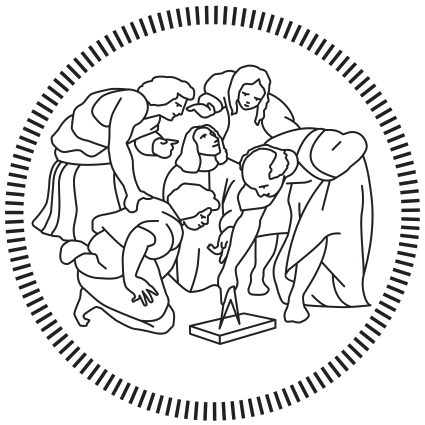
POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria



A method for predicting the performance of SLAM algorithms

Relatore: **Prof. Francesco AMIGONI**
Correlatore: **Dr. Matteo LUPERTO**

Tesi di Laurea Magistrale di:
Valerio CASTELLI, matr. 853992

Anno Accademico 2016-2017

Alla mia famiglia

Ringraziamenti

Questa tesi è il punto di arrivo di un percorso lungo, iniziato quasi sei anni fa e che da allora si è sviluppato attraverso una serie di circostanze diverse e talora inaspettate. Ma soprattutto, è l'ultima tappa di un viaggio che ho condiviso con tante persone, vicine e lontane, senza le quali non sarei mai arrivato dove mi trovo oggi. È quindi giusto e doveroso iniziare questo manoscritto ringraziandole dal profondo del cuore per essere state al mio fianco in tutti questi anni.

Voglio innanzitutto sentitamente ringraziare il Prof. Francesco Amigoni e il Dott. Matteo Luperto per tutto il tempo ed il supporto dedicatomi durante lo svolgimento di questa tesi. Ho imparato molto da loro, sia stilisticamente che umanamente, ed i loro suggerimenti e consigli sono stati punti di riferimento fondamentali per lo sviluppo di questo lavoro.

Ringrazio inoltre il Prof. Matteo Matteucci ed il Prof. Andrea Bonarini per avermi permesso l'utilizzo di Robocom e del laboratorio AIRLab per le prove sperimentali, nonché l'Ing. Giulio Fontana e il Dott. Enrico Piazza per il loro prezioso supporto.

Un sincero grazie va ai miei genitori e alla mia famiglia, per tutto l'incoraggiamento ricevuto in questi anni e per aver sempre sostenuto con passione e generosità tutte le mie scelte. Un pensiero speciale va ai miei zii Domenico e Alberto, per avermi trasmesso la passione per l'informatica e avermi indicato la strada da seguire.

Voglio anche ringraziare tutti gli amici e i compagni con cui ho avuto il privilegio di condividere il mio percorso di studi, per aver contribuito a rendere questi anni un'esperienza unica e indimenticabile. In particolare ringrazio Fabrizio, Lorenzo, Marco e Luca per i tanti momenti felici vissuti dentro e fuori l'Università e per l'affetto e il sostegno morale che non hanno mai mancato di manifestarmi.

Un ringraziamento speciale va a tutti gli amici della vecchia guardia, ed in particolare a Matteo, Gabriele, Lorenzo, Shiyao, Davide e Jessica: grazie per tutti i pranzi, le cene, i viaggi, i campeggi, le feste, le uscite in compagnia

e i bei momenti che abbiamo trascorso insieme. Nonostante il tempo che passa, mi siete sempre rimasti vicini anche quando sarebbe stato più semplice restare in disparte, sopportandomi persino dall'altra parte del mondo e durante i miei momenti peggiori. Per questo, e per molte altre cose che non c'è bisogno di raccontare, avrete sempre un posto speciale nel mio cuore.
Voglio ringraziare anche i miei amici del Vintage Computing Club: Alessandro, Damiano, Francesco e Matteo. Sono contento di poter condividere la mia passione con voi, e spero che continueremo a salvare insieme numerose macchine dall'abisso della distruzione per molti anni a venire.
Dedico inoltre un pensiero particolare a Valerio Enrico Salvo, fedele compagno di avventure da ormai oltre 13 anni: grazie per avermi sempre incoraggiato e per aver creduto in me anche quando non c'era nessun buon motivo per farlo.
Infine, dedico il mio ringraziamento più affettuoso ai miei nonni: a Paolo e Antonio, che sono ancora qui, e a Marilena e Carolina, che mi guardano da lassù. Grazie per tutto l'amore che mi avete donato e per essere stati preziosi esempi e maestri di vita, spero che siate orgogliosi della persona che sono diventato tanto quanto io lo sono di voi.

