of CoW structures referencing a *patch* before entering the

critical section. If the reference is unique, i.e. a single CoW structure references the *patch*, then the procedure terminates without entering the critical section. Otherwise, it enters the critical section and repeats the verification, to ensure that the duplication only occurs if the reference is **not** unique, as it could have been changed by other execution thread. A pseudo-code of the procedure, called *detach*, is presented in algorithm 13.

Algorithm 13: Thread-safe CoW data duplication with double-check lock.

```
CoW::detach( ):
    if p̂_ref is unique then return                    // First verification
    [enter critical section]
    if p̂_ref is not unique then                       // Second Verification
        p ← duplicate p̂_ref
        release p̂_ref
        p̂_ref ← reference of p
    end
    [leave critical section]
    return
```

Compression and decompression are two mutually excluded regions within a *patch*. However, due to the online data compression mechanism with implicit data sharing, the same *patch* can be in compressed and decompressed states at the same time. This paradox happens because each volumetric grid has its own cache list, thus the same *patch* can be present in one of the cache lists, ergo in a decompressed state, while not being present in the remaining cache lists, which implies a compressed state. The paradox is dismissed by introducing a simple rule: the presence of a *patch* in any cache list guarantees the decompressed state. This is solved by implementing a counting mechanism where a decompression increases the counting value and a compression decreases the counting value. In practice, the number of compression calls must be equal to the decompression number calls before an actual compression action takes place. The pseudo-code for compression and decompression are shown in algorithm 14 and algorithm 15 respectively. Consequently, the counting mechanism also works, at no cost, for the previously discussed *asynchronous compression*.

Algorithm 14: Thread-safe compression.

```
Patch::compress( ):
    [enter critical section]
    if count > 1 then
        count ← count − 1
        if count == 0 then
            ĉ ← lossless-compress(ĉ)
        end
    end
    [leave critical section]
    return
```

Algorithm 15: Thread-safe decompression.

```
Patch::decompress( ):
    [enter critical section]
    count ← count + 1
    if count == 1 then
        ĉ ← lossless-decompress(ĉ)
    end
    [leave critical section]
    return
```

4.3.5 Implemented Models

In the context of this thesis, two models were implemented on top of our sparse-dense volumetric grid: an Euclidean Distance map and an Occupancy Grid map. The reader may notice that these models are used in Chapter 3 as an integral part of our scan matching approach to localization. In fact, these implementations were the ones utilized for that work.

Distance Map

The Euclidean Distance map is an implementation of the dynamic Euclidean Distance map proposed by Lau et al. (2013), where only cells affected by the change of state, i.e. from occupied to free and vice versa, are updated, instead of the full map like in a traditional implementation. For a better understanding of the solution, its pseudo-code is presented in Algorithm 16, and for a detailed description please consult the aforementioned reference.

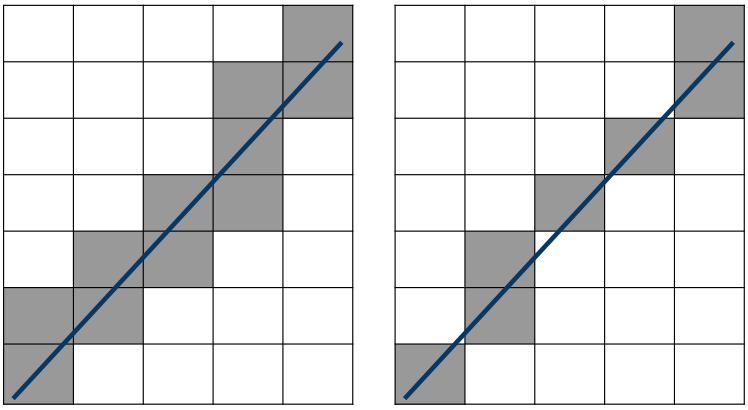
Algorithm 16: Pseudo-code for updating a dynamic Euclidean distance map (Lau et al., 2013).

```
setObstacle(s)
    obst(s) ← s
    D(s) ← 0
    insert(open, s, 0)

removeObstacle(s)
    clearCell(s)
    toRaise(s) ← true
    insert(open, s, 0)

updateDistanceMap()
    while open ≠ ∅ do
        s ← pop(open)
        if toRaise(s) then
            raise(s)
        else if isObst(s) then
            lower(s)
    end
    return D

raise(s)
    forall n ∈ Adj(s) do
        if obst(n) ≠
          cleared ∧ ¬toRaise(n)
          then
            insert(open, n, D(n))
            if ¬isOcc(n) then
                clearCell(n)
                toRaise(n) ←
                  true
    end
    toRaise(s) ← false

lower(s)
    forall n ∈ Adj(s) do
        if ¬toRaise(n) then
            d ← ||obst(n) − s||
            if d < D(n) then
                D(n) ← d
                obst(n) = obst(s)
                insert(open, n, d)
    end
```

Probabilistic Occupancy Map

Probabilistic occupancy grid maps are used to generate consistent metric maps from noisy and uncertain measurement data. Each grid cell is treated as an independent binary random variable, which specifies the probability that a grid cell is occupied. Independently of how the probability is updated, the integration of a measurement z_t requires the casting of all rays z_t^i to mark free space. The end-points of the rays z_t^i are used to mark occupied space. The casting of a ray can be achieved by using well known line drawing algorithms, which are the Digital Differential Analyzer (DDA) (Amanatides et al., 1987) and Bresenham (Bresenham, 1965). In the former algorithm all grid cells traversed by the ray are marked as a free. However, this process is time consuming for a large amount of rays, thus the introduction of the faster Bresenham algorithm that only uses integer operations. The Bresenham algorithm, although faster, has the downside of neglecting some grid cell that are traversed by the ray. A visual difference of the algorithms is depicted in Figure 4.7.

The ray casting operation can introduce unwanted effects due to its discretization. During a scan sweep of a flat surface, a volume can be measured as occupied in one ray and then as free in the following ray casts (Hornung et al., 2013). To minimize the effect, cells that are marked as occupied, during the same scan, will always supersede any "mark as free" operation. We call this procedure as occupied superseding. However, this safeguard also introduces execution time overheads, which may not be necessary — as we will show in the evaluation section. For that reason, this is an implemented feature with optional use.



(a) DDA (b) Bresenham

Figure 4.7: Grid ray casting with line drawing algorithms, visited cells being in gray. In (a), the Digital Differential Analyzer (DDA) algorithm marks all grid cells that are traversed by the ray. In (b), the Bresenham algorithm follows the direction of the slope and only marks only one grid cell per (integer) step, which corresponds to the grid cell with higher ray coverage.

4.4 Evaluation

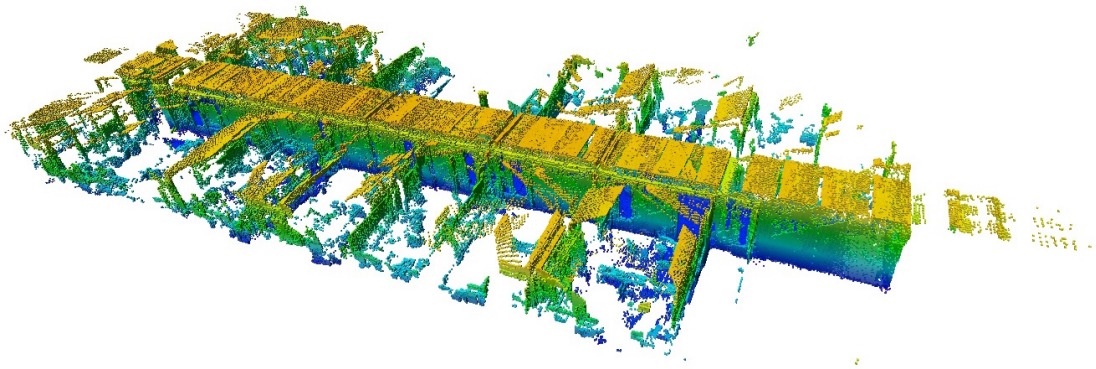
The evaluation of our volumetric grid approach is supported by several real-world datasets using the probabilistic occupancy model. The datasets in question are the Freiburg Building 079 corridor (Fr-079), Freiburg Campus (Fr-Campus) and the New College - Epoch C (N. College). The first and second dataset represent an indoor and an outdoor environment, respectively, with 360º laser scans (laser on a pan-and-tilt unit). The last dataset is an outdoor environment with laser data provided by two lasers mounted vertically. Additionally to the laser data, a ground truth of the robot's pose is also provided. The datasets were made available by Hornung et al. (2013) and are part of Octomap evaluation[5]. These datasets allows us to directly compare our approach with Octomap.

4.4.1 Mapping with Known Poses

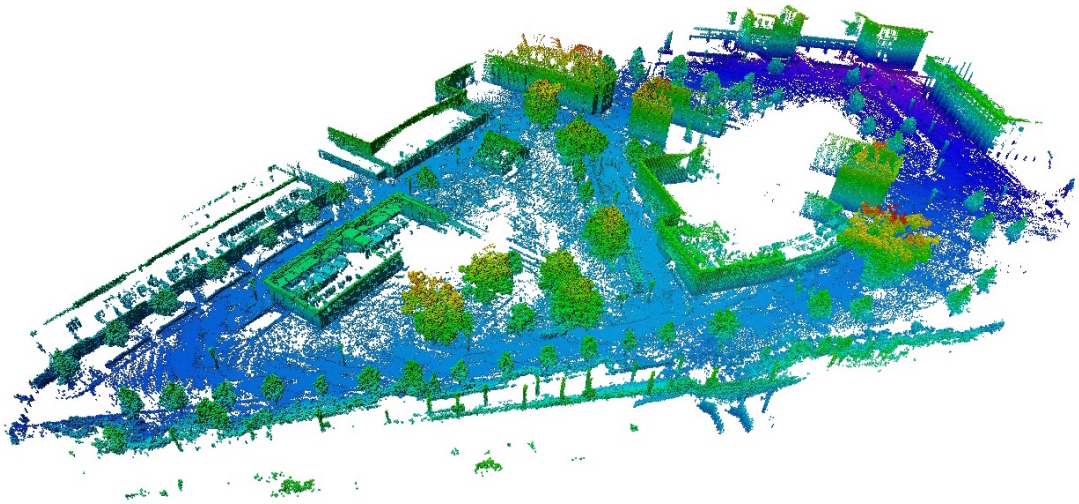
The first step to validate our volumetric grid solution is to evaluate its ability to provide the necessary grounds to construct a map of the environment for each dataset. Using the available datasets, the laser data is integrated using the implemented probabilistic occupancy map. With the availability of a ground truth, the construction of the environment is simplified to a problem of mapping with known poses, using the approaches described in Subsec 4.3.5. Here, the correctness of the obtained maps, depicted in Figure 4.8, is subjective to the visual observation of the reader. The expected outcome are maps that present a credible structure of the environment without artifacts that could hint to inconsistencies in the underlying data structure, i.e. in our sparse-dense volumetric data structure.

From our observation, a coherent and structured map is obtained for the three datasets. From the Fr-079 indoor dataset it is possible to clearly identify the corridor structure contained in this dataset, while in the outdoor datasets, Fr-Campus and N. College, building facades and tree structures are certainly visible.

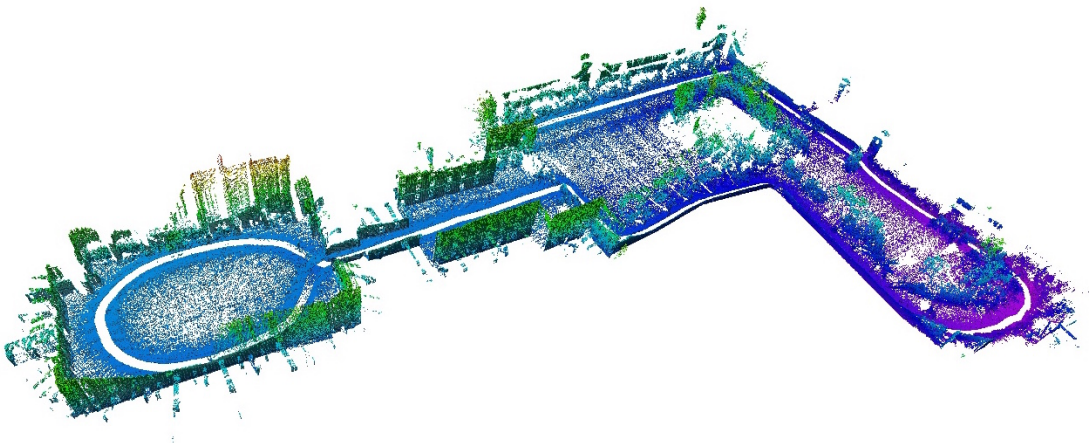
[5] `http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/`



(a) Fr-079 at 0.05m resolution (size of scene: $46.45 \times 36.4 \times 4.4$ m^3)



(b) Fr-Campus at 0.2m resolution (size of scene: $292.4 \times 167.1 \times 27.7$ m^3)



(c) N. College at 0.2m resolution (size of scene: $161 \times 249.7 \times 33$ m^3)

Figure 4.8: Resulting 3D occupancy maps of all datasets with the proposed volumetric grid. For clarity, free space is not depicted and occupied grid cells are color coded by height.

4.4.2 Map Accuracy

We seek to evaluate the accuracy of the obtained 3D occupancy map using our approach and how it compares to OctoMap. Following Hornung et al. (2013) proposal, the accuracy is measured as percentage of correctly mapped grid cells for the entire 3D scan data. A grid cell is considered to be correctly mapped if its state (free or occupied) in the map is the same in the evaluated scan ray. In practice, every scan ray is treated as an integration on an already built map where the grid cells along the sensor origin and ray endpoint must be free and the grid cell at the endpoint (of the scan ray) must be occupied. Accompanying the accuracy is the cross-validation of the map where each fifth scan ray is skipped when building the map but then is used to evaluate the percentage of correctly mapped grid cells, i.e. 80% of data is used to build the map and the remaining 20% is used to measure the accuracy of the map.

The obtained results are presented in Table 4.2. They show that our solution, which we call SDMapping, is capable of representing the environment with considerable accuracy and is on a pair with Octomap. The residual errors can be justified by sensor noise and discretization effects introduced by the grid representation and also ray casting. Switching from DDA to Bresenham does not introduce a significant decline in accuracy, even though it fails to visit all grid cells traversed by a scan ray, and in fact, in some situations it actually increases accuracy. We can conclude that Bresenham can be used to reduce computational effort without loosing significant accuracy. Disabling occupancy superseding incurs in an increase of accuracy but at a cost of possible side effects in the resulting map as previously discussed. Nonetheless, it offers an increase in time efficiency that can be advantageous if the side effect of disabling occupancy superseding is minimal.

Table 4.2: Map accuracy and cross-validation. Both metrics are presented as percentage of correctly mapped grid cells for the 3D scan data (for each dataset). Several strategies are evaluated, which includes our volumetric grid proposal with different ray casting algorithms complemented by occupied superseding (enable or disabled) and OctoMap. The best values are shown in bold.

Dataset	**Accuracy (%)**					**Cross-Validation (%)**				
(strategy)	1		2		3	1		2		3
	a	b	a	b		a	b	a	b	
Fr-079	97.72	98.16	97.95	**98.23**	97.30	97.65	**98.01**	97.48	97.69	96.25
Fr-Campus	97.80	**98.81**	97.33	98.29	97.73	97.98	**98.59**	97.26	97.91	95.74
N. College	98.54	98.77	98.68	**98.84**	98.63	98.42	98.67	98.53	**98.70**	98.34

1, 2 Our proposal with DDA and Bresenham, respectively.
3 Octomap (which uses DDA and occupied superseding).
a, b With occupied superseding enabled and disabled, respectively.

4.4.3 Optimal Density

In the proposed volumetric grid, the size of the *patch* edge, designated L, defines the density of a *patch*. As discussed in subsection 4.2.2, the value of L offers a trade-off between space and time efficiency. This raises the following question: what actual value of L provides the best balance between space and time efficiency? To answer this question we carried out the *mapping with known poses* experiment for different L values and recorded the total execution time and memory usage. The obtained results, are shown in Figure 4.9.

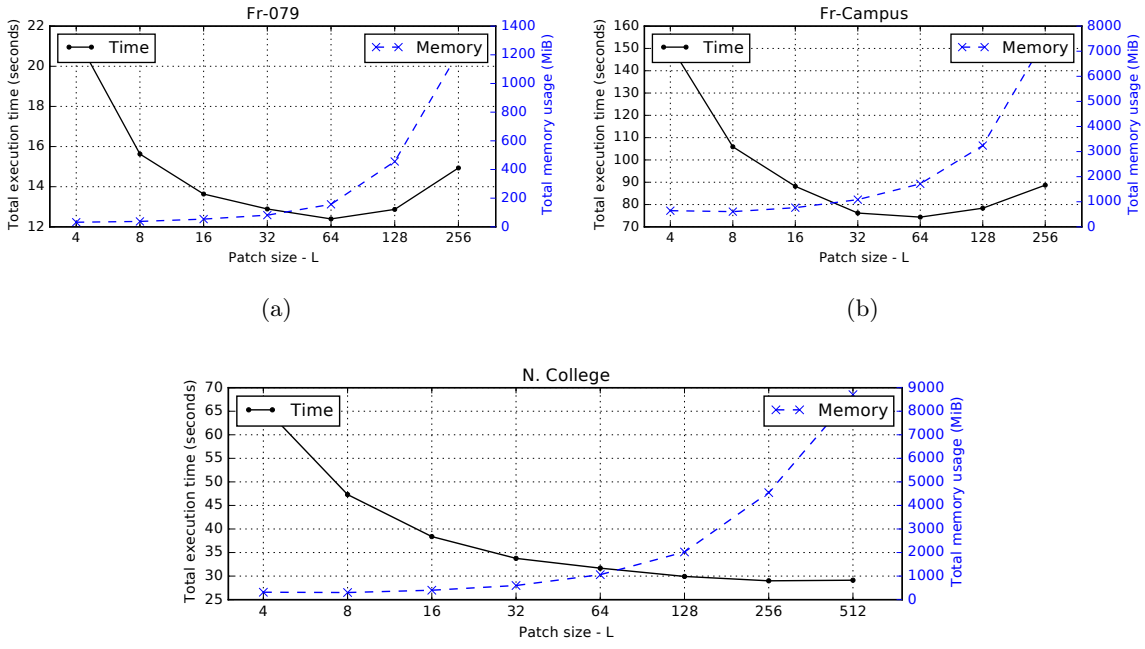


Figure 4.9: The impact of *patch* size L in execution time (left axis) and memory usage (right axis).

From the analysis of these results we can infer a value of L that provides the optimal density, i.e. the *patch* density that offers the most balanced execution time and memory efficiency. As expected, as the value of L increases, the total execution time decreases and the total memory usage increases. Unlike the memory usage, the total execution time is not monotonic in all datasets. Since up to $L \leq 64$ the *patch* memory size is lower than the processor cache size, our educated guess is that for higher values of L the execution times increases due to a higher number of processor cache misses. This explanation could be defeated by the dataset where this does not happen, as a counterexample. However, the data in that dataset is arranged as vertical scans that are cache friendly due to our columnwise flat array memory organization, therefore our hypothesis is still valid.