Sommario

Uno dei principali obiettivi della robotica mobile autonoma è lo sviluppo di robot che siano in grado di agire indipendentemente dal controllo umano e di operare in modo efficiente nell'ambiente in cui si trovano. In molti casi, ciò richiede che il robot abbia la capacità di costruire una mappa dell'ambiente e, simultaneamente, di localizzarsi al suo interno, un compito che è noto con il nome di *Simultaneous Localization And Mapping (SLAM)*. L'importanza di questo problema ha spinto la comunità scientifica a sviluppare un elevato numero di algoritmi SLAM e a proporre una vasta gamma di soluzioni per la valutazione delle loro prestazioni. Tali soluzioni sono però utilizzabili solo per valutazioni a posteriori su dati già collezionati e non permettono di fare previsioni sulla prestazione attesa di un algoritmo SLAM su un ambiente non ancora visitato.

L'obiettivo di questa tesi è di contribuire a colmare questa lacuna mediante lo sviluppo di uno strumento software che permetta la predizione della prestazione attesa di un algoritmo SLAM su un ambiente non ancora visitato, sulla base di caratteristiche note dell'ambiente stesso. Il nostro metodo utilizza simulazioni robotiche automatizzate per misurare le prestazioni di un algoritmo SLAM su un dato insieme di ambienti, costruisce un modello della relazione tra i valori misurati e le caratteristiche degli ambienti in cui sono stati osservati, e utilizza tale modello per predire la prestazione attesa dell'algoritmo SLAM in nuovi ambienti partendo dalle loro caratteristiche.

Il nostro studio considera diverse caratteristiche per descrivere gli ambienti e molteplici metodi di regressione per costruire i modelli, e prevede l'analisi delle loro prestazioni in diversi scenari di valutazione. I risultati ottenuti sia in ambienti simulati sia in esperimenti con robot reali mostrano che il nostro metodo è in grado di catturare in modo adeguato la relazione esistente tra la struttura di un ambiente e la corrispondente prestazione di un algoritmo SLAM, e di predire la prestazione dell'algoritmo SLAM in un nuovo ambiente con notevole accuratezza.

Abstract

One of the main goals of autonomous mobile robotics is the development of robots that are able to act independently from continuous human control and efficiently operate in their environments. In many cases, this requires a robot to build a map of its surroundings while simultaneously keeping track of its position within it, a task that is known as *Simultaneous Localization And Mapping (SLAM)*. The relevance of this problem has led the research community to develop a huge number of SLAM algorithms and to propose several methods for their evaluation. However, such evaluation methods are designed to operate on already collected data and cannot predict the expected performance of a SLAM algorithm on a yet to be explored environment.

The goal of this thesis is to take a first step towards overcoming the limitations of the current SLAM evaluation techniques by developing a software tool that employs known features of unexplored environments to predict the performance of SLAM algorithms in those environments, without requiring the availability of already collected data. The proposed method uses automatized robotic simulations to collect the performance measures of a SLAM algorithm on a number of environments, builds a model of the relationship between the measured performance values and the features of the environments, and exploits such model to predict the performance of the algorithm in unseen environments, starting from the analysis of their features.

We investigate the usage of several types of environmental features and models based on regression methods, and we assess their performance in different evaluation scenarios. Our results on both simulated environments and real robot experiments show that our approach is able to adequately capture the relationship between an environment's structure and SLAM performance and to predict the performance of a SLAM algorithm in an unseen environment with high accuracy.

Contents

Chapter 1

Introduction

One of the main goals of autonomous mobile robotics is the development of robots that are able to act independently from continuous human control and efficiently operate in their surroundings. Among the many prerequisites that a mobile robot must possess in order to fulfill this goal, moving safely in an environment and being able to reach a goal location are fundamental [1]. These tasks usually require the robot to use its sensors to continuously track its position within an internal representation of the environment known as *map*.

In some cases, a map of the environment may be already known to the robot prior to the execution of its assigned task. In most cases, the robot has to build a map of its surroundings while simultaneously keeping track of its position within it, a problem that is known as *Simultaneous Localization And Mapping (SLAM)* or *Concurrent Mapping and Localization (CML)*. Its relevance towards the achievement of higher levels of robot autonomy has lead the research community to develop a wide variety of SLAM algorithms, with new ones being continuously proposed, as surveyed in publications like [2–4], and to develop an equally broad range of approaches for their evaluation, both in terms of testing scenarios [5–8] and of performance metrics [9–12]. However, a common trait of all the proposed evaluation methodologies is that they are designed to conduct evaluations on already collected data, i.e., to assess the performance of a SLAM algorithm in a certain environment only *after* the environment has been explored. This represents a significant obstacle towards a widespread adoption of autonomous mobile robots, as ex post performance evaluation is often impractical and offers very little information about the expected level of performance of a SLAM algorithm in environments other than those on which the evaluation has already been performed.

The goal of this thesis is to take a first step towards overcoming the limitations of the state of the art SLAM evaluation techniques by developing a software tool that employs known features of unexplored environments to predict the performance of SLAM algorithms in those environments. This approach provides a significant impact on the development and deployment of autonomous mobile robots, as it allows designers and manufacturers to assess the suitability of a SLAM algorithm for a given application scenario at design time, eliminating the need for extensive field testing sessions and therefore reducing the cost and length of the development cycle. It also opens up the possibility to perform more extensive, albeit predictive, comparisons between SLAM algorithms, and to estimate the SLAM performance of an already developed robot in a real world scenario in absence of accurate ground truth positioning data.

The starting point of our work is the selection of a performance metric to evaluate the performance of a SLAM algorithm in an environment. Several approaches have been proposed in the literature for this purpose, some of which rely on the visual similarity between the map built by the SLAM algorithm and a ground truth map of the environment [3, 13], while others are based on the assessment of particular properties of the reconstructed map or of the estimated trajectory [9–11]. In this work, we use the *localization error* performance metric proposed by Kümmerle et al. in [11], which measures the performance of a SLAM algorithm as a function of its ability to accurately reconstruct the trajectory followed by a robot in a run executed in an environment. However, the original method proposed in [11] requires a significant amount of human intervention and does not capture the potential variability of SLAM algorithm performance across different runs in the same environment.

We therefore introduce a generalization of this metric to represent the *expected* localization error of a SLAM algorithm in an environment and we propose a system for its prediction that does not depend on the availability of already collected data. In this sense, the proposed method opens the possibility to perform a *predictive benchmarking* of SLAM algorithms, i.e., to anticipate the performance results that would be obtained by a SLAM algorithm in a previously unseen environment.

The proposed method uses automatized robotic simulations to collect the performance measures of a SLAM algorithm on a number of environments, builds a model of the relationship between the measured performance values and the features of the environments, and exploits such model to predict the performance of the algorithm in unseen environments, starting from the

analysis of their features.

In this work, we consider the well-known GMapping [14] SLAM algorithm as the subject of our analysis in view of its widespread usage in the mobile robotics research community. Although GMapping uses odometry and laser data to perform localization and mapping, our methodology does not assume the usage of any particular type of sensor, and is therefore applicable to a wide range of algorithms and application scenarios.

We investigate the usage of several features of the environments based on their geometrical, topological, and structural properties, and we propose two novel quantities that characterize the environments based on the analysis of the environments' skeletons represented by their Voronoi graphs.

We explore the effectiveness of both simple linear regression models and multiple linear regression models in terms of average prediction accuracy and explained error variance on a wide range of simulated indoor environments. We also validate our method on sensory data collected by real robots, both on a publicly available dataset and on our own set of experiments conducted at the AIRLab laboratory at Politecnico di Milano. Our evaluations show that simple linear models based on properties of Voronoi graphs are able to adequately capture the relationship between an environment's structure and the expected localization error of GMapping, and that predictions based on our methodology are able to achieve a high level of accuracy in simulations as well as in real world experiments.

The thesis is structured as follows.

In Chapter 2, we present an extensive, although not exhaustive, review of the state of the art of the field of autonomous exploration and mapping in mobile robotics, we survey some relevant examples of SLAM algorithms, and we review some of the most significant techniques for the evaluation of their performances. We also introduce the problem of performance generalization and prediction, examining some of the shortcomings of the aforementioned evaluation solutions and reviewing the research on this topic within the broader field of autonomous robotics.

In Chapter 3, we discuss the main motivations and goals of this research, we provide a formal characterization of the problem that we aim to solve, and we review the assumptions behind our work.