The actual choice of a value for L, supported by the results shown in Figure 4.9, is, in our opinion, a subjective decision – it depends on the application. For applications where memory usage is the priority, a lower value for L should be used, but when execution times is the main concern, a higher value for L is the best option. For our applications we found that $L = 32$ provides a good balance between execution time and memory usage, also, it is the value that is used through the ubiquitous usage of the proposed volumetric grid in this thesis.

4.4.4 Time and Space Efficiency

The objective of the following experiments is to analyse the execution times and memory usage, including lossless data compression and the impact of cache size, and how they are comparable with OctoMap. Data was obtained by carrying out *mapping with known poses* experiments for the datasets Fr-079 and Fr-Campus. For reference, all experiments were run in a MacBook Pro with an Intel Core i7-2640M 2.8GHz and 16GB of RAM.

Optimal Cache Size in Online Data Compression

Before going into the analysis of time and space efficiency, we need to address yet another parameter that has a direct impact in execution times and memory usage, the cache size C used during online data compression. Although only relevant when data compression is used, its proper definition is necessary to validate the usability and overall advantage of online data compression. The objective of this system is to reduce the memory usage as much as possible with the lowest possible impact in computations, in the most desirable situation the overall execution time overhead would be negligible. We used *mapping with known poses* with different values for C to record the update times and the rate of cache misses – accessing a grid cell that is part of a *patch* that is not in the cache list is a cache miss. The obtained results are shown in Figure 4.10.

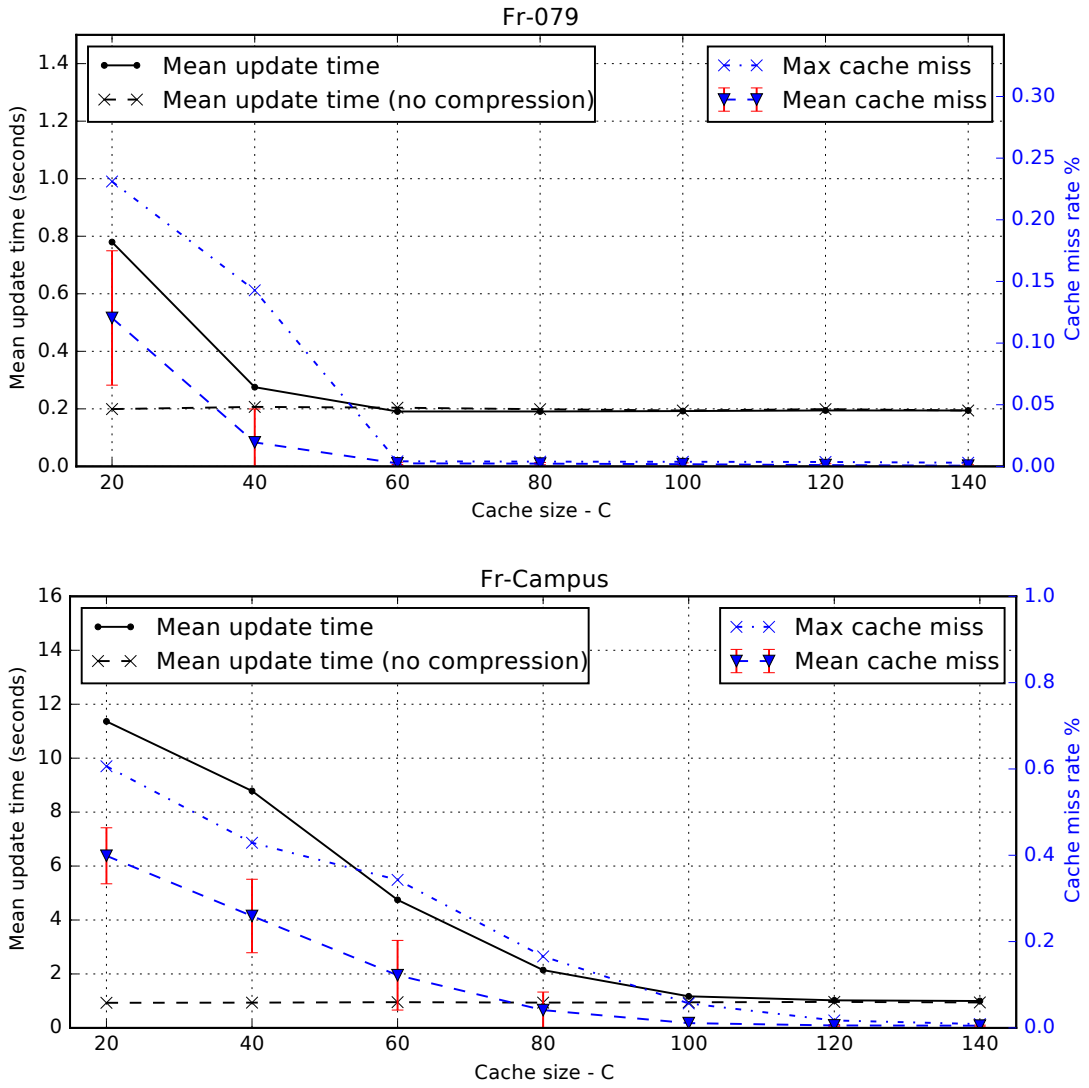


Figure 4.10: The impact of online compression cache size C in update time (left axis) and cache miss rate (right axis). As the cache size value increases the cache miss rate approximates to zero and the mean update time has a fast asymptotic behaviour towards the mean update time without online compression.

From the analysis of these results we can infer a value for C that has the lowest impact in update times. As hypothesized, as the cache size value increases, the update time overhead introduced by the online compression diminishes with an asymptotic behaviour towards the mean update time without online compression. This is the most desirable behaviour, because it allows us to further reduce the memory footprint of the volumetric grid without compromising time efficiency.

The cache miss rate is also a metric worth analysing. A zero rate is the obvious asymptote that can only be reached when compression is not in place. Therefore, it is no surprise that the optimal cache size value has a cache miss rate close to zero. However, "close to zero" is subjective and may well depend on external factors that we are not considering. Nonetheless, we observed that optimality is achieved when the miss rate variance is at its minimum, and a visible consequence of such is a maximum cache miss rate similar to the mean miss rate.

The obtained results show that the optimal value for C depends on the data being used, i.e. different environments result in different optimal cache sizes. This is due to the properties of the data such as the number of rays per scan, the mean range per scan and how close (in space) two consecutive scans are when gathered. The optimal cache size C is, therefore, a value that is tailored to individual datasets that needs to be obtained through experimentation. However, the cache size C has an asymptotic effect in update times as it grows in value. By overestimating C we can trade a fixed but manageable increase of memory for the need to find its optimal value through experiment. For the remaining experiments that use online data compression a value of $C = 200$ was used.

Execution Times

The impact (or significance) of this metric depends on *when* the measurements are integrated: soon after each measurement is obtained or offline after all data are collected. In the former scenario time efficiency is important, as, if the current measurement is not processed before the next one is available, information will probably be lost and as consequence the environment can be misrepresented. In the latter scenario time is not an issue, as measurements are integrated sequentially and no information is lost, nonetheless one can argue that a total execution time of one minute is better than one hour. Despite any scenario, achieving the best possible execution times is always desirable, for example it reduces computational requirements and operational times such as development-test cycles.

In order to assess the execution times of our solution we use OctoMap's times as baseline. The objective is to evaluate the speedup that our proposal offers when confronted against the current state-of-the-art. The execution times of every experiment is quantified by the wall time it takes to update (or integrate) a complete scan, which we call *update time* and is denoted as s_{update}. The total execution time is now given by

$$s_{\text{total}} = \sum_{k=1}^{N} s_{\text{update}}^{k} \tag{4.13}$$

where N is the total number of scans. The total execution time s_{total} is then used to calculate the speedup of a strategy:

$$S^{i} = \frac{s_{\text{total}}^{i}}{s_{\text{ref}}} \tag{4.14}$$

where i is the strategy index and s_{ref} is the reference total execution time, that is, the total execution time of Octomap. The obtained results are shown in Table 4.3.

Table 4.3: Mapping execution times for the datasets Fr-079 and Fr-Campus. The mean update time, total execution time and speedup are presented for different strategies, namely OctoMap and our solution with and without online data compression.

Dataset	**Strategy**	$\bar{s}_{\text{update}}$ (s)	s_{total} (s)	**Speedup**
Fr-079	Octomap	0.770 ± 0.146	50.858	1.0x
(5cm res.)	SDMapping	0.521 ± 0.052	34.411	1.4x
	SDMapping-fast	0.207 ± 0.033	13.699	3.7x
	SDMapping-fast (lz4)	0.216 ± 0.041	14.278	3.5x
	SDMapping-fast (zstd)	0.209 ± 0.040	13.83	3.6x
Fr-Campus	Octomap	6.365 ± 1.756	515.571	1.0x
(10cm res.)	SDMapping	2.750 ± 0.573	222.808	2.3x
	SDMapping-fast	0.957 ± 0.171	77.565	6.6x
	SDMapping-fast (lz4)	0.972 ± 0.187	78.768	6.5x
	SDMapping-fast (zstd)	1.179 ± 0.262	95.564	5.3x

When comparing with OctoMap there is a clear speedup in every variant of our solution, which implies a better all-around time efficiency. The base variant of our solution, named SDMapping, utilizes the same techniques for data integration as OctoMap, both using DDA and occupied superseding. Therefore, by providing a better time efficiency we can infer that our underlying space management structure is responsible for the speedup. Additionally, a variant of our mapping solution with Bresenham and without occupancy superseding, named SDMapping-fast, provides a further speedup without losing accuracy (as shown in Table 4.2). A detailed view of these update times is depicted in Figure 4.11

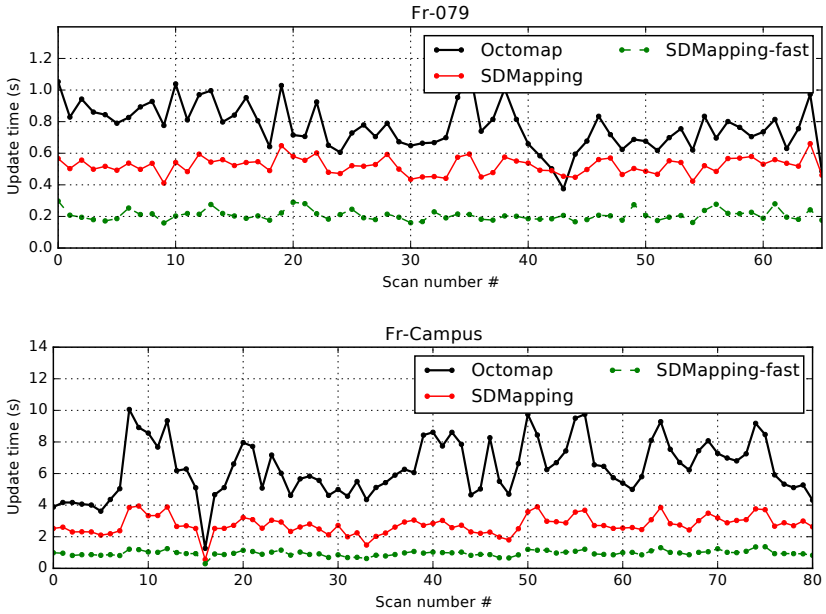


Figure 4.11: Amount of time it takes to update each scan for the datasets Fr-079 and Fr-Campus. It includes the update times of OctoMap and SDMapping (our proposal). For our proposal we include an experiment with data integration similar to OctoMap (DDA for ray casting and occupancy superseding) and another experiment with fast data integration (Bresenham for ray casting and without occupancy superseding).

To further reduce the amount of memory used by our proposal, we have introduced (and implemented) an online data compression mechanism (subsection 4.2.2). It was developed to maximize the space gain while minimizing the execution time overhead. From the obtained results that goal was achieved. The update time overhead is noticeable but it is not significant considering the space gain that it offers — as it will be shown ahead. Figure 4.12 depicts in more detail the impact of the online data compression during data integration. It also shows that even with a compressor like **zstd**, that trades execution times for better compression ratios, the time overhead is negligible.

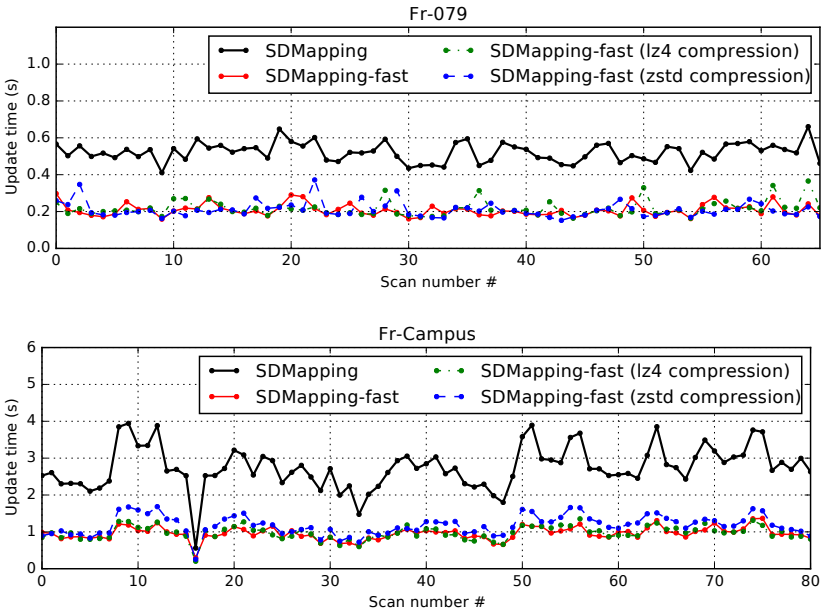


Figure 4.12: The impact of online data compression in the update time of fast data integration. The update times of the base version of data integration is presented for comparison purposes.

Memory Usage

The memory usage is a metric that can not be overlooked. In applications such as mapping, the more information is obtained the more memory is required to hold it – assuming that information is not discarded. The memory usage of the regular volumetric grid to store the probabilistic information of an occupancy map is given by

$$m_{\text{grid}} = \frac{x \times y \times z}{r^3} 4\text{Bytes} \tag{4.15}$$

where x, y and z are the size of the minimal bounding that covers the mapped area and r is the resolution of the map. For large environments the volumetric grid would require a considerable amount of memory that could go beyond the capacity of the computational system. Therefore, any volumetric grid solution has to considerer space efficiency in its implementation and the less memory it uses the better.

Once again, to assess the memory usage of our solution we will use OctoMap's memory usage in a 64-bit architecture as baseline. The objective is to evaluate the memory gain that our solution offers when compared with the current state-of-the-art. The memory usage of

every experiment is quantified by the memory usage after the update (or integration) of each scan. The total memory usage is defined as m_{total} and is the memory usage after the last scan integration. The gain in memory is the memory savings relative to the baseline and is defined by

$$G^i = 1 - \frac{m^i_{\text{total}}}{m_{\text{ref}}} \tag{4.16}$$

where i is the strategy index and m_{ref} is the reference total memory usage, that is, OctoMap's total memory usage. The memory usage reported by our solution, without compression, is equal to

$$m_{\text{total}} = n_p(L^3 \times 4\text{Bytes} + H) \tag{4.17}$$

where n_p is number of *patches*, L is the *patch* size and H is an estimated memory overhead of the sparse-dense structure. When data compression is used, the reported value is the sum of the compressed data size of each *patch*, which includes the memory overhead H, plus the cache size. The current estimated overhead is 96 Bytes but it is possible to be an underestimation, nonetheless the calculated values are not very different from the ones reported from the system. The calculation of OctoMap's memory usage is available at Hornung et al. (2013). The obtained results are shown in Table 4.4 and both our solution and OctoMap have better space management than the regular volumetric grid, as it was expected.

Table 4.4: Total memory usage for the datasets Fr-079 and Fr-Campus. The total memory usage and gain are presented for different strategies, including OctoMap and our solution with and without online data compression. In the indoor dataset, Fr-079, OctoMap and our solution have similar memory usage but in the outdoor dataset, Fr-Campus, our sparse-dense structure provides a significant cut in memory usage. When using online data compression our solution provides a significant space gain in both datasets.

Dataset	**Mapped area** (m^3)	m_{grid} (MB)	**Strategy**	m_{total} (MB)	**Gain**
Fr-079	$46.45 \times 36.4 \times 4.4$	170.27	Octomap	84.93	0%
(5cm res.)			SDMapping	81.55	3.97%
			SDM-fast	81.55	3.97%
			SDM-fast (lz4)	28.55	66.37%
			SDM-fast (zstd)	27.01	68.19%
Fr-Campus	$292.4 \times 167.2 \times 27.8$	5184.63	Octomap	1982.75	0%
(10cm res.)			SDMapping	1085.79	45.23%
			SDM-fast	1084.79	45.28%
			SDM-fast (lz4)	144.44	92.71%
			SDM-fast (zstd)	86.83	95.62%

The datasets that were used represent an indoor and an outdoor environment. In the indoor dataset, Fr-079, our solution (without online data compression) and OctoMap, have similar memory usage but with advantage to our solution when execution times are also considered, which are shorter, specially in the *fast* variant of the solution. The difference in memory usage then widens considerable when online data compression is used. Consequently, this proves our hypothesis that a *patch* can have significant data redundancy for an efficient usage of data compression algorithms. This statement if further supported by the results of the outdoor dataset, Fr-Campus, where the memory gain is above 90%. Additionally, observing that our solution without online data compression in the same dataset holds a considerable memory gain (approximately 45%) it is then sensible to assume that the difference in memory usage between our solution and OctoMap can be justified by the memory overhead of the octree

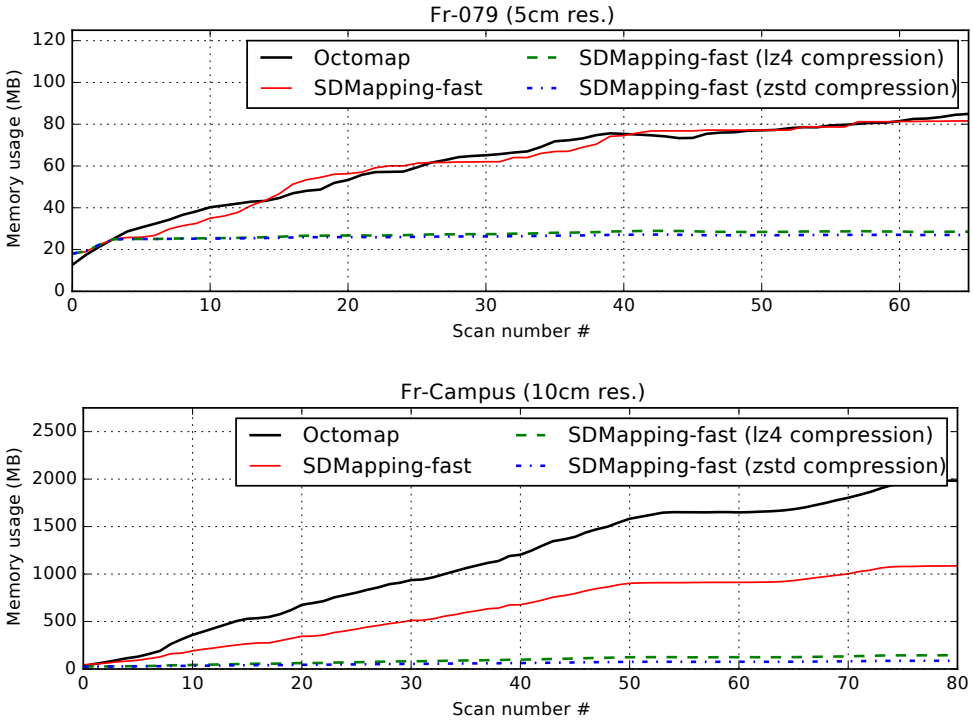


Figure 4.13: Memory usage of our solutions (SDMapping-*) and OctoMap after each scan update.

structure. Our findings are further supported by a detail view of memory usage after each scan update (Figure 4.13).