In Chapter 4, we provide a detailed explanation of our approach to data collection, highlighting the main limitations of the methodology proposed by Kümmerle et al. in [11] and proposing a series of enhancements to improve the scalability, accuracy, and representativeness of the localization error performance metric.

In Chapter 5, we provide an in-depth explanation of the logical design of our solution. We discuss the features that we consider for the characterization of the environments, we present the regression techniques that we adopt for model learning, and we explain how we use the obtained models to perform prediction.

In Chapter 6, we present the architecture of our system and we provide a thorough explanation of the main software components that implement our solution. For each component, we discuss the details of its implementation and we review the technical choices behind its design.

In Chapter 7, we present the setup and the results of the experiments that we conducted to evaluate the validity of our approach. We start by discussing the evaluation procedure and the metrics that we adopted to assess the quality of our models, and we subsequently describe the tests that we performed on our system and the results that we obtained.

In Chapter 8, we give a summary of our work, we draw some conclusions on our results and we offer some suggestions for future research and improvements.

In Appendix A, we provide a reference manual of the parameters that control the behavior of our system and we detail the configurations we adopted for our experiments.

Chapter 2

State of the art

In this chapter, we present an extensive, but not exhaustive, review of the state of the art of the field of autonomous exploration and mapping in mobile robotics. At first, we overview some of the most relevant issues connected with the navigation, exploration and mapping of environments by autonomous robots. Next, we survey some relevant examples of systems taken from the landscape of algorithms that have been proposed to solve the problem of Simultaneous Localization and Mapping (SLAM); this section is by no means a complete study of all the available solutions, but is intended to provide an overview of the most successful approaches. Afterwards, we review some of the most significant techniques for the evaluation of the performance of SLAM algorithms, focusing on several evaluation methodologies and performance metrics. Finally, we introduce the problem of performance generalization and prediction, examining some of the shortcomings of the aforementioned evaluation solutions in real-world applications and reviewing the research on this topic within the broader field of autonomous robotics.

2.1 Exploration and mapping

As noted by Ceriani et al. [1], among the many abilities that a mobile robot must possess in order to act autonomously, moving safely in an environment and being able to reach a goal location are fundamental ones. In particular, this requires the robot to be able to localize itself and its goal in the environment, a task that typically involves the usage of some form of explicit representation of the environment, i.e., a *map*, and the localization of the pose of the robot and of its goal on such map.

In some cases, a map of the environment may be already available for the robot to use. The robot is thus only required to perform self-localization

and navigation, using its sensors to gather sufficient information to produce an accurate estimate of its pose within the map and planning a feasible path from its current pose to the goal.

In other instances, a map of the environment in which the robot operates may not be known in advance. In these cases, the robot must also be capable of building a map of the environment by itself. The combined problem of constructing or updating a map of an unknown environment while simultaneously keeping track of the agent's location within it is called *Simultaneous Localization and Mapping (SLAM)* and is one of the most difficult and challenging tasks that autonomous agents are required to perform.

In order to devise an effective solution for the SLAM problem, several other sub-problems have to be faced and dealt with first; in the following sections, we are going to review some of the most significant ones.