4.5 Conclusion

This chapter presents a Sparse-Dense framework for efficient mapping, which we named SDMapping. A regular volumetric grid is a popular method to represent the environment, however for large three-dimensional environments it requires a large amount of memory that may not even be available. Overtime, several solutions were proposed to address the issue of memory management such as elevations maps, multi-level surface maps and point clouds. Nevertheless, they all have several limitations for three-dimensional application that lead to the creation of OctoMap by Hornung et al. (2013), an efficient state-of-the-art probabilistic three-dimensional mapping framework based on octrees.

OctoMap, like any solution before it, has its limitations, as for example: no explicit support for two-dimensional mapping; no complete independence from the semantics of probabilistic occupancy mapping; further memory compactness achieved by a lossy method; noticeable memory overhead for large environments; and no efficient method to duplicate a map when needed. Due to this limitations OctoMap did not provided the necessary functionalities for our SLAM applications. Therefore we developed SDMapping, a volumetric grid space representation managed by a sparse-dense structure with efficient execution times and memory management that provides support for any model. It includes features such as: the separation between space management and model; the recognition that not all applications re-

quires a three-dimensional representation of space, thus allowing the user to also choose a two-dimensional representation of space that increases space efficiency; an online data compression mechanism that further reduces memory usage by taking advantage of the statistical redundancy present in the data of the dense blocks of our structure that is managed by an LRU cache mechanism; and an implicit data sharing structure that efficiently duplicates the data of a map using a copy-on-write strategy.

To evaluate our solution we conducted several probabilistic occupancy mapping experiments with publicly available datasets and compared their results with OctoMap's, namely mapping accuracy, execution times and memory usage. Overall, our solution provides a similar accuracy to OctoMap while providing a better time and space efficiency and additional features that covers the requirements of our applications. Furthermore, we also showed that for mapping the Bresenham algorithm is a viable alternative to DDA that offers faster execution times without losing accuracy.

Chapter 5

Improved Grid-based SLAM

Having the proper and correct models of the environment is a crucial part of many robotic applications as they are used by robotic agents to adapt their decisions to the current state of the environment. There are different models that can be obtained by sensing the environment but one of the most important is the one that allows the robot to infer its most likely pose in the environment in which it is operating. Such model is usually known as the *map* of the environment and a popular method to create it is to integrate sensor data while the robot explores the environment. This process is known as *Simultaneous Localization and Mapping* (SLAM) (Smith and Cheeseman, 1986; Thrun, 2002).

Having only access to measurement $z_{1:t}$ and control $u_{1:t}$ data, SLAM has to obtain the map that best represents the environment. That is, in addition to estimate the pose of the robot, it has to evaluate the spatial relation between the robot's own poses and estimated objects in the map, making it a complex problem to solve. Moreover, it has to handle with the fact that measurement and control data are noisy and easily disrupted in dynamic environments. The representation of objects in the map can be landmarks in a feature-based representation or the environment shape captured by a range finder in a grid-based representation. SLAM solutions have been using both feature-base and grid-based map representations with proficiency and each representation has unique characteristics that defines how the SLAM problem is solved.

Feature-based representations require the use of maps that maintain the list of perceived landmarks. One of the earliest SLAM solution to use this type of representation is the EKF SLAM (Smith and Cheeseman, 1986) that in essence is the EKF localization algorithm applied to *online* SLAM using maximum likelihood data association (e.g. Neira and Tardos (2001)). But this approach has several limitations. The number of tracked landmarks does not scale well due to the computational requirements of the sparse map that maintains the conjoint position of the landmarks. Furthermore, the Gaussian noise assumption in this method has the tendency to introduce unrecoverable errors if the level of landmark ambiguity is too high. To reduce the ambiguity in landmark detection, artificial landmarks or detectors with higher sophistication are commonly used (Thrun et al., 2005).

The FastSLAM algorithm (Montemerlo et al., 2002) is another solution that estimates landmarks with EKFs but now with a *Rao-Blackwellized particle filter* (RBPF) approach (Murphy, 1999; Doucet et al., 2000). It allows the estimation of the robot's pose separated from the landmarks estimation. In addition to the robot's pose, calculated by sampling, each particle contains several low-dimensional EKF that are used to estimate the state of each individual landmark that reduces the complexity of updating information and allows the use

of negative information to reduce the number of landmarks overtime. As consequence, the filter maintains several maps with different data associations and not only the most likely one, which is an approximation to the full posterior. This categorizes the FastSLAM as *full* SLAM but because a particle filter estimate a pose at-a-time it also fits in the *online* SLAM category (Thrun et al., 2005). Despite the advantage of FastSLAM over EFK SLAM, its efficiency also depends on the quality of data association (i.e. landmark detection ambiguity, virtual landmarks) and a manageable number of particles to find a solution.

GraphSLAM (Grisetti et al., 2010a) is another SLAM method that uses a feature-base representation and is capable of calculating a solution over all poses and all features in the map, hence it is labeled as a *full* SLAM solution. The localization and mapping problem is defined by a graph with nodes and likelihood constraints (or arcs) between those nodes. The nodes represent robot poses and landmarks while constraints represent *motion links* between two consecutive poses and *measurement links* between landmarks and the pose where they were observed. A solution is then found by minimizing the negative log likelihood of all constraints which results in a non linear least square problem. Typically, a solution is found offline for two reasons: the necessary data is only available after exploring the environment and the computational requirement is usually too high to be used during operations. The mapping quality is often high with GraphSLAM but like every method that uses a feature-base representation it also depends on the quality of landmark detection and data association.

The Grid-based representation, or more specifically *occupancy grid* map as a volumetric representation of the environment, has the advantage over feature-based maps of not requiring a prior definition of any landmark. Instead, it uses the natural features present in the environment (e.g. walls, furniture, pillars) to map the arbitrary shape of that environment. Localization algorithms based on scan matching have been used successfully in SLAM and its initial usage can be tracked to Lu and Milios (1997a). Its main advantage lies in the lower computational complexity that it requires when compared to other localization algorithms such as histogram and monte-carlo localization. Its efficiency relies on good inputs and a search bounded by small deviations of scans. However, these conditions are not always met, hence several scan matching variations have been proposed over the years in order to improve the localization process. For example: Biber and Strasser (2003) uses the Normal Distribution Transformations (NDT) with several grids to match range scans; (Kohlbrecher et al., 2011) uses an inverse occupancy map pyramid with an interpolated occupancy lookup to funnel the scan matching and reduce the effect of discretization; and Olson (2009, 2015) proposes a global scan matching with multi-resolution maps for a global coarser search. All of this grid-based SLAM proposals fall into the *online* SLAM category, meaning that if something goes wrong they may never recover.

Scan matching localization algorithms have the tendency to accumulate errors but they are usually corrected by the map. But in SLAM, the map is the also the result of localization and it will reflect the accumulated errors. In environments without loops, or even with small loops, those errors may not manifest but that is not guaranteed in environments with large loops. Developed by Eliazar and Parr (2003), DP-SLAM picks on the ideas of FastSLAM and extends them to occupancy grid maps. Each particle contains the pose of the robot, which is sampled from the robot motion, and a full occupancy grid of the environment. However, an actual full occupancy grid for each particle is not feasible (memory wise), therefore, DP-SLAM employs a volumetric grid where each cell contains an *observation tree* of the particles that changed the cell. The method also requires the maintenance of an *ancestry tree* of particles so that a particle can find the ancestor that made the most recent change to the same cell. Although

DP-SLAM is capable of producing occupancies grid of high quality it has a noticeable problem: the necessary amount of particles is considerably high, e.g. 9000 as reported by Eliazar and Parr (2003).

To reduce the number of particles and possible particle depletion (Van Der Merwe et al., 2000) in Rao-Blackwellized approaches, Grisetti et al. (2005) introduced a better proposal distribution that considers the accuracy of the sensor being used (such as LIDARS) and an adaptive resampling technique that maintains a reasonable variety of particles to avoid depletion. The novelty in the proposal distribution is the use of scan matching to obtain the most likely particle-dependent pose to sample from, which provides a more accurate model than just using the last odometry reading as in, for example, DP-SLAM. Combined with the adaptive resampling technique, this approach considerably reduces the necessary number of particles to obtain a viable map (e.g. 15 or 30 particles). Additionally, it maintains a low memory footprint by sharing duplicated data of the occupancy grids among the particles. This SLAM approach has become one of the most popular grid-based SLAM solution and its reference implementation is known as GMapping[1].

Although GMapping improved the efficiency of RBPF SLAM, we will show in this chapter that such efficiency can be further improved, most notably the execution times required for localization and mapping. This development is based on a better scan matching algorithm for localization, a more flexible space management data structure for mapping and a multi-threading mechanism that takes advantage of the independence between particles to speedup execution time. The localization algorithm that we propose is the one already presented in chapter 3 but with the necessary changes for a map that is being built incrementally. The space management data structure for mapping was also presented in this thesis in chapter 4. Its sparse-dense structure with implicit data sharing and online data compression is used to improve space efficiency while retaining the time efficiency of the SLAM solution. Applying multithreading to GMapping has already been tried (Gouveia et al., 2014), however, only the scan matching process is parallelizable in this study. The map update process has to maintain a sequential order because its mapping data structure fails when multithreading is used. Because our mapping data structure does not have this limitation (see subsection 4.3.4) it is possible to parallelize the scan matching and map update processes to reduce the execution time.

The RBPF SLAM has an interesting characteristic: although it is categorized as a *full* SLAM solution, the particle filter estimates one pose at-a-time which is by itself an *online* SLAM solution (Thrun et al., 2005). In essence, each particle has an independent scan matching localization process followed by the integration of the current measurement into its own occupancy grid map. This opens the opportunity to first develop a self contained *online* SLAM solution, based on our scan matching algorithm, that can be later used as an integral part of an RBPF SLAM solution. Therefore, this chapter focuses on the development of two grid-based SLAM solutions that cover both *online* and *full* categories.

[1] `https://github.com/OpenSLAM/slam_gmapping`

5.1 An Improved Scan Matching Approach to *Online* SLAM

In probabilistic terms, the SLAM problem can be posed as the posterior estimation of the transient pose together with the map:

$$p(\mathbf{x}_t, m \mid z_{1:t}, u_{1:t}) \,, \tag{5.1}$$

where $\mathbf{x}_t$ is the pose of the robot at time t and m is the map. This approach is named *online* SLAM because only variables that persist at time t are considered and therefore estimated. As result, solutions for this type of SLAM discard past measurement and control data after they have been processed which attenuates the complexity of the problem. Additionally, as showed by Thrun (2001), the formulation in (5.1) can be factorized into:

$$p(\mathbf{x}_t, m \mid z_{1:t}, u_{1:t}) = \underbrace{p(\mathbf{x}_t \mid z_{1:t}, u_{1:t})}_{\text{localization}} \underbrace{p(m \mid \mathbf{x}_{1:t}, z_{1:t})}_{\text{mapping}} \tag{5.2}$$

where the first factor is the pose estimation and the second factor is the map creation. In this factorization both factors are independent and, as result, *online* SLAM can be treated as a problem of localization and then mapping with known poses. In practice, the localization procedure is carried out using the map built so far, and then the map is updated using the obtained pose estimation.

5.1.1 Scan Matching Localization with a Dynamic Likelihood Field

Scan Matching based localization algorithms are a good fit for *online* SLAM (e.g. Biber and Strasser (2003); Holz and Behnke (2010); Kohlbrecher et al. (2011)). They have low computational requirements and are capable of providing good results as long as measurements are sufficiently accurate and big loops are non existing. In scenarios where we can assume that successive poses are close to each other and large loops do not exist, scan matching localization algorithms can be accurate enough to assist in the construction of a globally consistent environment. Fortunately nowadays, sensors (such as LIDARS and odometers) provide accurate readings that make scan matching solutions viable for SLAM applications. Additionally, not all environments have big loops, therefore, a solution that offers good accuracy with low computational requirements such as scan matching is a viable choice.

In chapter 3, we introduced a fast and accurate localization algorithm based on scan matching, and we hypothesize that it can be used in the localization process of an *online* SLAM solution and benefit from its speed and accuracy. Our localization solution utilizes a likelihood field model and expects an priori map from which the likelihood is computed into a discrete grid. In essence, the estimation of the robot pose x_t with this model is the result of several roto-translations for likelihood look-ups, performed in a precomputed likelihood table, during the optimization process to find the local minimum. The created table is expected to be static over time, therefore, the overhead introduced by its creation is negligible because it happens only once.

In SLAM, the map m is built incrementally and as consequence the map changes overtime and the static assumption of the likelihood table is no longer valid. For the likelihood field to work with an incremental map it must be rebuilt each time the map topology changes. Because we introduced a continuous likelihood field, it is not the likelihood that is cached in a grid, but instead an Euclidean distance grid. Therefore, the Euclidean distance grid, denoted

$\hat{\varepsilon}$, is what we need to recalculate. If the map m has a dimension $i \times j$ then the calculation of $\hat{\varepsilon}$ can run in linear time bounded by $O(ij)$ (Felzenszwalb and Huttenlocher, 2004). But as the map grows in size, there will be a dimension $i \times j$ such that the computation time of $\hat{\varepsilon}$ is bigger than the period of measurements. Furthermore, a single change in the map m triggers a full calculation of $\hat{\varepsilon}$. To address the shortcomings of the likelihood field in a SLAM context, we propose the development of a dynamic likelihood field capable of adjusting to incremental changes.

To solve the execution time degradation we seek to employ an incremental strategy to construct the Euclidean distance grid $\hat{\varepsilon}$, an integral part of the likelihood field. The reasoning behind an incremental update is that changes to the topology of the map m are local, therefore the update of the Euclidean distance are, for the majority of cases, bounded to a local area. The dynamic Euclidean transform proposed by Lau et al. (2013) is a good fit for the proposed incremental strategy. It updates only the cells affected by the change of state, i.e. from occupied to free and vice versa, involved in the update of the occupancy grid map that reflects the current perceived state of the environment. The algorithm expects the information about the change of state to be provided by the caller. This information can be created by tracking the changes of state that occur at each update of the map m. Additionally, the maximum Euclidean distance $\hat{\varepsilon}_{\max}$ defines an upper bound to the propagation of the Euclidean distance change to an object in the map. This is a trade-off between information and computation —the lower the distance the faster the update.

5.1.2 Incremental Mapping

In this approach, the environment is represented by a probabilistic occupancy volumetric grid that captures its shape. Typically, the occupancy probability of each cell is modelled by a binary random variable that specifies whether a cell is occupied or free (see subsection 2.3.1). But localization can be improved by using reflection probability grids that model the occupancy probability by the relation between the number of *hits* and *misses* that are counted for each cell (Bennewitz et al., 2009). The number of *hits* corresponds to the number of times an endpoint of a ray hits (or is reflected) by an object in that cell and the number of *misses* matches the number of times a cell is visited by a ray without a hit (or reflection). The cells that are visited by a ray are calculated through ray-casting using the Bresenham algorithm (Bresenham, 1965). In this reflection model the occupancy probability of a cell, defined by p_{cell}, at location $\langle x, y \rangle$, is given by

$$p_{cell}(x, y) = \frac{hits_{x,y}}{hits_{x,y} + misses_{x,y}} , \tag{5.3}$$

and a cell is assumed to be occupied when its probability is above a certain value

$$\text{occupied}(x, y) = \begin{cases} \text{true}, & \text{if } p_{cell}(x, y) > occ_{\text{thresh}} \\ \text{false}, & \text{otherwise} \end{cases} \tag{5.4}$$

Here, occ_{thresh} is the threshold that defines the occupancy and in our solution we used a value equal to 0.25. The implication of this value is that a reflected object has higher persistence in the occupancy map than visited free space.

As discussed in the previous section, the localization algorithm uses a dynamic likelihood field as measurement model that is supported by an Euclidean distance map with dynamic

adaptation to the changes. The mapping procedure has to construct not one but two maps: an occupancy grid map and an Euclidean distance grid that reflects the current changes to the former grid map. The pseudocode of the incremental mapping process is shown in algorithm 17 and the dynamic update of the Euclidean distance — updateDistanceMap — was already presented in Section 4.3 (algorithm 16) as part of the implemented models with the sparse-dense mapping framework. The volumetric grids used for mapping are implemented with the sparse-dense mapping framework to take advantage of its efficient space management, including transparent size growth and online data compression.

Algorithm 17: Incremental mapping process.

updateMaps$(\mathbf{x}_t, m, \hat{\varepsilon}, z_t)$
- $z \leftarrow \mathbf{x}_t \oplus z_t$ `// Use world coordinates`
- $\mathbf{x}_o \leftarrow$ SensorOrigin()
- $O \leftarrow \emptyset$ `// Occupied cells`
- $F \leftarrow \mathbf{x}_o$ `// Free cells`
- **foreach** $\langle x, y \rangle \in z$ **do**
  - $O \leftarrow O + \langle x, y \rangle$
  - `// ray casting`
  - Bresenham$(\mathbf{x}_o, \langle x, y \rangle, F)$
- **end**
- `// Update occupancy and track changes`
- $O_t \leftarrow$ updateOcc(O)
- $F_t \leftarrow$ updateFree(F)
- `// Update the Euclidean distance map`
- **foreach** $\langle x, y \rangle \in O_t$ **do** setOccupied$(\hat{\varepsilon}, \langle x, y \rangle)$
- **foreach** $\langle x, y \rangle \in F_t$ **do** setFree$(\hat{\varepsilon}, \langle x, y \rangle)$
- updateDistanceMap(ε)

updateOcc(O)
- $C \leftarrow \emptyset$ `// List of changed cells`
- **foreach** $\langle x, y \rangle \in O$ **do**
  - $state \leftarrow$ occupied(x, y)
  - $hits_{x,y} \leftarrow hits_{x,y} + 1$
  - **if** $\neg state \wedge$ occupied(x, y) **then**
    - $C \leftarrow C + \langle x, y \rangle$
- **end**
- **return** C `// Cells changed to occupied`

updateFree(F)
- $C \leftarrow \emptyset$ `// List of changed cells`
- **foreach** $\langle x, y \rangle \in F$ **do**
  - $state \leftarrow$ occupied(x, y)
  - $misses_{x,y} \leftarrow misses_{x,y} + 1$
  - **if** $state \wedge \neg$occupied(x, y) **then**
    - $C \leftarrow C + \langle x, y \rangle$
- **end**
- **return** C `// Cells changed to free`

5.1.3 Evaluation and Benchmarking

We seek to analyze the quality of the trajectories and maps generated by our SLAM proposal, and also its computational efficiency. For that purpose, we used a set of publicly available datasets provided by The Robotics Data Set Repository (Howard and Roy, 2003), and ran them through our algorithm. The datasets in question are the ACES Building, the Intel Research Lab, the MIT CSAIL Building and the Freiburg Indoor Building 079. For reference, all experiments were run in a MacBook Pro with an Intel Core i7 2.8GHz and 16GiB of RAM, and all data was used without pre-processing.