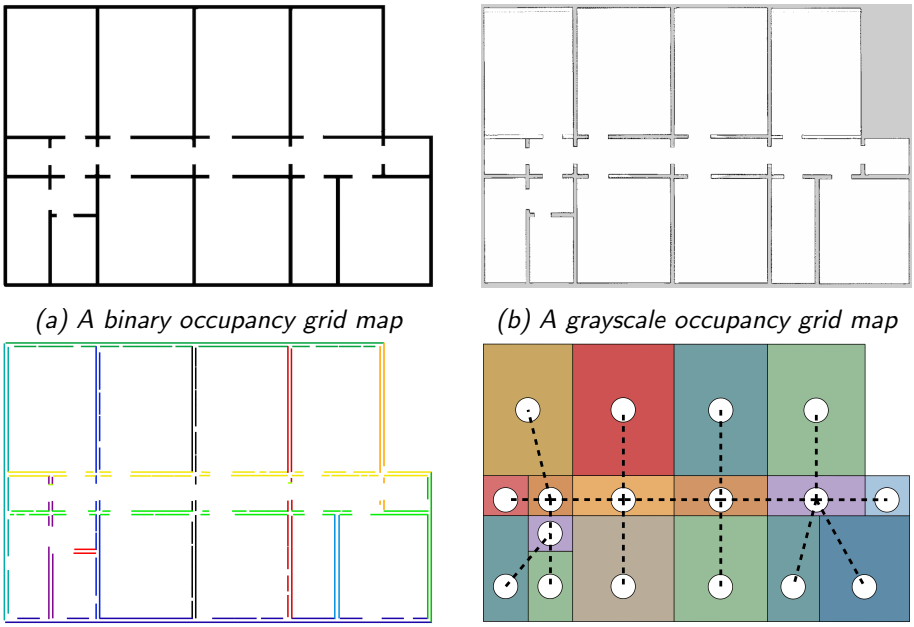
2.1.1 Map representation

A first problem to be solved while designing a comprehensive solution for autonomous navigation and mapping is the choice of the map representation.

A typical choice is to use *occupancy grid maps*, where the environment is modelled as a two-dimensional matrix where each cell stores the probability of its corresponding region of the environment being occupied by an obstacle. Examples of grid-based approaches can be found in [15], [16], and [17]. One way to store and visualize occupancy grid maps is to normalize the probability values of the cells to the range of natural numbers between 0 and 255 and thus employ a 8-bit grayscale representation. Figure 2.1a illustrates this approach when the state of each cell is known with certainty, while Figure 2.1b shows the usage of gray shades to denote areas for which no occupancy information is available.

Another approach that is often used for its memory efficiency is *line-based mapping*. In this case, the map uses line segments anchored to an absolute two-dimensional metric frame to represent obstacles, while free areas are not explicitly stored. Figure 2.1c shows an example of this approach. Some works using line segments for mapping purposes are [18], [19], [20], [21], and [22].

Both occupancy grid maps and line-based maps belong to the family of *metric maps*, as they place objects in a 2D or 3D space. On the contrary, *topological maps* only consider places and relations between them, building a graph whose nodes represent places and arcs represent paths. An example of this approach is shown in Figure 2.1d, in which each node is associated to a room and arcs represent the possibility to directly move from a room to another, for example by means of a door connecting the two. These maps



(a) A binary occupancy grid map

(b) A grayscale occupancy grid map

(c) A line-segments map

(d) A topological map

Figure 2.1: The same floor plan represented with four different types of maps.

often, but not necessarily, also store the relative distances between places. Topological maps have been used, either alone or in combination with metric maps, in several works, including [23], [24], [25], and [26].

2.1.2 Exploration strategy

Besides the choice of an appropriate map representation, another significant problem is the selection of the exploration strategy. Although a robot could roam the environment in a completely random fashion and still eventually collect enough data to build a complete map, real-world applications require a much higher level of efficiency.

One possible option that has traditionally been employed in mobile robotics is to have the robot being remotely controlled by a human operator. However, this approach severely limits the ability of the robot to act on its own and is therefore of limited interest towards the achievement of higher levels of robot autonomy.

Early proposals to overcome this problem include [27], in which Mataric introduced the idea of *wall-following exploration*, and [23], where Thrun et al. discussed a greedy approach in which the robot uses value iteration to always move on a minimum-cost path to the nearest unexplored grid

cell. However, both these approaches have very limited applicability, as they require all walls to intersect at right angles and be clearly visible to the robot.

To overcome these shortcomings, Yamauchi [28] proposed a *frontier-based exploration* paradigm, in which the robot tries to maximize the amount of useful gathered information by moving towards one of several available *frontiers*, i.e., regions on the boundary between free known space and unexplored space. Figure 2.2 illustrates this concept by highlighting frontiers in blue. This approach has essentially become a de-facto standard since its introduction, both in single agent and multiple agents scenarios [29,30]. However, the order in which frontiers are explored has a significant impact on the time required to build a complete map of the environment. Several frontier selection approaches have been proposed, from very simple exploration strategies like nearest frontier [28], farthest frontier, and nearest frontier cluster, to more complex ones, as in [31], [32], [33] and [34]. Work has also been done to quantitatively compare the performance of these approaches, as in [35], [36] and [37].

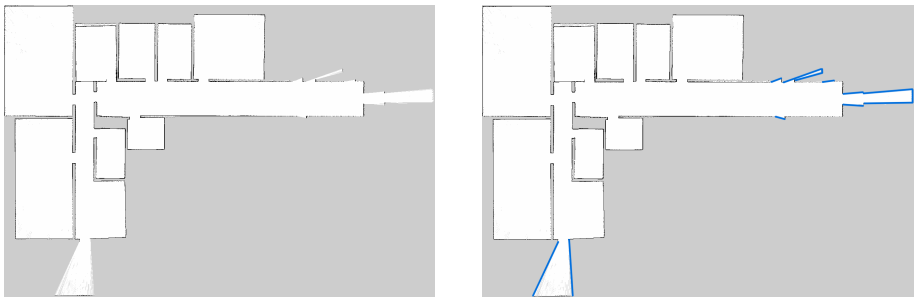


Figure 2.2: A snapshot of the occupancy grid map of an indoor environment during exploration. Frontiers are highlighted in blue in the rightmost image.

2.1.3 Sensors selection

Finally, a fundamental aspect in the development of SLAM algorithms is the selection of available sensors. Much like the various human senses cover different aspects of reality, the sensors of a robot determine which facets of the environment it is able to perceive and use for localization and mapping.

Ultrasound proximity sensors, or sonars, are typically insufficient to perform reliable localization, although there have been attempts to do so [38,39], and are most frequently used in conjunction with other sensors [40] or to perform collision avoidance. Odometry measurements, either coming from rotary informations on wheeled robots or obtained through an *Inertial Mea-*

surement Unit (IMU) [41], are also used to keep track of the amount of distance and rotation travelled by the robot.

Laser range scanners represent a much more complete source of information, as they can operate on distances that range from few millimeters to several meters, are able to work in the darkness, and have a significantly higher accuracy than sonars. In fact, they are a typical choice for high performance applications, and several algorithms [42–45] have been designed to take advantage of their capabilities.

Many approaches to perform localization and mapping using visual information, known as *Visual SLAM*, have also been proposed. In this case, the information may come from plain monochrome [46, 47] or RGB [48] cameras, or from RGB-D cameras that are also able to capture information about depth [49, 50].

The integration of multiple sensors to perform the same task is called sensor fusion and can substantially enhance the performance of SLAM algorithms by compensating for the limitations of each sensor type. However, the choice of the sensors is often dictated by requirements about cost, space, or power consumption, thus practically limiting the amount and diversity of sensory information that can be exploited.

2.2 SLAM algorithms

With SLAM being such a significant problem to solve towards achieving higher levels of robot autonomy, a lot of research has been done on developing efficient solutions.

In 2003, Thrun [51] conducted a survey of the main 2D SLAM techniques available at the time, without focusing on any particular implementation; a similar approach was adopted in 2009 by Kümmerle et al. [11], who classified several 2D SLAM methods according to their underlying estimation techniques. On a more practical note, a 2007 study by Balaguer et al. [2] identified three popular SLAM algorithms for laser-equipped robots. In 2013, Santos et al. [3] surveyed four additional laser-based SLAM techniques available as packages for the *Robot Operating System (ROS)* middleware. Similar surveys continue to be conducted [4] in order to keep track of the latest advancements on SLAM techniques.

Here, we present an overview of the most significant SLAM approaches that have been proposed over the years, as well as some of the most widely used implementations. This section is by no means a complete survey of the available techniques, but is intended to give a general perspective on the

methodologies that have been explored and that continue to be developed by the research community.

2.2.1 Extended Kalman Filters

Extended Kalman Filters (EKFs) were among the first solutions to be proposed to perform simultaneous localization and mapping [52–54], but are still actively explored and improved by the research community [55,56]. The idea of EKF-SLAM algorithms is to exploit *landmarks*, i.e., predefined items that can be perceived by the robot sensors, to identify recurrent locations in the environment. The type of landmarks to be used is strictly dependent on the kind of sensors the robot is equipped with. A landmark may be identified through collisions, proximity detection, computer vision techniques or other means; for instance, Leonard et al. [57] proposed a method that uses sonar information to detect the presence of geometric beacons in the environment.

In EKF-based SLAM, the map does not directly represent a 2D depiction of the environment, as with the case of occupancy grid maps; instead, it is a Gaussian variable that links the current estimate of the robot pose to the estimated pose of every other landmark in the environment.