The outputs of the SLAM solution presented in this paper are influenced by its parameters. The most relevant parameters are: the accumulated translation ρ and rotational ϑ motions between each scan processing; the maximum Euclidean distance $\bar{\varepsilon}_{\max}$ of the dynamic likelihood field; and the strategy used for optimization, that may be the Gauss-Newton (GN) or the Levenberg-Marquardt (LM) method. To find the best outcome, we did a grid search over all parameters with $\rho \in \{0.01\text{m}, 0.02\text{m}, \ldots, 0.5\text{m}\}$, $\vartheta \in \{0.01\text{rad}, 0.02\text{rad}, \ldots, 0.5\text{rad}\}$, $\bar{\varepsilon}_{\max} \in \{0.5\text{m}, 1.0\text{m}, \ldots, 5.0\text{m}\}$ and strategy $\in GN|LM$. To compare strategies, the best outcomes for both GN and LM are kept and presented. The parameters that provide low translational an rotational errors in the benchmark for each dataset are shown in Table 5.1.

Table 5.1: Localization parameters that provide low translational and rotational errors in the experiments for SLAM benchmark and evaluation.

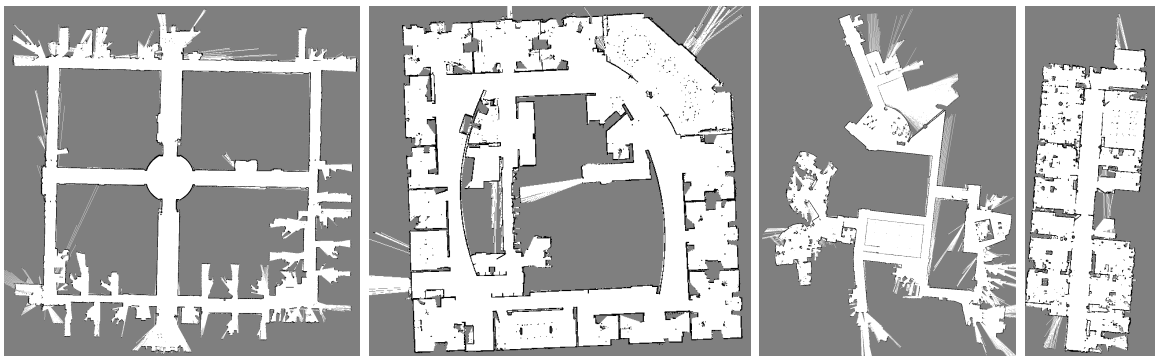
	Gauss-Newton				Levenberg-Marquardt		
	ρ (m)	ϑ (rad)	$\bar{\varepsilon}_{\max}$ (m)		ρ (m)	ϑ (rad)	$\bar{\varepsilon}_{\max}$ (m)
ACES	0.01	0.19	0.5	ACES	0.01	0.19	0.5
Intel	0.01	0.01	0.5	Intel	0.01	0.07	0.5
CSAIL	0.01	0.33	0.5	CSAIL	0.01	0.23	0.5
Fr079	0.01	0.03	0.5	FR079	0.01	0.01	0.5

There is a clear trend for the best ρ in all datasets for the two different optimization strategies. It is the same low value (i.e. $\rho = 0.01$) in all situations. This is not a completely unexpected value, as the employed local optimization works the best when the error is smaller. It is common in other SLAM solutions to have ρ in the approximate range $1.0 \leq \rho \leq 0.5$, but that is the result of high computation needs that require a higher interval time between scan processing. Another trend is the value of $\bar{\varepsilon}_{\max} = 0.5$, which is the parameter that most influences the overall execution time, a bigger value corresponding to higher execution times. In the grid search this value was not always the best one, but the difference between the trajectories obtained at $\bar{\varepsilon}_{\max} = 0.5$ with the ones at higher values were not significant enough to trade computational efficiency. Hence, for all datasets and strategies, $\bar{\varepsilon}_{\max} = 0.5$ is used.

Objective Benchmark

The quality of a map, usually interpreted by visual inspection (Figure 5.1), is correlated with the accuracy of the obtained trajectory. But this leads us to a subjective evaluation of the obtained results, and an apparent degraded map can actually preserve the correct topology of the environment and therefore represent the correct local metric structure of the environment (Kümmerle et al., 2009).

Kümmerle et al. (2009) proposes an objective benchmark to evaluate SLAM solutions that does not rely on a global appreciation of the resulting map. They introduced a set of relations $\{\delta_{i,j}\}$ that are based on local displacements between robot poses, defined by $\delta_{i,j} = x_j \ominus x_i$, instead of absolute poses. The benchmark metric is given by the mean error in translation



(a) ACES (b) Intel (c) CSAIL (d) Fr079

Figure 5.1: Constructed occupancy grid maps of the evaluated datasets using the proposed approach.

$\bar{\epsilon}_{\text{trans}}$ and mean error in rotation $\bar{\epsilon}_{\text{rot}}$ calculated from the trajectory relation $\delta_{i,j}$ and the ground truth relation $\delta^*_{i,j}$:

$$\epsilon(\delta) = \underbrace{\frac{1}{N}\sum_{i,j} \text{trans}(\delta_{i,j} \ominus \delta^*_{i,j})^2}_{\bar{\epsilon}_{\text{trans}}} + \underbrace{\frac{1}{N}\sum_{i,j} \text{rot}(\delta_{i,j} \ominus \delta^*_{i,j})^2}_{\bar{\epsilon}_{\text{rot}}} ,$$

where the functions *trans* and *rot* extract the translational and rotational parts of $(\delta_{i,j} \ominus \delta^*_{i,j})$, respectively. Another key point of this benchmark is the capability to compare different SLAM algorithms independently of the type of map representation they use, such as occupancy grids or feature maps. This allow us to compare our solution with other scan matching based solutions and also with different methods such as particle filters or Graph based optimization.

The evaluation of all obtained trajectories, using the proposed approach, with the described benchmark as well as the results reported by Kümmerle et al. (2009), Holz and Behnke (2010) and Hess et al. (2016), are summarized in Table 5.2 and Table 5.3. Our solution is compared against five different SLAM approaches. The first two approaches, ICP (Holz and Behnke, 2010) and Scan Matching (Kümmerle et al., 2009), are based on scan matching but differ on the matching strategy, i.e. the former does a scan-to-map matching while the latter does a scan-to-scan matching. The third approach is also based on scan matching but has an additional active loop closure process (Hess et al., 2016). The fourth is a RBPF approach (Grisetti et al., 2007a) while the last one is based on Graph mapping optimization (Grisetti et al., 2007b).

The mean error and corresponding standard deviation provides a global appreciation of the algorithm's computational performance. But it is also interesting to understand how it performed locally over time for the different datasets. By applying a rolling average $\bar{\mu}$ and rolling standard deviation $\bar{\sigma}$ to our errors, we smooth the information locally and at the same time we highlight long-term trends (Figure 5.2 and Figure 5.3).

Table 5.2: Benchmark quantitative results for the tested datasets on the **translation** error with the corresponding standard deviation. The best values are presented in bold.

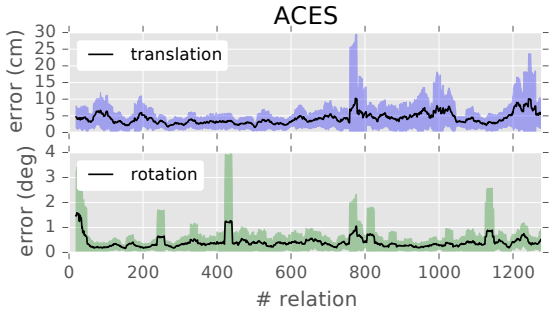
Trans. error (cm)	Proposed (GN)	Proposed (LM)	ICP	SM*	SMLC **	RBPF	GM***
ACES	4.2 ± 4.7	4.1 ± 3.9	6.0 ± 5.5	17 ± 61	$\mathbf{3.7 \pm 4.2}$	6.0 ± 4.9	4.4 ± 4.4
Intel	$\mathbf{2.0 \pm 1.9}$	2.1 ± 1.8	4.3 ± 5.8	22 ± 29	2.2 ± 2.3	7.0 ± 8.3	3.1 ± 2.6
CSAIL	$\mathbf{3.0 \pm 2.7}$	3.2 ± 3.1	4.3 ± 5.3	10 ± 32	3.1 ± 3.5	4.9 ± 4.9	0.4 ± 0.9
Fr079	$\mathbf{3.9 \pm 3.0}$	3.9 ± 3.2	5.7 ± 4.3	25 ± 42	4.5 ± 3.5	6.1 ± 4.5	5.6 ± 4.2

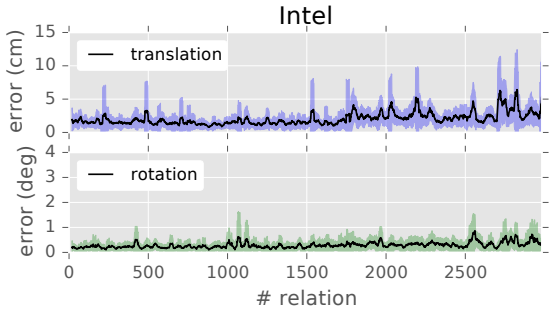
*Scan Matching **Scan Matching with Loop Closure ***Graph Mapping

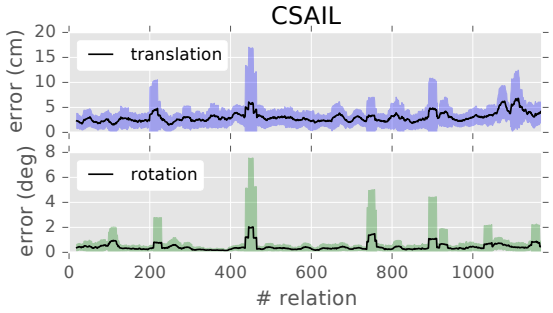
Table 5.3: Benchmark quantitative results for the tested datasets on the **rotational** error with the corresponding standard deviation. The best values are presented in bold.

Rot. error (deg)	Proposed (GN)	Proposed (LM)	ICP	SM*	SMLC **	RBPF	GM***
ACES	0.4 ± 0.6	0.4 ± 0.6	1.2 ± 1.6	1.2 ± 1.5	$\mathbf{0.3 \pm 0.4}$	1.2 ± 1.3	0.4 ± 0.4
Intel	$\mathbf{0.2 \pm 0.3}$	0.4 ± 1.1	1.5 ± 3.0	1.7 ± 4.8	0.4 ± 1.3	3.0 ± 5.3	1.3 ± 4.7
CSAIL	0.4 ± 1.0	0.5 ± 1.6	1.6 ± 2.6	1.4 ± 4.5	0.3 ± 0.3	0.6 ± 1.2	$\mathbf{0.1 \pm 0.1}$
Fr079	0.4 ± 0.5	0.4 ± 0.6	1.4 ± 1.7	1.7 ± 2.1	0.5 ± 0.7	0.6 ± 0.6	$\mathbf{0.6 \pm 0.6}$

*Scan Matching **Scan Matching with Loop Closure ***Graph Mapping







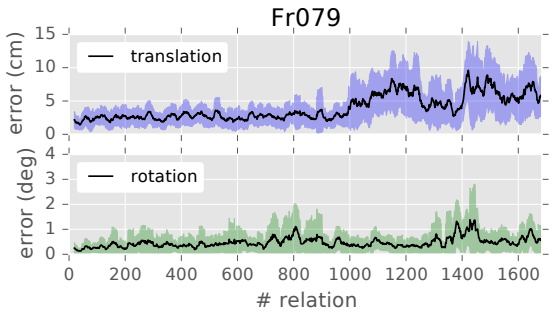
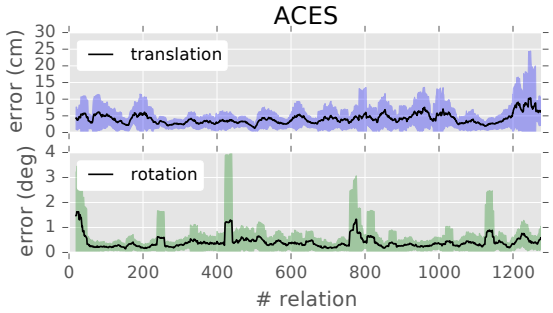
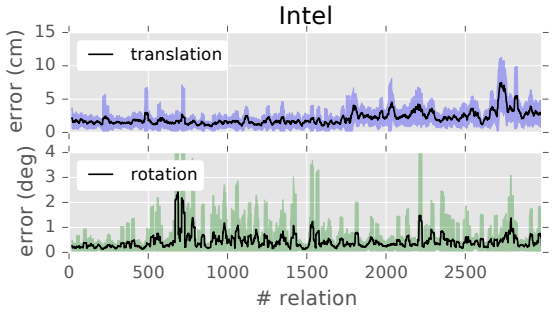
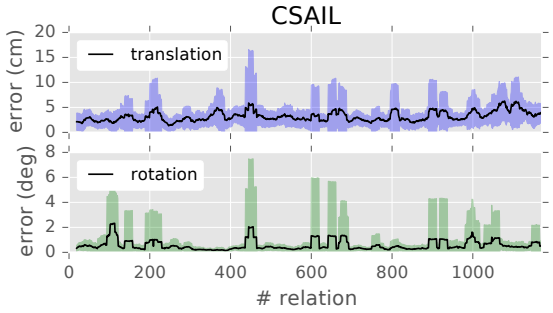


Figure 5.2: Error rolling mean $\mu \pm \theta$ for the tested datasets using the **Gauss-Newton** optimization strategy. The data window used in the rolling mean is equal to 20.







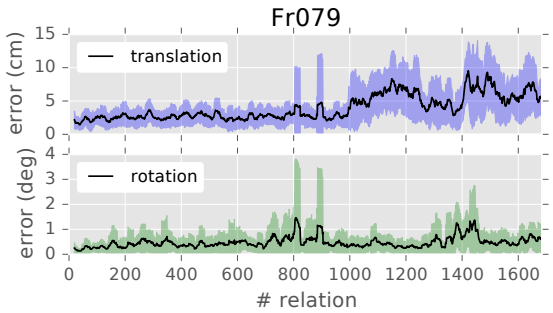
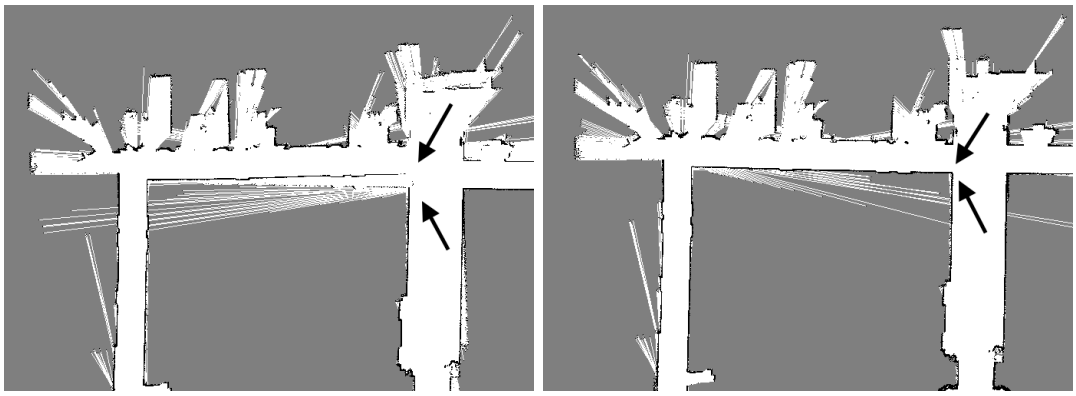


Figure 5.3: Error rolling mean $\mu \pm \theta$ for the tested datasets using the **Levenberg-Marquardt** optimization strategy. The data window used in the rolling mean is equal to 20.

For all datasets our solution is capable of constructing consistent environment models with mean errors that are comparable with well established methods, such as RBPF and Graph Mapping. But there is one exception, the ACES map constructed with Gauss-Newton has a local inconsistency on the top left corridor — Figure 5.4(a). Fortunately, this inconsistency disappears when using Levenberg-Marquardt — Figure 5.4(b). On the translation error, the



(a) (b)

Figure 5.4: Partial constructed occupancy grid maps from ACES. (a) Was built using Gauss-Newton and contains a local inconsistency. (b) With Levenberg-Marquardt the inconsistency disappears.

absolute mean values are mostly lower than the other solutions. On the rotational error, the best results are shared by our solution, a scan matching solution with loop closure and Graph Mapping. The obtained errors are very similar for both strategies, Gauss-Newton and Levenberg-Marquardt. However, because the difference in results is mostly residual, it is not possible to discern from the benchmark alone which strategy is clearly the best choice.

Computational Efficiency Analysis

We are also interested in analyzing the computational efficiency of our SLAM solution, more specifically, the mean processing time (or update time) between each scan s_{proc}, which encloses the optimization step and mapping. The latter includes the integration of the current scan into the occupancy grid map and the update of the Euclidean distance map used by the dynamic likelihood field.

When performing SLAM offline to build a model of the environment the time it takes to process the data is, in principle, of little relevance. However, there are several tasks that require the robot to construct its internal representation of the environment during operations, as for example, autonomous exploration of unknown space. In these scenarios time does matter. We refer to this as the capability to perform SLAM online, or real-time if the time it takes to process a scan is lower than the data acquisition time. Since the beginning, our SLAM solution was designed to be fast. By using the likelihood field as a measurement model we avoided explicit data association, usually a time consuming task, and allowed us to use least squares for pose estimation. Also, the dynamic likelihood field addresses the execution time degradation that arises from the dynamic nature of the incremental build of a map that is subsequently used for localization. A summary of the execution metrics is presented in Table 5.4.

Knowing that all datasets have a data rate of approximately 6Hz (a 166 milliseconds period) we can state that our SLAM algorithm can run in real-time. The difference between strategies is not significant, but from the detailed view of the execution times (Figure 5.5 and Figure 5.6) there is a visible increase of execution time during the optimization phase when Levenberg-Marquardt is used. The low number of iterations required from both strategies is an indication that the initial pose estimation is not far from the local minimum and/or the least squares method has good convergence.

Table 5.4: Mean execution times with min-max and mean number of iterations during optimizations.

Gauss-Newton

	$s_{\text{proc}}\ ms$	min-max	#iter
ACES	1.5 ± 0.5	$0.5 - 7.2$	6 ± 3
Intel	0.9 ± 0.2	$0.3 - 7.4$	4 ± 2
CSAIL	3.0 ± 1.4	$0.9 - 15.5$	7 ± 4
Fr079	1.9 ± 0.5	$0.6 - 8.3$	5 ± 2

Levenberg-Marquardt

	$s_{\text{proc}}\ ms$	min-max	#iter
ACES	1.8 ± 0.6	$0.6 - 10.0$	12 ± 5
Intel	1.0 ± 0.3	$0.4 - 12.9$	4 ± 2
CSAIL	3.8 ± 1.7	$1.1 - 14.8$	14 ± 6
FR079	2.3 ± 0.7	$0.9 - 11.7$	10 ± 4

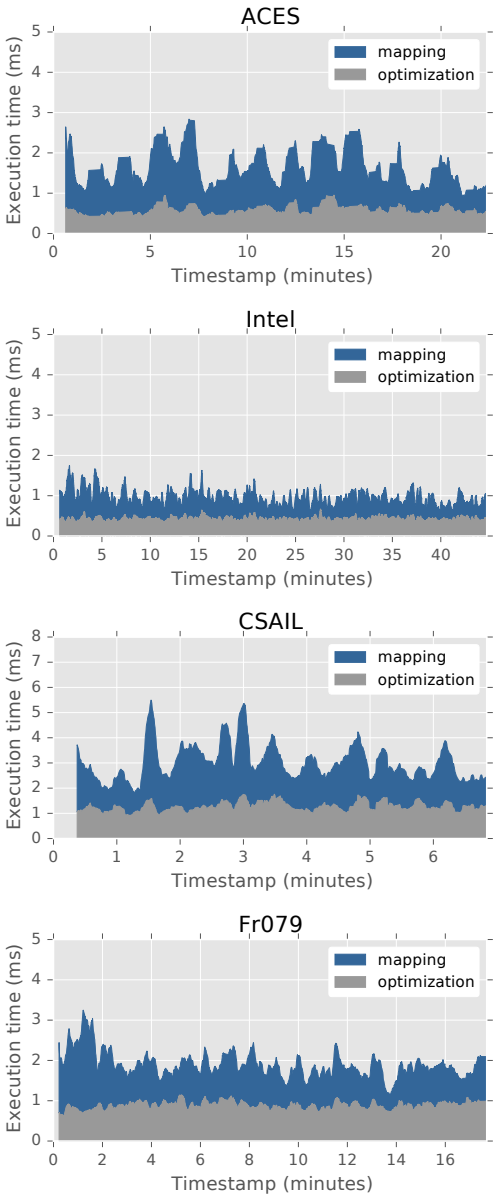


Figure 5.5: Rolling mean $\mu \pm \theta$ of stacked processing time (mapping + optimization) using **Gauss-Newton**. The data window is equal to 50.

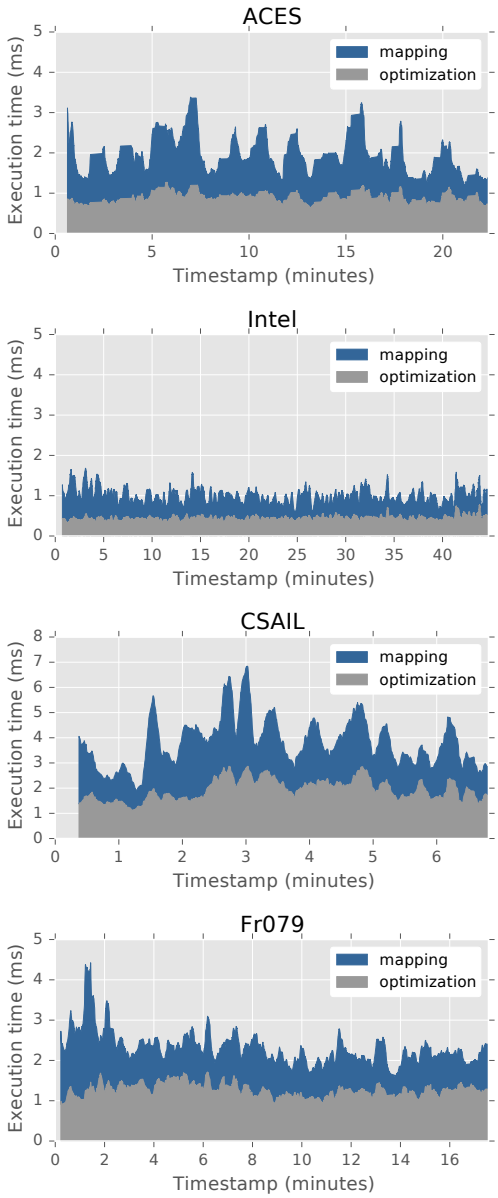


Figure 5.6: Rolling mean $\mu \pm \theta$ of stacked processing time (mapping + optimization) using **Levenberg-Marquardt** . The data window is equal to 50.

Space Efficiency Analysis

Thanks to the sparse-dense mapping framework, modeling the environment with an occupancy grid map does not have a significant impact in memory usage, even with the additional Euclidean distance grid map required by our localization algorithm. Nonetheless, by taking advantage of online data compression the memory usage can be further reduced (Figure 5.7).

The Euclidean distance grid map consumes at least four times (4x) more memory than the occupancy grid map. The former maintains a data structure for dynamic update that requires 16 bytes per grid cell while the latter requires 4 bytes to bookkeep the number of misses and hits (per grid cell). For our *online* solution this is not a problem because it does not have to scale beyond a single pair of grid maps. But for the foreseeable use of these grid maps in a particle filter SLAM solution they need to scale, thus the importance of the memory usage reduction provided by the online data compression.

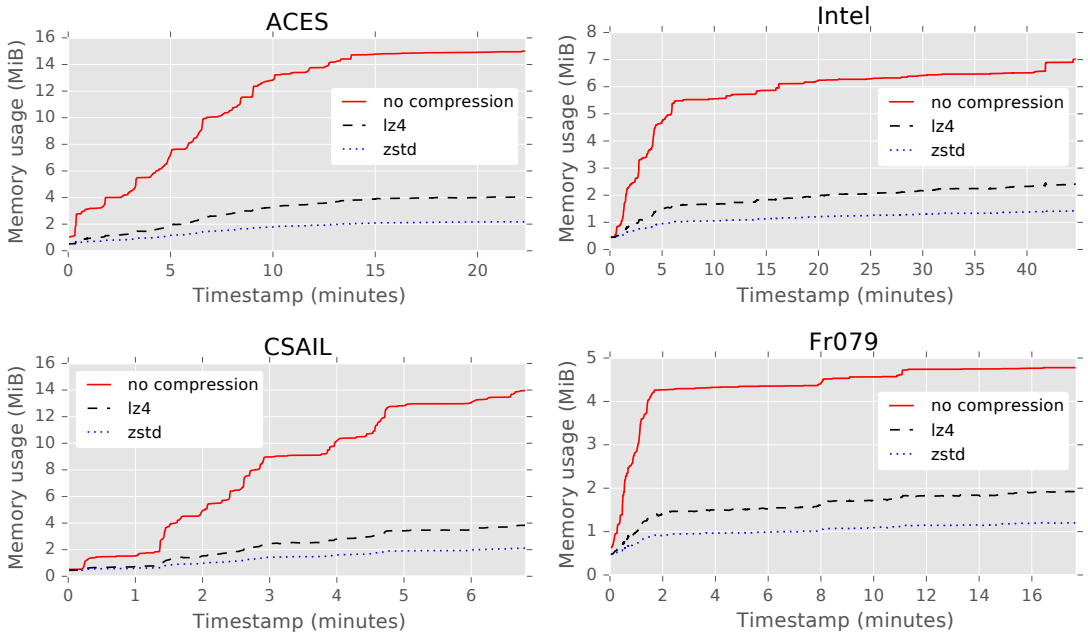


Figure 5.7: Memory usage during mapping for the considered datasets. All plots contain memory usage without compression and with compression using the lz4 and zstd data compressors. The compression mechanism is provided by the online data compression of the sparse-dense mapping framework.

5.2 Improving Rao-Backwellized Particle Filter SLAM with Scan Matching and Multi-Threading

The *full* SLAM problem defines SLAM as the posterior estimation of the entire path together with the map:

$$p(\mathbf{x}_{1:t}, m \mid z_{1:t}, u_{1:t}) \ , \tag{5.5}$$

where $\mathbf{x}_{1:t}$ is the potential trajectory laid down by the robot over time $[1, t]$ given the measurements $z_{1:t}$ and control data $u_{1:t}$ (usually odometry). The difference between *online* and *full* SLAM may seem subtle but its influence on the type of algorithms that can solve the *full* problem is noticeable (Thrun et al., 2005).

A way to solve the SLAM problem defined by (5.5) was presented by Murphy (1999). It introduces Rao-Backwellized particles filters (RBPF) as an efficient method to solve the *full* SLAM problem. The posterior over trajectories and maps is defined by the factorization

$$p(\mathbf{x}_{1:t}, m \mid z_{1:t}, u_{1:t}) = \underbrace{p(\mathbf{x}_{1:t} \mid z_{1:t}, u_{1:t})}_{\text{localization}} \underbrace{p(m \mid \mathbf{x}_{1:t}, z_{1:t})}_{\text{mapping}} . \tag{5.6}$$

Here, the mapping posterior $p(m \mid \mathbf{x}_{1:t}, z_{1:t})$ can be calculated efficiently by handling it as a problem of mapping with known poses. The localization posterior $p(\mathbf{x}_{1:t} \mid z_{1:t}, u_{1:t})$ can be estimated by a particle filter in which every particle has its own map. The map is built given the measurements $z_{1:t}$ and the trajectory $\mathbf{x}_{1:t}$ of the corresponding particle. The evolution of the robot's trajectory is defined by the robot's own motion, thus the proposal distribution is approximated by the probabilistic odometry motion model.

The most commonly used variant of particle filters is the sampling importance resampling (SIR) filter which approximates a probability distribution by a weighted set of N particles

$$\left\{\mathbf{x}^{[i]}, w^{[i]}\right\}_{i=1,\ldots,N} , \tag{5.7}$$

where $\mathbf{x}^{[i]}$ is a hypothetical pose (i.e. a sample of $\mathbf{x}$) and $w^{[i]}$ is the weight of that sample. A Rao-Blackwellized SIR filter, used for incremental mapping, integrates the available data (i.e. measurements and odometry) as they are available. This integration updates the particles (or samples) that represent the posterior of the robot's trajectory and map. The update procedure can be summarized in the following four steps:

1. **Sampling:** The first step of the filter is to generate the next set of particles $\{\mathbf{x}_t^{[i]}\}$ from the current set $\{\mathbf{x}_{t-1}^{[i]}\}$ by sampling from a proposal distribution π. A close form of the posterior is not usually available, therefore, a popular and simple choice for the proposal distribution is the motion model that represent the change of state over time (Dellaert et al., 1999; Montemerlo et al., 2002; Eliazar and Parr, 2003)

2. **Importance Weighting:** After the sampling process, an individual importance weight $w^{[i]}$ is calculated for each particle and is defined by

$$w_t^{[i]} = \frac{p(\mathbf{x}_{1:t}^{[i]} \mid z_{1:t}, u_{1:t})}{\pi(\mathbf{x}_{1:t}^{[i]} \mid z_{1:t}, u_{1:t})} . \tag{5.8}$$

   Note that the use of the importance weight $w^{[i]}$ accounts for the fact that the proposal distribution is not equal to the true distribution of the robot's change of states. This formulation requires the calculation of the weight for the complete trajectory when a new measurements is available, which is not efficient. According to Doucet et al. (2001) a recursive formulation of the importance weight can be obtained by imposing the following restriction to the proposal distribution π:

$$\pi(\mathbf{x}_{1:t} \mid z_{1:t}, u_{1:t}) = \pi(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, z_{1:t}, u_{1:t}) \cdot \pi(\mathbf{x}_{1:t-1} \mid z_{1:t-1}, u_{1:t-1}) . \tag{5.9}$$

   Now, expanding (5.9) into (5.8) the importance weight is computed as

$$w_t^{[i]} = \eta \cdot \frac{p(z_t \mid m_t^{[i]}, \mathbf{x}_t^{[i]})\, p(\mathbf{x}_t^{[i]} \mid \mathbf{x}_{t-1}^{[i]}, u_t)}{\pi(\mathbf{x}_t^{[i]} \mid \mathbf{x}_{1:t-1}^{[i]} z_{1:t}, u_{1:t})} \cdot w_{t-1}^{[i]} . \tag{5.10}$$

   where η is the normalization factor from the Bayes' rule and is the same for all particles.

3. **Resampling:** In this step, samples with low importance weight w are potentially replaced by other samples that hold a higher importance weight. This step is required because the number of particles to approximate the distribution is finite and allows the application of the filter in situations where proposal distribution is different from the true distribution. However, the resampling step can also delete good samples and lead to particle depletion (Van Der Merwe et al., 2000; Thrun et al., 2005). To prevent such effect it is necessary to define a criterion to when the resampling step occurs.

4. **Map Estimation:** The final step of the filter is to update the map $m_t^{[i]}$ of each particle using the corresponding pose $\mathbf{x}_t^{[i]}$ and current observations z_t. The map $m_t^{[i]}$, although built incrementally, is the reflection of the trajectory and history of measurements according to $p(\mathbf{x}_{1:t}^{[i]} \mid z_{1:t}, u_{1:t})$.

The efficiency of the particle filter is grounded on the proposal distribution and the reduction of the particle depletion problem. The work of Grisetti et al. (2005) addresses these two issues for grid-based RBPF SLAM by presenting an improved proposal distribution that samples around a local maximum found by a scan matching process and an adaptive resampling technique based on the so-called effective number of particles (Liu, 1996).

Our proposal for an efficient grid-based RBPF SLAM is based on similar ideas: an improved proposal distribution supported by scan matching and adaptive resampling controlled by the effective number of particles. Additionally, the particle filter update is speeded by a pool of multithreaded workers that leverages the independence between particles.

5.2.1 A Proposal Distribution with Scan Matching Refinement

The sampling step of the particle filter needs to draw samples from a proposal distribution π that is close to the true distibution $p(\mathbf{x}_{1:t} \mid z_{1:t}, u_{1:t})$ as much as possible, nonetheless the choice being arbitrary. As previously discussed, the motion model is a common choice for the proposal distribution. Because it models the motion of the robot, i.e the change of state of the robot, and is easy to calculate, it is a natural candidate for the proposal distribution:

$$\begin{aligned} \mathbf{x}_t^{[i]} &\sim p(\mathbf{x}_t^{[i]} \mid \mathbf{x}_{t-1}^{[i]}, u_t) \\ &\sim \mathcal{N}(\mathbf{x}_{t-1} \oplus u_t, \Sigma) : \Sigma \propto u_t \end{aligned} \tag{5.11}$$

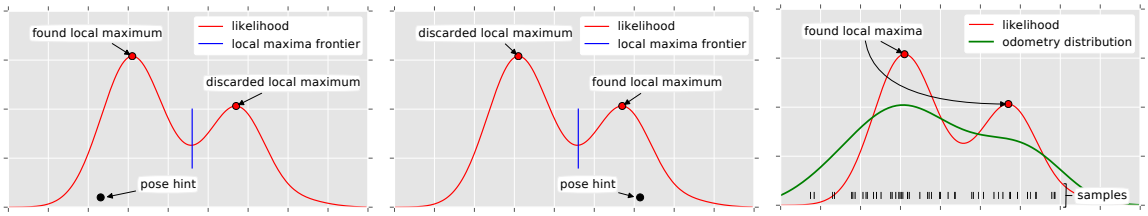
where $\mathcal{N}$ is a multivariate normal distribution with covariance Σ proportional to the magnitude of the motion control data u_t. By using the motion model as the proposal distribution π the importance weight is given by the measurement model $p(z_t \mid m, \mathbf{x}_t)$, as it can be shown by replacing π in (5.10) with the motion model $p(\mathbf{x}_t \mid \mathbf{x}_{t-t}, u_t)$

$$\begin{aligned} w_t^{[i]} &= \eta \cdot \frac{p(z_t \mid m_t^{[i]}, \mathbf{x}_t^{[i]})\, p(\mathbf{x}_t^{[i]} \mid \mathbf{x}_{t-1}^{[i]}, u_t)}{p(\mathbf{x}_t^{[i]} \mid \mathbf{x}_{t-1}^{[i]}, u_t)} \cdot w_{t-1}^{[i]} \\ &\propto p(z_t \mid m_t^{[i]}, \mathbf{x}_t^{[i]}) \cdot w_{t-1}^{[i]} \,. \end{aligned} \tag{5.12}$$

In mobile robots the perception of motion is captured by odometers that are susceptible to errors induced by sliding and/or skewing, thus to approximate the true distribution a high number of particles is required. Our approach to reduce the errors introduced by the odometry is to use a scan matching process to correct it, a process similar to Hahnel et al. (2003) that

uses scan matching to correct the odometry before sampling. But such approach requires the modeling of the scan matching process error for sampling that depends on several factors (e.g. sensor accuracy). To avoid this necessity our proposal inverts the order: first we sample from the motion model and then we refine the pose of the sample using scan matching. This way we avoid the necessity of integrating the scan matching uncertainty into the sampling process.

One of the problems of scan matching algorithms is their greedy nature of following the closest local maximum, discarding any possible local maximum with higher likelihood. But, by sampling from the motion model there is a higher chance of covering more areas of high likelihood with the current sample set (Figure 5.8). With this method, samples under the same likelihood area will converge to the same maximum, therefore, the required number of samples to represent the localization distribution can be drastically reduced. The downside of this approach is the eventual convergence to a highly peaked distribution with low sample diversity that results from the resampling step of the filter. To prevent this effect we must guarantee a sufficient diversity before generating the next sample set. This can be achieved by using an adaptive resampling technique (Grisetti et al., 2005).



(a) (b) (c)

Figure 5.8: Handling the greedy scan matching local search by sampling from motion. In scenario (a), the initial pose for scan matching optimization is under the area of the highest local maximum, so highest local maximum is found. However, if the initial pose fall under the area of a lower local maximum, like in scenario (b), the highest local maximum is discarded. By using multiple samples drawn form motion, e.g. scenario (c), both local maxima are found.

5.2.2 Adaptive Resampling

The resampling step is necessary since the number of particles to approximate the posterior distribution is finite. Particles with higher importance weight are more likely to generate new particles for the next generation, which will result in the elimination of particles with low importance weight. Particles with lower importance weight are not necessarily bad samples and their elimination can result in an impoverishment of the sample set. To avoid the depletion of good samples it is important to define a criterion for deciding when to perform the resampling process. The effective sample size metric, defined as N_{eff}, was introduced by Liu (1996) and it estimates how well the current particle set represents the target posterior. Its application in grid-based RBPF SLAM to control the resampling process was first proposed by Grisetti et al. (2005) with good results, and in some scenarios it was essential. The value of this metric is calculated according to the following formulation (Doucet et al., 2001):

$$N_{\text{eff}} = \frac{1}{\sum_{i=1}^{N} \left(w_t^{[i]} \right)^2} \qquad \text{with} \qquad \sum_{i=1}^{N} w_t^{[i]} = 1 \tag{5.13}$$

If the samples were to be drawn from the target (or true) distribution, their importance weight would be the same, which follows the importance-sampling principle. But a bad approximation of the target distribution results in a sample set with importance weights of high variance. The value of N_{eff} can also be seen as a measure of the dispersion of the importance weights, thus it is a valid metric to evaluate the quality of the posterior approximation by the current particle set. Following Grisetti et al. (2005), the resampling step is executed whenever N_{eff} drops below the threshold of $N/2$, with N being the number of particles.