As with plain Kalman Filters, there are two steps that contribute to the estimation of the robot's pose: the prediction step, which is performed whenever the robot moves and reflects the expected effect of the motor control process on the robot pose, and the correction step, which is triggered by the robot observations of the landmarks. This process can be seen as an application of the Bayes formula, in which the Kalman prediction represents the prior and is combined with the information obtained by the sensory observations to produce a more accurate estimation of the robot's pose. A similar principle is also used to continuously update the map of the environment.

An example of an algorithm that relies on EKFs is *MonoSLAM*[1,2] [48], which uses a standard monocular camera to perform real-time 3D localization and mapping.

The main limitation of EKFs for SLAM applications is their computational complexity, which is quadratic in the number of features, or landmarks, in the environment. This effectively reduces the applicability of full EKFs solutions to small and relatively feature-poor environments, and hinders their scalability to more complex and realistic scenarios. To overcome this limitation, Neira et al. [56] proposed an EKF implementation based on divide-and-conquer that has a reduced computational complexity of $O(n)$.

[1] `https://openslam.org/ekfmonoslam.html`

[2] `https://github.com/rrg-polito/mono-slam`

Independently, Thrun et al. [58] and Eustice et al. [59] investigated the possibility to introduce approximations to enforce sparsity in the information form of EKFs, also known as *Extended Information Filters (EIFs)*, to enhance their performance.

2.2.2 Particle Filters

Similarly to EKFs, *Particle Filters (PF)* are an application of Bayes filters. A particle filter is a non-parametric and recursive Bayes filter in which the posterior probability is directly represented by a set of weighted samples known as *particles*. Each particle is an hypothesis about the state of the environment, which may comprise an estimate of the robot pose, a candidate map of the environment seen so far, or both. The fundamental assumption behind PFs is that the next state of the environment depends only on the current one, i.e., the estimation process is Markovian [60]. Among the many advantages of PFs over traditional EKFs, their ability to represent uncertainty through multimodal distributions and to deal with non-Gaussian noise are particularly important ones.

Some of the most relevant algorithms that are based on this kind of approach are FastSLAM, TinySLAM, DP-SLAM, and GMapping.

FastSLAM [61] uses a peculiar combination of PFs and EKFs to obtain better performance than plain EKF-based algorithms. While at its core it is built around a Kalman filter that tracks the position of a fixed number of predetermined landmarks, it also uses a particle filter to estimate the path posterior and update the estimated robot pose. The particle filter is subject to *Rao-Blackwellization*, a factorization technique that breaks the problem of jointly estimating the robot pose and the map in two separate problems; as the map strongly depends on the estimated pose of the robot, this marginalization process makes the estimation considerably more efficient. FastSLAM has been shown to work with over 50,000 landmarks at a time.

TinySLAM[3,4] [43], also known as CoreSLAM[5] in one of its implementations for the Robot Operating System (ROS) robotic middleware[6], is a 200 lines of C-language code SLAM algorithm that was designed to be as simple and easy to understand as possible, while simultaneously maintaining an acceptable level of performance. In order to perform localization and

[3] `https://openslam.org/tinyslam.html`
[4] `http://wiki.ros.org/tiny_slam`
[5] `http://wiki.ros.org/coreslam`
[6] `http://www.ros.org/`

mapping, TinySLAM requires both odometry information and obstacle distance data provided by a laser range scanner. In TinySLAM, each particle is an hypothesis on the current pose of the robot in the environment and has an associated weight, which represents the likelihood of that hypothesis being true, that gets updated at every new laser observation according to a scan-to-map distance function; the best hypotheses are kept, the worst are eliminated, and new ones are generated. To keep complexity to a minimum, the algorithm maintains a single estimate of the environment's map at any time.

Although still based on particle filters, *DP-SLAM*[7] [62] implements a much more complex technique for SLAM estimation. From a sensory point of view, it requires the same data of TinySLAM, i.e., odometry and laser scans. However, instead of using particles to just represent hypotheses on the robot pose, DP-SLAM uses them to store both the estimated robot pose and a candidate map of the environment. It does so by using a peculiar data structure known as *Distributed Particle (DP)*, which allows the algorithm to share common map parts across several particles, thus significantly reducing the memory footprint per particle and allowing for a much higher number of hypotheses at any given time.