When equipped with accurate sensors, such as laser ranger finders, the importance weight of each particle can differ significantly due to the peaked likelihood function of the measurement model even when close to an area of interest. As consequence, a single sample can dominate the importance weight of the complete set, e.g. Figure 5.9(a), which will not prevent particle depletion. This can be handled by smoothing the likelihood function to avoid low importance weight of particles that are close to the relevant area:

$$\bar{w}_t^{[i]} = w^{\frac{1}{gN}} \quad \text{with} \quad w = w_t^{[i]} \propto p(z_t \mid m_t^{[i]}, \mathbf{x}_t^{[i]}) \cdot w_{t-1}^{[i]} \tag{5.14}$$

where $\bar{w}_t^{[i]}$ is the smoothed importance weight and g is the gain value that smooths the likelihood (see Figure 5.9).

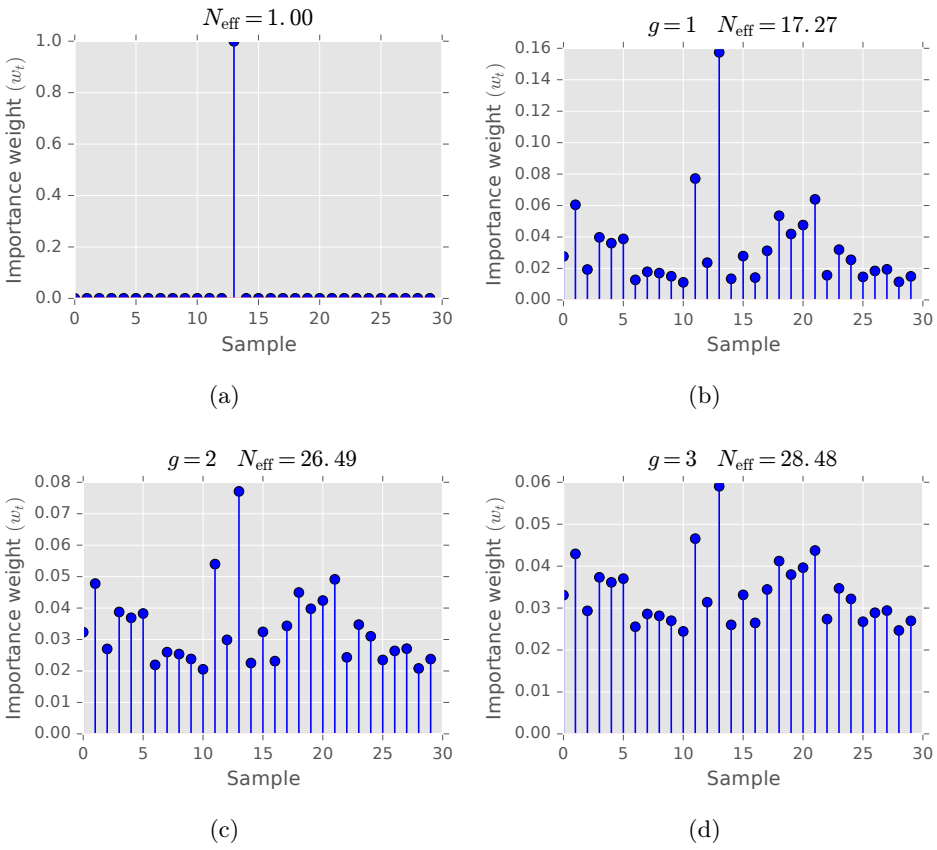


Figure 5.9: Handling highly peak importance weights with likelihood smoothing (N=30). In (a), likelihood smoothing is not used and a single sample dominates the importance weight of the set. In (b)(c)(d), likelihood smoothing is used and as the gain g value increases the variance of the importance weights decreases.

5.2.3 Multithreaded RBPF SLAM

The Rao-Blackwellized particle filter assumes independence between particles, therefore, at each update step every particle has an isolated scan matching process and consequent map update. The non existence of data flow between particles should allow their update to be parallelized. However, the particles are not completely isolated. To reduce the space complexity of the filter, partial content of the particle's map is shared among its siblings. This detail introduces additional concurrency to the parallel update of the particle set.

The scan matching process performs read-only operations to the map's content, so it is not a problem for simultaneous access and can be parallelized without much effort. This is shown by Gouveia et al. (2014) by parallelizing the GMapping scan matching *for-loop* with a single OpenMP[2] compiler directive. By parallelizing the scan matching process, the filter update already gains noticeable speedup. Unfortunately, the map update process of GMapping can not be parallelized because it fails when read-write concurrency is at place, loosing the opportunity for further speedup. Unlike the current state-of-the-art, our RBPF SLAM proposal seeks to take full advantage of the ubiquitous existence of multi-core systems with several execution threads by parallelizing both scan matching and map update processes with concurrency support.

The parallelization used in our solution is based on the fixed size *Thread Pool* model (see Figure 5.10). Instead of launching a new thread for each task, a fixed number of threads (named workers) are pre-allocated and have the capability to receive new tasks and execute them. Their status and state are monitored and controlled by a *pool* that is responsible for assigning the asynchronous execution of requested tasks. This model prevents the continuous creation and destruction of threads that introduces an additional execution overhead and implementation complexity. Instead, the user can push all tasks to the queue and then wait for their completion. The number of pre-allocated thread workers is user-defined, which delegates to the user the adequate amount of computational resources to be allocated.

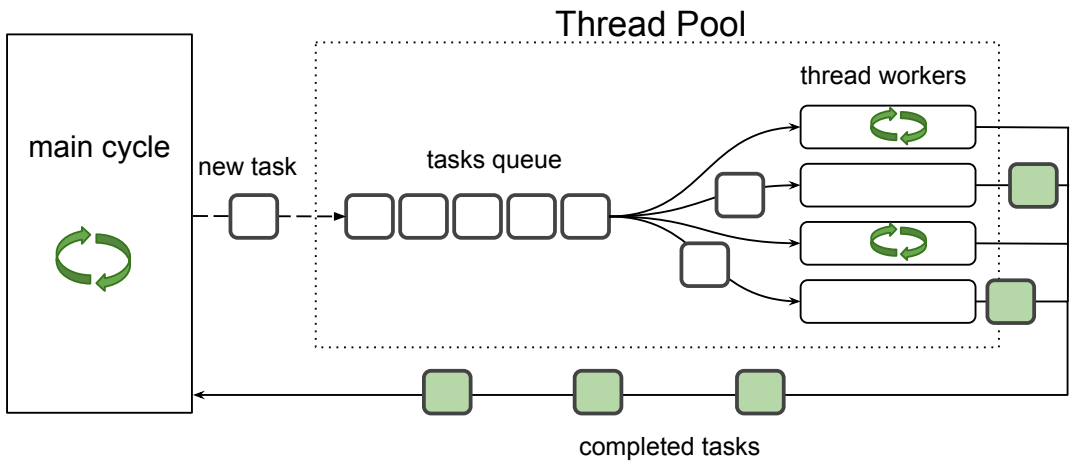


Figure 5.10: Diagram of an asynchronous thread pool. A thread pool has a queue of task that are assigned to thread workers that execute the tasks. Tasks are only assigned to idle thread workers. If all workers are busy the pool will wait. Once a task is completed its completion is reported to the caller. The number of thread workers is fixed during operations but the initial amount can be defined by the user.

[2] `http://www.openmp.org`

The integration of the proposal distribution with scan matching refinement and adaptive resampling enhanced by multithreading is summarized in algorithm 18. Each time a new measurement pair $\langle z_t, u_t \rangle$ is obtained, the particle filter is updated with the following steps:

1. The robot's pose $\mathbf{x}_t^{[i]}$ of particle i is sampled from a normal distribution with mean obtained from the previous pose $\mathbf{x}_t^{[i]}$ and odometry measurement u_t with covariance proportional to the magnitude of u_t. Then, the scan matching pose refinement procedure using $\mathbf{x}_t^{[i]}$ as initial hint and the previous map $m_t^{[i]}$ is added to the *thread pool* for asynchronous execution.
2. Wait until all scan matching tasks are completed.
3. Now that all particle poses are refined, we update their respective smoothed importance weight $w_t^{[i]}$. The map update process is then added to the *thread pool* for asynchronous execution. At this step we can already add a new sample to the sample set with the refined pose, the updated importance weight and a reference to particle's map $\langle \hat{\mathbf{x}}_t, w_t^{[i]}, m_t^{[i]} \rangle$.
4. Wait until all map update tasks are completed.
5. Normalize all importance weights and calculate the effective number of samples N_{eff}.
6. Do resampling if the number of effective sample size N_{eff} if lower than $N/2$.

Algorithm 18: Improved Multithreaded Rao-Blackwellized Particle Filter SLAM.

updateFilter$(z_t, u_t, g, \mathcal{S}_{t-1}, \mathcal{P})$

- $\mathcal{S}_t = \{\}$ `// The new sample set`
- **forall** $\langle \mathbf{x}_{t-1}^{[i]}, w_{t-1}^{[i]}, m_{t-1}^{[i]} \rangle \in \mathcal{S}_{t-1}$ **do**
  - $\mathbf{x}_t^{[i]} \sim \mathcal{N}(\mathbf{x}_{t-1} \oplus u_t, \Sigma) : \Sigma \propto u_t$ `// Sample from motion`
  - $\mathcal{P} \leftarrow \left\{ \hat{\mathbf{x}}_t^{[i]} = \underset{\mathbf{x}_t^{[i]}}{\text{argmax}}\, p(z_t | \mathbf{x}_t^{[i]}, m_{t-1}^{[i]}) \right\}$ `// Enqueue scan matching pose refinement`
- **end**
- waitForAllTasks$(\mathcal{P})$ `// Block until all tasks are completed`
- **forall** $\hat{\mathbf{x}}_t^{[i]}$ **do**
  - $w_t^{[i]} = p(z_t \mid m_{t-1}^{[i]}, \hat{\mathbf{x}}_t^{[i]}) \cdot w_{t-1}^{[i]}$ `// Calculate the sample importance weight`
  - $w_t^{[i]} = \left(w_t^{[i]} \right)^{\frac{1}{gN}}$ `// Smooth the sample importance weight`
  - $\mathcal{P} \leftarrow \left\{ m_t^{[i]} = \text{integrateMeasurement}(\hat{\mathbf{x}}_t^{[i]}, m_{t-1}^{[i]}, z_t) \right\}$ `// Enqueue grid maps update`
  - $\mathcal{S} = \mathcal{S} \cup \langle \hat{\mathbf{x}}_t, w_t^{[i]}, m_t^{[i]} \rangle$ `// Update sample set`
- **end**
- waitForAllTasks$(\mathcal{P})$ `// Block until all tasks are completed`
- **forall** $w_t^{[i]}$ **do** $\hat{w}_t^{[i]} = w_t^{[i]} / \sum_j^N w_t^{[j]}$ `// Normalize the sample importance weight`
- $N_{\text{eff}} = 1 / \sum_i^N (\hat{w}_t^{[i]})^2$ `// Calculate the effective sample size`
- **if** $N_{\text{eff}} < N/2$ **then** $\mathcal{S}_t = \text{resample}(\mathcal{S}_t)$
- **return** $\mathcal{S}_t$

The core components of the proposed grid-based RBPF SLAM are the scan matching procedure and the data structure that supports the volumetric grid. These are considered core components because, in theory, they can be implemented by any solution, that is, the filter is not bound to a specific scan matching algorithm or volumetric data structure. Nonetheless, the efficiency of the SLAM solution is the reflection of the choices for the core components.

The scan matching solution used in our RBPF SLAM is the same used for localization with a likelihood field, proposed in this thesis in Chapter 3, with the necessary adaptations for *online* SLAM, also proposed in this thesis in Section 5.1. The solution has proven to be an accurate and computationally efficient scan matching algorithm. Its computational efficiency is a desirable feature, specially in the context of a particle filter where each particle runs its own scan matching procedure. The downside of this scan matching solution is the requirement of a second volumetric grid that holds the Euclidean distance map. If not handled properly, it can introduce a significant space overhead in the exchange for lower execution times.

The volumetric grid data structure that is used to support the occupancy map and Euclidean distance map is the Sparse-Dense framework proposed in this thesis for efficient mapping (see Chapter 4). It provides a transparent management of space (e.g. automatic growth) with additional features: implicit data sharing and online data compression. The implicit data sharing provides an efficient method to share data between particles that are siblings (i.e. generated from the same parent particle), thus reducing space complexity. It also support multithreading by means of a thread-safe CoW mechanism that guarantees data consistency. The multithreading RBPF SLAM is only possible due to this mechanism. The online data compression is a feature that allows further reduction of space complexity without compromising executions times. This feature is used to reduce the amount of memory allocated by the Euclidean distance map and occupancy map. Additionally, the procedure used to update the maps of each particle is the same used in our *online* SLAM solution, see Subsec 5.1.2.

If the particle filter is reduced to a single particle, the SLAM solution is conceptually no different from our *online* solution. It executes the same scan matching method for localization and the same incremental mapping for capturing the geometric shape of the environment. Every particle in the filter is independent, therefore, when multiple particles are used, the RBPF SLAM solution has in practice a running *online* SLAM process in each particle.

5.2.4 Experiments and Analysis

To validate our RBPF SLAM solution we used a variety of indoor environments that demonstrate its efficiency and accuracy. The environments that are part of the experiments are the same used in the evaluation and benchmark of our *online* SLAM (see section 5.1.3) but with the addition of an environment (also part of the The Robotics Data Set Repository) that could not be solved by the *online* solution. This extra environment, named MIT Killian Court, has several long loops, contains nested loops and covers a large area (Figure 5.11), which makes it a challenging problem to be solved by a Rao-Blackwellized mapper due to particle depletion (Grisetti et al., 2005).

The objective of this section is to evaluate and compare the results of our RBPF SLAM solution with the current state-of-the-art, GMapping. This includes the capacity to generate topologically correct maps and to be computational and space management efficient. The benchmarking of the quality of the obtained trajectories is not considered, however, because part of our solution is based on an *online* SLAM with good benchmarks scores, which can potentially be transferred to the RBPF SLAM.

Mapping Results

In total, five datasets representing different environments were use in the experiments: ACES Building, Intel Research Lab, MIT CSAIL Building, Freiburg Indoor Building 079 and MIT Killian Court. For all of them, our RBPF SLAM solution was capable of creating a topologically correct map with good quality. For most datasets the filter is only updated after accumulating 0.5m of movement or 25° of rotation. The exception is the Fr09 environment for which an accumulated movement of 0.15m was necessary to handle its erratic odometry.

The maps of the first four environments are similar to the ones obtained by the *online* SLAM solution but in general they have a higher quality, specially on locations where a loop closure exist, e.g. Figure 5.12. This quality was achieved using no more than 15 particles. Excluding the MIT Killian Court from the dataset pool, it is possible to obtain a topologically correct map in 80% of the time using 5 particles.

The MIT Killian Court proved to be a challenging dataset. Due to its long and nested loops, particle depletion happens frequently. In this experiment, the adaptive resampling with smoothed likelihood and a higher number of particles were essential to obtain a topologically correct map. Nonetheless, our filter requires 50 particles to obtain a map without artificial double walls while GMapping requires 80 particles for the same result, an increase of 60%.

Computational Efficiency Analysis

Having a computationally efficient mapper has several advantages, for example: it frees computational resources for others tasks; it makes the mapping cycle until the ideal parameters are found faster; and it opens the possibility for a real-time grid-based RBPF SLAM

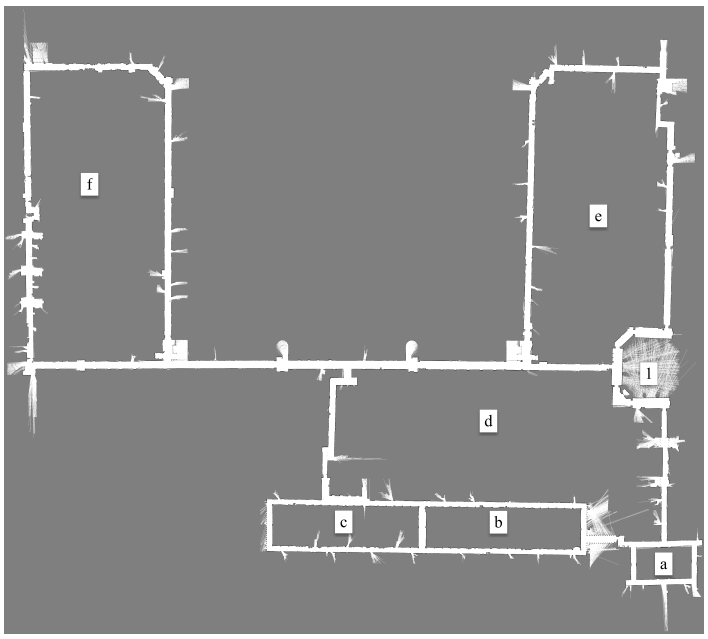


Figure 5.11: MIT Killian Court map. The robot starts and finishes the data capture around the point labeled by *1*. The point *a* is the first loop that is visited. Then, it move to *b*, *c* and returns to *a* and *1* before visiting the loops *e* and *f*. The environment has a size of $248m \times 222m$ and its map was generated with 60 particles.

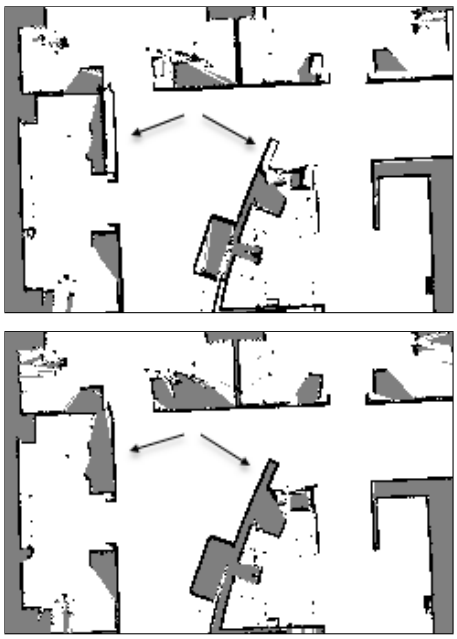


Figure 5.12: Example of map quality improvement on a loop closure location. The top map was obtained with the *online* SLAM solution and the bottom map was obtained with the RBPF SLAM solution.

solution. In this section, we present the total execution times for each dataset as well as their mean execution times for our approach and GMapping. Their comparison is based on the single-thread implementation of both approaches, afterwards we will present and analyse the advantages of our multithread version. The number of particles used for mapping is 30 for all datasets except for the MIT Killian Count, for witch we used 60 particles. For reference, all experiments were run in a HP ProLiant with two (2x) Intel Xeon 2.5GHz and 32GiB of RAM for a total availability of 12 cores and 24 execution threads. This provides enough computational and memory resources to evaluate our multithreading RBPF SLAM solution.

The obtained single-threaded execution times are shown in Figure 5.13, which contain a comparison between our solution and GMapping with the corresponding speedup. The difference in execution times is significant. It provides a speedup to the dataset processing that ranges from 6 to 25 times faster. The speedup is provided by our core components, the scan matching and the space management data structure that have good computational efficiency. In both solutions, the total execution time to process the datasets is always bellow the data time span. However, it does not assure that the solution is running in real-time. To evaluate such property, the mean update time is a better metric. A summary of single-threaded mean update times is presented in Figure 5.14, witch contains a comparison between our solution and GMapping and the corresponding speedup. Notably, our solution provides a considerable speedup over GMapping. Considering that all datasets have a measurement period no greater than 166 milliseconds (i.e. a 6Hz frequency) a mean update time bellow this threshold is a strong indicator that the solution is capable of operating in real-time, even in the worst case where an update occurs at every single measurement. That is the case of our solution, where the single-threaded mean update time is below the threshold for all datasets.

In practice, an update only occurs after the robot accumulates a certain amount of motion, which binds the SLAM update to the velocity of the robot. For example, a robot traveling at $1m/s$ with an accumulated $0.5m$ of movement before updating, provides to the SLAM system 0.5 seconds to update. This can be generalized with the following formulation:

$$t = \max\left(\frac{a}{v}, T\right) \implies v = \frac{a}{t} \tag{5.15}$$

where a is the accumulated movement, v is the velocity of the robot, T is the measurement period and t is the available time for update lower bounded by T. By replacing t with an estimate of the mean update time it is possible to obtain a robot velocity for witch we

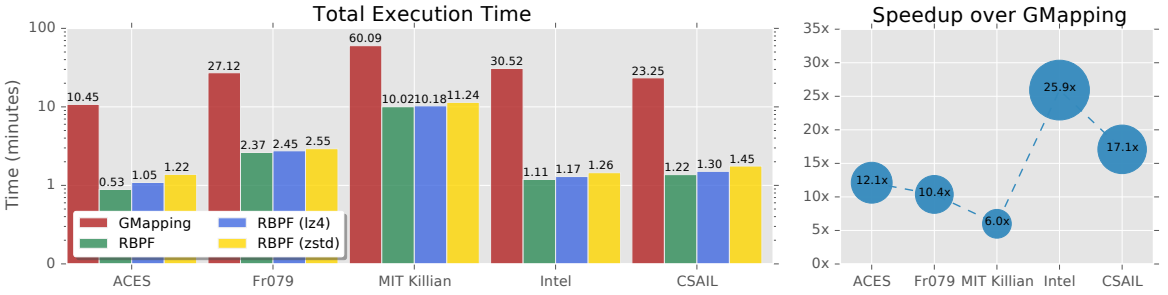


Figure 5.13: Comparison of single-threaded total execution times between our solution and GMapping. Note that the time axis on the left graphic is on a logarithmic scale and our solution is labeled as RBPF. The total execution graphic also contains our solution with online data compression activated using two different compressors. The speedup graphic, on the right, only considers our solution without compression.

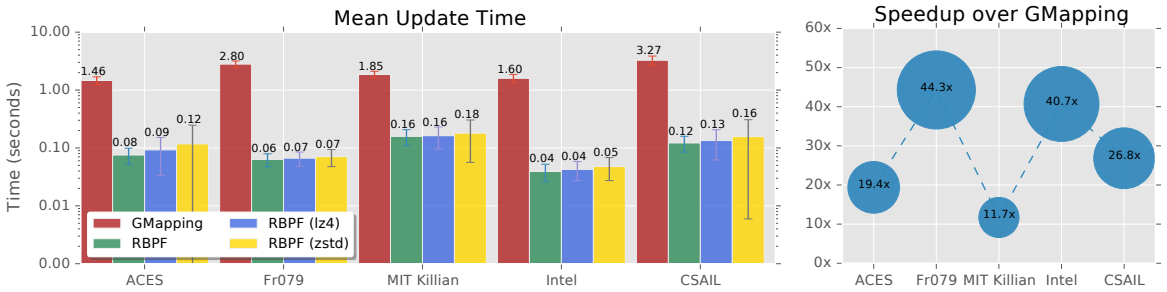


Figure 5.14: Comparison of single-threaded mean execution times between our solution and GMapping. Note that the time axis on the left graphic is on a logarithmic scale and our solution is labeled as RBPF. The mean update time graphic also contains our solution with online data compression activated using two different compressors. The speedup graphic, on the right, only considers our solution without compression.

can operate in real-time. For the dataset with the highest mean update time, GMapping would required the robot to move at $0.15m/s$, which depending on the application can be an acceptable velocity or not. On the other hand, our solution is not restricted by the robots velocity because its mean update time is below the measurement period.

It is already established that our solution has a good computational efficiency. But this is only considering its single-threaded version and our solution was design with multithread support to further improve its computational efficiency. The obtained results are shown in Figure 5.15 and they show that multithreading is a valuable tool to increase computational efficiency. When compared with GMapping the difference in computational efficiency just grows wider. The speedup curve, as the number of threads increases, grows until a saturation point is achieved where more threads can, in fact, be a bottleneck. This is shown in the speedup decrease in the transition from 16 to 24 threads. Note that for a different number of particles the saturation point can also be different.

The speedup values of the MIT Killian Court dataset stands from the other datasets, as they are higher. This happens because it uses a higher number of particles than the other datasets and it was observed that multithreading speedup favors higher number of particles, i.e. the higher the number of particles the higher the speedup value provided by multithreading.

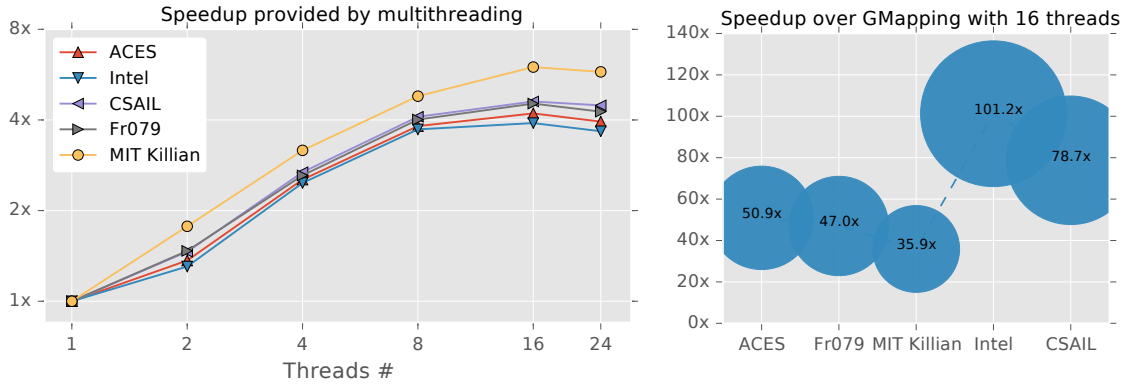


Figure 5.15: Total execution time speedup provided by multithreading per dataset. The left graphic shows the speedup that is obtained each time the number of threads is increased. The right graphic shows the speedup of our solution over GMapping when 16 threads are used.

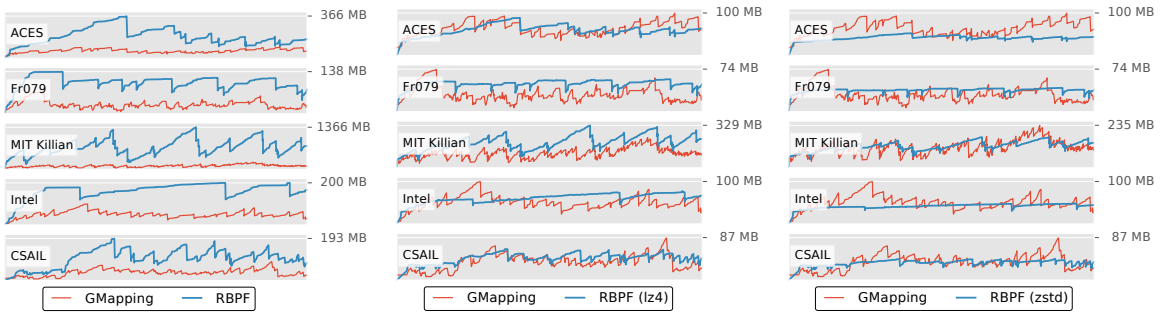
Space Efficiency Analysis

A good space (or memory) management is essential for any grid-based RBPF SLAM solution. For sufficiently large environment with a reasonable number of particles a naive approach can require an unmanageable amount of memory. For space management, GMapping follows a strategy similar to Eliazar and Parr (2003) where grid cells in particles that derived from a common parent particle are shared among its siblings until they are changed. This provides a manageable amount of memory even for a reasonable number of particles. The caveat is that is does not support multithreading. Our solution uses of the sparse-dense framework for data management that helps reducing the amount of memory usage. It has the same principle of data sharing between particles but it manages blocks of several grids cells instead of a single one and supports multithreading. The maximum memory usage reported by our solution and GMapping are shown in Table 5.5 and the evolution over time in Figure 5.16.

Table 5.5: Maximum memory usage reported by GMapping and our solution per dataset. Memory usage values with lz4 and zstd compression are also reported.

Dataset	GMapping	RBPF	RBPF (lz4)	RBPF (zstd)
ACES	99.70 MiB	365.19 MiB	88.93 MiB	52.26 MiB
Fr079	73.71 MiB	137.24 MiB	56.42 MiB	41.40 MiB
MIT Killian	234.21 MiB	1365.38 MiB	328.26 MiB	170.05 MiB
Intel	99.24 MiB	199.79 MiB	77.40 MiB	48.78 MiB
CSAIL	86.39 MiB	192.51 MiB	63.49 MiB	43.67 MiB

Without data compression, our solution has a space efficiency that is noticeable worst than GMapping. The computational efficiency of our solution requires a second grid map as trade-off. Unavoidably, it increases memory consumption. As already discussed, this second grid uses at least four time (4x) more memory than the occupancy grid and it can grow considerably as the number of particles increases. It is for situations like this that the online data compression was developed. With the addition of a negligible computational overhead, our space efficiency is greatly improved, reaching a level similar or even better than GMapping.



(a) (b) (c)

Figure 5.16: Memory usage over time of our solution compared with GMapping memory usage. In (a), the reported memory usage of our solution does not utilize online memory compression. In (b) and (c), online memory compression with lz4 and zstd, respectively, is used. Without online data compression our space efficiency is noticeable worst than GMapping, but with compression the efficiency is comparable if not better most of the times.

5.3 Conclusion

This chapter presents the development of two SLAM solutions, one addressing the *online* SLAM problem and the other the *full* SLAM problem. The *online* SLAM solution is based on a fast and accurate scan matching algorithm, developed in this thesis, that is adapted to the incremental map building nature of simultaneous localization and mapping. The mapping is supported by the sparse-dense framework, also developed in this thesis, that provides transparent (and automatic) map size growth and efficient space management. The end result is an accurate, light-weight and noticeable fast *online* SLAM solution with benchmark results that supports our claims.

The *full* SLAM solution is based on the insights of GMapping, a state-of-the-art grid-based RBPF SLAM introduced by Grisetti et al. (2005). Our solution contains an improved proposal distribution with samples drawn from motion and refined with scan matching, the same scan matching used in our *online* SLAM. The mapping procedure has the same origin, the *online* SLAM. Adaptive resampling with smoothed likelihood is used to avoid particle depletion in environments with large and nested loops. To further improve the computational efficiency, parallel scan matching and concurrent map update procedures are used. This is handled by multithreading with a *Thread Pool* model. The end result is RBFP SLAM solution capable of producing maps of good quality at a pace that goes from 6 to 25 times faster. Using multithreading further widens the computationally efficiency gap, reaching speedups such as 100 times faster. In terms of space efficiency, when data compression is used, it is in the same level of efficiency as GMapping. Considering the obtained results, our RBPF SLAM solution is a viable alternative to GMapping.

The remaining question is which SLAM solution to use. Both provide accurate maps with high computational and space efficiency. However, the *full* SLAM solution is capable of handling the same environments as the *online* SLAM solution and more. On the other hand, the latter is even more computationally and space efficient than the former. The final answer lies in the environment to map, the available resources and choice of the user. The *online* SLAM solution is a good fit for environments such as households or office spaces and because of its low resources requirements it is ideal for real-time domestics mobile robotic applications. Our RBPF SLAM is an "all-around" solution, capable of handling simple and challenging environments with good scalability. As an example, when more resources are available, such as thread count, offline mapping of a challenging large environment can leverage the multithreading support of our solution to reduce the time it takes to find a solution.

Chapter 6

Conclusion

The purpose of this thesis is to contribute to the state-of-the-art of localization and mapping by providing improved algorithms and tools that offer better accuracy and efficiency. Four contributions are proposed that addressed the problems of localization and mapping and their simultaneous resolve (i.e. SLAM). The first contribution focuses on the problem of mobile robot localization and the second on the problem of space management in grid mapping. The third and fourth contributions address the problems of *online* and *full* SLAM, respectively.

The problem of mobile robot localization is approached using scan matching with improved optimization and uncertainty handling. It uses a likelihood field as measurement model that is in part responsible for its reduced computational complexity. The optimization on a manifold showed to have fast convergence to the local minimum, which also contributed to the low execution times. The final result is a localization algorithm that offers a viable alternative to the current state-of-the-art with a high computational efficiency (Pedrosa et al., 2017).

A regular volumetric grid is a popular and easy to implement data structure to represent the environment but it quickly grows in memory usage as the environment increases in size, specially in 3D applications. In this thesis, we propose a sparse data structure that points to dense areas of the environment. This sparse-dense approach offers the reduced data allocation of a sparse structure but locally it still offers an access speed similar to the regular volumetric grid. The local density assumption allows the data usage to be further compressed by using a lossless data compression method, such as lz4 or zstd. The mechanism that handles data compression is based on a LRU cache and functions online and transparently during mapping. Additionally, for an efficiency duplication of data, an implicit data sharing mechanism (i.e. CoW) with multithreading is also provided. The presented sparse-dense approach improves the current state-of-the-art by providing a more efficient data structure in execution time and memory usage with the additional multithreading support that can be applied to any grid-based model that utilizes our framework.

In this thesis, the simultaneous localization and mapping problem is initially approached from an *online* SLAM perspective and then from a *full* SLAM standpoint. By adapting our scan matching based localization algorithm to incremental building of a map, we were able to develop an *online* SLAM solution that is accurate and fast. The adaptation of the localization algorithm for mapping required the introduction of the dynamic likelihood field that is supported by an Euclidean distance map that efficiently updates only the grid cell affected by the measurements integration. The sparse-dense framework was used to model both the occupancy grid for mapping and the dynamic Euclidean distance grid to take advantage of its

compact data management. By using a SLAM benchmark we demonstrated that our solution offers an accuracy similar or better than the current state-of-the art with the additional advantage of a high computational efficiency. The *online* SLAM solution proposed in this thesis was published in Pedrosa et al. (2016) and it is a continuation of a previous solution published in Pedrosa et al. (2013).

Like all *online* SLAM solutions, our approach is not capable of handling all types of environments, more specifically, environments of significant size that contain large and even nested loops. Therefore, our final efforts were dedicated to develop a *full* SLAM solution that combines all the knowledge obtained so far. Our last proposal is a Rao-Blackwellized particle filter SLAM that uses a proposal distribution with scan matching pose refinement, adaptive resampling with smoothed importance weight and multithreading support. Each particle of the filter integrates an instance of our *online* SLAM that, for efficiency, shares unchanged data with its particle siblings – courtesy of our sparse-dense framework. Because particles are independent, we speeded up the processing times of scan matching and mapping by implementing a *Thread Pool* model that manages multiple concurrent executions threads. The final result is an improved grid-based RBPF SLAM that requires less particles than the current state-of-the-art to generate topologically correct maps and utilizes the available computational resources more efficiently.

Autonomous mobile robots are and will be part of the future. And, as more research is put into the field of robotics and artificial intelligence, mobile robots will be increasingly autonomous and perform more complex operations. This increase of complexity also puts pressure on the available resources, as more systems compete for them. Therefore, improving the efficiency of any system should also be part of its evolution. The contribution presented in this thesis provides a step forward in terms of space and computational efficiency without compromising or even improving accuracy. The obtained results are aligned with the initial statement of this thesis. It is indeed possible to further improve the quality of localization, mapping and SLAM while reducing their computational and memory requirements. By reducing the footprint of these systems, more can be done with the same resources.

6.1 Other Applications and Future Work

The scan matching algorithm presented in this thesis is not restricted to localization. Combined with the sparse-dense framework, it was adapted to be used as a 3D template matcher of an object partial view, obtained from an RBGD sensor, with its complete model. An application of this method can be found in Amaral et al. (2017), where it is used to find the pose of a 3D object for manipulation. Additionally, all the contributions presented in this thesis were also applied to a contemporaneous real robot.

The extension of the scan matching algorithm to a third dimension opens the possibility of exploring 3D robotic localization using our solution, for example, for unmanned aerial vehicles (or drones). Consequently, extending our SLAM solutions to 3D grid-based SLAM is also a possibility. Dense SLAM in an extra dimension is known to be computationally heavy, but the good efficiency of our algorithms and tools can be explored to find a viable solution.

Lifelong mapping is another topic worth of consideration. Traditionally, it is assumed that the environment being mapped is mostly static with small changes over time. But that is not always true, since there are scenarios where the environment can change considerable within hours, e.g. parking lot. Here, our sparse-dense framework could take a central role in providing

support for lifelong mapping. For example, without the direct intervention of the robotic agent it can be used to introduce the notion of temporal validity of information where visited areas would fade over time, thus requiring a remapping of the area. Furthermore, it should also be possible to modify (i.e. add, delete, change) different parts of the map when requested by the agent. Thus allowing different actions to be taken depending on the information (e.g. semantic, topological) that is available.

References

Amanatides, J., Woo, A., et al. (1987). A Fast Voxel Traversal Algorithm for Ray Tracing. In *Eurographics*, volume 87, pages 3–10.

Amaral, F., Pedrosa, E., Lim, G. H., Shafii, N., Pereira, A., Azevedo, J. L., Cunha, B., Reis, L. P., Badini, S., and Lau, N. (2017). Skill-based Anytime Agent Architecture for Logistics and Manipulation Tasks: EuRoC Challenge 2, Stage II - Realistic Labs: Benchmarking. In *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 198–203. IEEE.

Bennewitz, M., Stachniss, C., Behnke, S., and Burgard, W. (2009). Utilizing Reflection Properties of Surfaces to Improve Mobile Robot Localization. In *2009 IEEE International Conference on Robotics and Automation*, pages 4287–4292, Kobe, Japan.

Besl, P. and McKay, N. D. (1992). A Method for Registration of 3-D Shapes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14(2):239–256.

Biber, P. and Strasser, W. (2003). The Normal Distributions Transform: A New Approach to Laser Scan Matching. In *Intelligent Robots and Systems (IROS), 2003 IEEE/RSJ International Conference on*, volume 3, pages 2743–2748. IEEE.

Borenstein, J., Feng, L., and Everett, H. R. (1996). *Navigating Mobile Robots: Systems and Techniques*. A. K. Peters, Ltd., Natick, MA, USA.

Bosse, M. and Roberts, J. (2007). Histogram Matching and Global Initialization for Laser-only SLAM in Large Unstructured Environments. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4820–4826.

Bosse, M. C. (2004). *ATLAS: A Framework for Large Scale Automated Mapping and Localization*. PhD thesis, Massachusetts Institute of Technology.

Bresenham, J. E. (1965). Algorithm for Computer Control of a Digital Plotter. *IBM Systems journal*, 4(1):25–30.

Burgard, W., Derr, A., Fox, D., and Cremers, A. (1998). Integrating Global Position Estimation and Position Tracking for Mobile Robots: the Dynamic Markov Localization Approach. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 2, pages 730–735.

Burgard, W. and Hebert, M. (2008). World Modeling. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 853–869. Springer Berlin Heidelberg, Berlin, Heidelberg.

Burguera, A., González, Y., and Oliver, G. (2009). On the Use of Likelihood Fields to Perform Sonar Scan Matching Localization. *Autonomous Robots*, 26(4):203–222.

Castellanos, J., Martinez-Cantin, R., Tardós, J., and Neira, J. (2007). Robocentric Map Joining: Improving the Consistency of EKF-SLAM. *Robotics and Autonomous Systems*, 55(1):21 – 29.

Censi, A. (2008). An ICP variant using a point-to-line metric. *2008 IEEE International Conference on Robotics and Automation (ICRA)*, pages 19–25.

Censi, A., Iocchi, L., and Grisetti, G. (2005). Scan Matching in the Hough Domain. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2739–2744.

Cole, D. M. and Newman, P. M. (2006). Using Laser Range Data for 3D SLAM in Outdoor Environments. In *IEEE International Conference on Robotics and Automation*, pages 1556–1563, Orlando, Florida. IEEE.

Cox, I. J. (1991). Blanche — An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle. *Robotics and Automation, IEEE Transactions on*, 7(2):193–204.

Cox, I. J. and Wilfong, G. T. (1990). *Autonomous Robot Vehicles*, volume 447. Springer-Verlag New York.

Davison, A., Reid, I., Molton, N., and Stasse, O. (2007). MonoSLAM: Real-Time Single Camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067.

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte Carlo Localization for Mobile Robots. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 2, pages 1322–1328 vol.2.

Dellaert, F. and Kaess, M. (2006). Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. *Int. J. Rob. Res.*, 25(12):1181–1203.

Dickmanns, E. D. and Graefe, V. (1988). Applications of Dynamic Monocular Machine Vision. *Machine Vision and Applications*, 1(4):241–261.

Diosi, A. and Kleeman, L. (2007). Fast Laser Scan Matching using Polar Coordinates. *Int. J. Rob. Res.*, 26(10):1125–1153.

Dissanayake, M. W. M. G., Newman, P., Clark, S., Durrant-Whyte, H., and Csorba, M. (2001). A solution to the simultaneous localization and map building (slam) problem. *Robotics and Automation, IEEE Transactions on*, 17(3):229–241.

Doucet, A. (1998). On Sequential Simulation-Based Methods for Bayesian Filtering. Technical Report CUED/F-INFENG/TR. 310, Cambridge University Department of Engineering.

Doucet, A., De Freitas, N., Gordon, N., et al. (2001). *Sequential Monte Carlo methods in practice*, volume 1. New York: Springer Verlag.

Doucet, A., Freitas, N. d., Murphy, K. P., and Russell, S. J. (2000). Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 176–183, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Eliazar, A. and Parr, R. (2003). DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, pages 1135–1142, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Engelson, S. and McDermott, D. (1992). Error Correction in Mobile Robot Map Learning. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, volume 3, pages 2555–2560.

Eurico Pedrosa, Lau, N., and Pereira, A. (2013). Online SLAM Based on a Fast Scan-Matching Algorithm. In Correia, L., Reis, L., and Cascalho, J., editors, *Progress in Artificial Intelligence*, volume 8154 of *Lecture Notes in Computer Science*, pages 295–306. Springer Berlin Heidelberg.

Eurico Pedrosa, Lau, N., Pereira, A., and Cunha, B. (2015). A skill-based architecture for pick and place manipulation tasks. In Pereira, F., Machado, P., Costa, E., and Cardoso, A., editors, *Progress in Artificial Intelligence*, volume 9273 of *Lecture Notes in Computer Science*, pages 457–468. Springer International Publishing.

Eurico Pedrosa, Pereira, A., and Lau, N. (2017). A Non-Linear Least Squares Approach to SLAM using a Dynamic Likelihood Field. *Journal of Intelligent & Robotic Systems*.

Eurico Pedrosa, Pereira, A., and Lau, N. (April 2018). A Sparse-Dense Approach for Efficient Grid Mapping. In *Autonomous Robot Systems and Competitions (ICARSC), 2018 International Conference on*, Torres Vedras, Portugal. IEEE.

Fairfield, N., Kantor, G., and Wettergreen, D. (2007). Real-Time SLAM with Octree Evidence Grids for Exploration in Underwater Tunnels. *Journal of Field Robotics*, 24(1-2):03–21.

Felzenszwalb, P. and Huttenlocher, D. (2004). Distance Transforms of Sampled Functions. Technical report, Cornell University.

Fischler, M. A. and Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting With Applications to Image Analysis and Automated Cartography. *Commun. ACM*, 24(6):381–395.

Fox, D. (2003). Adapting the Sample Size in Particle Filters Through KLD-Sampling. *The International Journal of Robotics Research*, 22(12):985–1003.

Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999a). Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proceedings of AAAI '99*, pages 343–349, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Fox, D., Burgard, W., Kruppa, H., and Thrun, S. (2000). A Probabilistic Approach to Collaborative Multi-Robot Localization. *Autonomous Robots*, 8(3):325–344.

Fox, D., Burgard, W., and Thrun, S. (1999b). Markov Localization for Mobile Robots in Dynamic Environments. *Journal of Artificial Intelligence Research*, 11:391–427.

Gouveia, B. D., Portugal, D., and Marques, L. (2014). Speeding Up Rao-Blackwellized Particle Filter SLAM with a Multithreaded Architecture. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1583–1588, Chicago. IEEE.

Grisetti, G., Kümmerle, R., Stachniss, C., and Burgard, W. (2010a). A Tutorial on Graph-Based SLAM. *Intelligent Transportation Systems Magazine, IEEE*, 2(4):31–43.

Grisetti, G., Kummerle, R., Stachniss, C., Frese, U., and Hertzberg, C. (2010b). Hierarchical Optimization on Manifolds for Online 2D and 3D Mapping. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 273–278.

Grisetti, G., Stachniss, C., and Burgard, W. (2005). Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2432–2437.

Grisetti, G., Stachniss, C., and Burgard, W. (2007a). Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *Robotics, IEEE Transactions on*, 23(1):34–46.

Grisetti, G., Stachniss, C., and Burgard, W. (2009). Nonlinear Constraint Network Optimization for Efficient Map Learning. *Trans. Intell. Transport. Sys.*, 10(3):428–439.

Grisetti, G., Stachniss, C., Grzonka, S., and Burgard, W. (2007b). A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps using Gradient Descent. In *In Proc. of Robotics: Science and Systems (RSS)*.

Guivant, J. and Nebot, E. (2001). Optimization of the Simultaneous Localization and Map-building Algorithm for Real-Time Implementation. *Robotics and Automation, IEEE Transactions on*, 17(3):242–257.

Gutmann, J.-S. and Fox, D. (2002). An Experimental Comparison of Localization Methods Continued. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 454–459 vol.1.

Hahnel, D., Burgard, W., Fox, D., and Thrun, S. (2003). An Efficient FastSLAM Algorithm for Generating Maps of Large-Scale Cyclic Environments from Raw Laser Range measurements. In *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 206–211. IEEE.

Hähnel, D., Burgard, W., Wegbreit, B., and Thrun, S. (2003). Towards Lazy Data Association in SLAM. In *Proceedings of the 11th International Symposium of Robotics Research (ISRR'03)*, pages 421 – 431, Siena, Italy. Springer.

Hall, B. (2015). *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*, volume 222. Springer.

Herbert, M., Caillas, C., Krotkov, E., Kweon, I. S., and Kanade, T. (1989). Terrain Mapping for a Roving Planetary Explorer. In *Proceedings, 1989 International Conference on Robotics and Automation*, pages 997–1002. IEEE Comput. Soc. Press.

Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-Time Loop Closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, Stockholm, Sweden.

Holz, D. and Behnke, S. (2010). Sancta Simplicitas - On the Efficiency and Achievable Results of SLAM Using ICP-based Incremental Registration. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 1380–1387, Alaska, USA.

Horn, B. K. P. (1987). Closed-form Solution of Absolute Orientation Using Unit Quaternions. *Journal of the Optical Society of America. A*, 4(4):629–642.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees. *Autonomous Robots*, 34(3):189–206.

Howard, A. and Roy, N. (2003). The Robotics Data Set Repository (Radish).

Jensfelt, P. and Kristensen, S. (2001). Active Global Localization for a Mobile Robot Using Multiple Hypothesis Tracking. *Robotics and Automation, IEEE Transactions on*, 17(5):748–760.

Jesus, F. and Ventura, R. (2012). Combining Monocular And Stereo Vision In 6D-SLAM for The Localization of a Tracked Wheel Robot. In *Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on*, pages 1–6.

Julier, S. and Uhlmann, J. (2001). A Counter Example to the Theory of Simultaneous Localization and Map Building. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 4238–4243.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J. J., and Dellaert, F. (2012). iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. *The International Journal of Robotics Research*, 31(2):216–235.

Kaess, M., Ranganathan, A., and Dellaert, F. (2007). iSAM: Fast Incremental Smoothing and Mapping with Efficient Data Association. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1670–1677.

Koenig, S. and Simmons, R. (1998). Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models. *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122.

Kohlbrecher, S., von Stryk, O., Meyer, J., and Klingauf, U. (2011). A Flexible and Scalable SLAM System with Full 3D Motion Estimation. In *Proc. of the IEEE Int. Symp on Safety, Securitym and Rescue Robotics (SSRR)*, pages 155–160, Kyoto, Japan.

Konolige, K., Grisetti, G., Kummerle, R., Burgard, W., Limketkai, B., and Vincent, R. (2010). Sparse Pose Adjustment for 2D Mapping. In *IROS*, Taipei, Taiwan.

Kümmerle, R., Steder, B., Dornhege, C., Ruhnke, M., Grisetti, G., Stachniss, C., and Kleiner, A. (2009). On Measuring the Accuracy of SLAM Algorithms. *Autonomous Robots*, 27(4):387–407.

Kwok, C., Fox, D., and Meila, M. (2004). Real-time Particle Filters. *Proceedings of the IEEE*, 92(3):469–484.

Lau, B., Sprunk, C., and Burgard, W. (2013). Efficient Grid-based Spatial Representations for Robot Navigation in Dynamic Environments. *Robotics and Autonomous Systems*, 61(10):1116–1130.

Lauer, M., Lange, S., and Riedmiller, M. (2006). Calculating The Perfect Match: An Efficient and Accurate Approach for Robot Self-Localization. In *Robocup 2005: Robot soccer world cup IX*, pages 142–153. Springer.

Lee, J. M. (2003). Smooth Manifolds. In *Introduction to Smooth Manifolds*, pages 1–29. Springer.

Leonard, J. and Durrant-Whyte, H. (1991). Simultaneous Map Building and Localization for an Autonomous Mobile Robot. In *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, pages 1442–1447 vol.3.

Leonard, J. J. and Feder, H. J. S. (1999). A Computationally Efficient Method for Large-scale Concurrent Mapping and Localization. In Hollerbach, J. and Koditscheck, D., editors, *Proceedings of the Ninth International Symposium on Robotics Research*, pages 169–176, Utah, USA.

Lima, P. U., Santos, P., Oliveira, R., Ahmad, A., and Santos, J. (2011). Cooperative Localization Based on Visually Shared Objects. In Ruiz-del Solar, J., Chown, E., and Plöger, P., editors, *RoboCup 2010: Robot Soccer World Cup XIV*, volume 6556 of *Lecture Notes in Computer Science*, pages 350–361. Springer Berlin Heidelberg.

Liu, J. S. (1996). Metropolized Independent Sampling with Comparisons to Rejection Sampling and Importance Sampling . *Statistics and Computing*, 6(2):113–119.

Lu, F. and Milios, E. (1997a). Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4(4):333–349.

Lu, F. and Milios, E. (1997b). Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. *Journal of Intelligent and Robotic Systems*, 18(3):249–275.

Madsen, K., Bruun, H., and Tingleff, O. (2004). Methods for Non-Linear Least Squares Problems. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark.

Martinelli, A., Pont, F., and Siegwart, R. (2005). Multi-Robot Localization Using Relative Observations. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2797–2802.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *Eighteenth national conference on Artificial intelligence*, pages 593–598, Menlo Park, CA, USA. American Association for Artificial Intelligence.

Montemerlo, M., Thrun, S., Roller, D., and Wegbreit, B. (2003). FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping That Probably Converges. In *Proceedings of the 18th international joint conference on Artificial intelligence*, IJCAI'03, pages 1151–1156, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Montesano, L., Minguez, J., and Montano, L. (2005). Probabilistic Scan Matching for Motion Estimation In Unstructured Environments. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3499–3504.

Moravec, H. (1989). Sensor Fusion in Certainty Grids for Mobile Robots. In Casals, A., editor, *Sensor Devices and Systems for Robotics*, volume 52 of *NATO ASI Series*, pages 253–276. Springer Berlin Heidelberg.

Moravec, H. P. (1996). Robot Spatial Perception by Stereoscopic Vision and 3D Evidence Grids. Technical report, CMU-RI-TR-96-34. Pittsburgh: Robotics Institute.

Murphy, K. P. (1999). Bayesian Map Learning in Dynamic Environments. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1015–1021, Cambridge, MA, USA. MIT Press.

Neira, J. and Tardos, J. (2001). Data Association in Stochastic Mapping Using the Joint Compatibility Test. *Robotics and Automation, IEEE Transactions on*, 17(6):890–897.

Olson, E. (2008). *Robust and Efficient Mapping*. PhD thesis, Massachusetts Institute of Technology.

Olson, E. (2015). M3RSM: Many-to-many multi-resolution scan matching. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5815–5821.

Olson, E., Leonard, J., and Teller, S. (2006). Fast iterative alignment of pose graphs with poor initial estimates. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2262–2269.

Olson, E. B. (2009). Real-Time Correlative Scan Matching. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 4387–4393, Kobe, Japan.

Pedrosa, E., Lau, N., and Pereira, A. (2013). Online slam based on a fast scan-matching algorithm. In Correia, L., Reis, L. P., and Cascalho, J., editors, *Progress in Artificial Intelligence*, volume 8154 of *Lecture Notes in Computer Science*, pages 295–306, Angra do Heroísmo, Azores, Portugal. Springer Berlin Heidelberg.

Pedrosa, E., Pereira, A., and Lau, N. (2016). A Scan Matching Approach to SLAM with a Dynamic Likelihood Field. In *Autonomous Robot Systems and Competitions (ICARSC), 2016 International Conference on*, pages 35–40, Braganca, Portugal.

Pedrosa, E., Pereira, A., and Lau, N. (2017). Efficient Localization based on Scan Matching with a Continuous Likelihood Field. In *Autonomous Robot Systems and Competitions (ICARSC), 2017 IEEE International Conference on*, pages 61–66, Coimbra, Portugal.

Pedrosa, **Eurico**., Pereira, A., and Lau, N. (April 2017). Efficient Localization based on Scan Matching with a Continuous Likelihood Field. In *Autonomous Robot Systems and Competitions (ICARSC), 2017 International Conference on*, Coimbra, Portugal. IEEE.

Pedrosa, **Eurico**., Pereira, A., and Lau, N. (May 2016). A Scan Matching Approach to SLAM with a Dynamic Likelihood Field. In *Autonomous Robot Systems and Competitions (ICARSC), 2016 International Conference on*, pages 35–40, Bragança, Portugal. IEEE.

Pfister, S. T., Kriechbaum, K. L., Roumeliotis, S. I., and Burdick, J. W. (2002). Weighted Range Sensor Matching Algorithms for Mobile Robot Displacement Estimation. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, volume 2, pages 1667–1674, Washington, DC, USA.

Press, W. H., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). *Numerical recipes in C*. Cambridge, UK: Cambridge University Press, 2 edition.

Rekleitis, I., Dudek, G., and Milios, E. (2002). Multi-robot Cooperative Localization: A Study of Trade-offs Between Efficiency and Accuracy. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2690–2695 vol.3.

Roferi, T. (2002). Using Histogram Correlation to Create Consistent Laser Scan Maps. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 625–630 vol.1.

Roth-Tabak, Y. and Jain, R. (1989). Building an Environment Model Using Depth Information. *Computer*, 22(6):85–90.

Roumeliotis, S. and Bekey, G. A. (2000). Bayesian Estimation and Kalman Filtering: A Unified Framework for Mobile Robot Localization. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 3, pages 2985–2992 vol.3.

Salomon, D. and Motta, G. (2010). *Handbook of data compression*. Springer Science & Business Media.

Schiele, B. and Crowley, J. L. (1994). A Comparison of Position Estimation Techniques Using Occupancy Grids. *Robotics and Autonomous Systems*, 12(3–4):163 – 171.

Smith, R. C. and Cheeseman, P. (1986). On the Representation And Estimation of Spatial Uncertainty. *The International Journal of Robotics Research*, 5(4):56–68.

Thrun, S. (2001). A Probabilistic On-Line Mapping Algorithm for Teams of Mobile Robots. *The International Journal of Robotics Research*, 20(5):335–363.

Thrun, S. (2002). Robotic mapping: A survey. *Exploring Artificial Intelligence in the New Millennium*, pages 1–35.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics*, volume 1. MIT press Cambridge.

Thrun, S., Diel, M., and Hähnel, D. (2003). Scan Alignment and 3D Surface Modeling With a Helicopter Platform. In *In Proceedings of the International Conference on Field and Service Robotics*.

Thrun, S., Fox, D., Burgard, W., and Dellaert, F. (2001). Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence*, 128(1–2):99 – 141.

Triebel, R., Pfaff, P., and Burgard, W. (2006). Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2276–2282. IEEE.

Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (2000). Bundle Adjustment — A Modern Synthesis. In Triggs, B., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer Berlin Heidelberg.

Tsardoulias, E. and Petrou, L. (2013). Critical Rays Scan Match SLAM. *Journal of Intelligent and Robotic Systems*, 72(3-4):441–462.

Van Der Merwe, R., Doucet, A., De Freitas, N., and Wan, E. (2000). The Unscented Particle Filter. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 563–569, Cambridge, MA, USA. MIT Press.

Weiss, G., Wetzler, C., and Von Puttkamer, E. (1994). Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans. In *Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on*, volume 1, pages 595–601.

Ziv, J. and Lempel, A. (1977). A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on information theory*, 23(3):337–343.