Developed by Grisetti et al. [14,42], *GMapping*[8,9] also uses sensory information from odometry and laser scans to perform localization and mapping. Differently from DP-SLAM, particles carry individual candidate maps of the environment, without sharing any data and with no explicit modelling of the robot pose. Similarly to FastSLAM, the particle filter is Rao-Blackwellized; however, GMapping also relies on two additional techniques to improve the overall mapping accuracy. First, the particles distribution is built by directly incorporating information from the latest laser observation into the model, instead of relying just on odometry data; this significantly reduces the estimation error, so that less particles are required to represent the posterior. Second, adaptive resampling is used to reduce the total number of particles only when needed, thus keeping a greater variety of hypotheses and increasing the accuracy of the estimation.

2.2.3 Constrained optimization techniques

Finally, the SLAM problem has also been faced from the point of view of *constrained optimization methods*. The idea behind this family of techniques

[7] `https://users.cs.duke.edu/~parr/dpslam/`
[8] `https://www.openslam.org/GMapping.html`
[9] `http://wiki.ros.org/slam_GMapping`

is that, at its core, SLAM is an optimization problem, whose goal is to find the most likely hypothesis for the environment's map and the robot pose within it given the available sensor observations. Traditional maximum likelihood approaches are unapplicable to SLAM due to the high number of constraints that need to be taken into account; however, several techniques have been presented to overcome this issue by using approximations [38,63].

One of the most wildly used algorithms based on this approach is *HectorSLAM*[10] [45]. HectorSLAM is designed to work with laser range scanners, or LIDARs, and it supports 3D navigation using an inertial sensing system. As it does not rely on odometry, it is an ideal candidate for aerial applications; however, this can be a drawback in applications where odometry information is available. The algorithm matches the scans obtained by the LIDAR sensor using a Gauss-Newton optimization method to solve a least square error minimization problem and find the rigid transformation that best fits the projected laser beams with the map. For aerial applications that need 3D state estimation, an EKF is also used.

Another method based on constrained optimization is *LagoSLAM* [64], a graph-based SLAM algorithm developed by Carlone et al. in which nodes represent the relative poses assumed by a mobile robot along a trajectory and edges denote the existence of a relative measurement between two poses. The algorithm attempts to estimate the set of absolute poses P in the reference frame F that maximize the likelihood of the observations by minimizing a nonlinear, non-convex cost function through a series of local convex approximations.

Finally, it is worth noting that progress in this field has not been confined to the academic research community, but has also seen active development within the robotic industry. An example of commercial graph-based SLAM algorithm is *KartoSLAM*[11], a localization and mapping solution developed by Karto Robotics[12].

2.3 SLAM performance evaluation

As the number of algorithms developed to solve the SLAM problem keeps growing, finding reliable ways to assess their performance has become a significant challenge for the robotics research community. This problem is in fact non trivial, as different SLAM algorithms applied to the same environment may produce very different results, as shown in Figure 2.3. In addition,

[10] `http://wiki.ros.org/hector_slam`

[11] `http://wiki.ros.org/slam_karto`

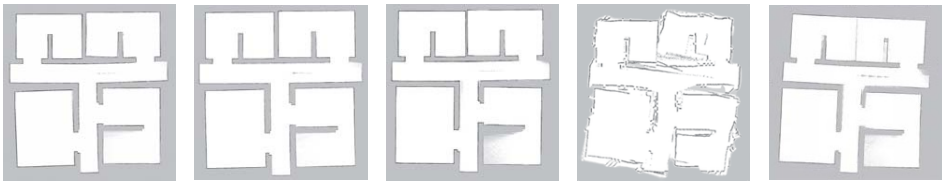
[12] `https://www.kartorobotics.com`

different SLAM algorithms may use different map representations, making the comparison of their results difficult.

2.3.1 Evaluation scenarios

The first step in the development of reliable SLAM evaluation approaches is the identification of appropriate evaluation scenarios in which SLAM algorithms can be tested.

One approach that has been extensively investigated by the research community is to perform *robotic competitions*. In a robotic competition, a custom evaluation environment is created to either mimic a real application environment or to be particularly hard for a robot to handle. The robot is then assigned a task whose execution typically requires it to move from one point of the environment to another; the performance of the used SLAM algorithm is then indirectly evaluated by means of the overall performance of the robot at the given task. As the quality of the reconstructed map affects the ability of the robot to successfully complete the task, it is assumed that the systems that achieve the highest overall performance are also the best performing on the SLAM subtask. This approach has been used to evaluate, among others, the performance of cleaning robots [IROS, 2002], self-driving cars in an urban area [Darpa, 2007], rovers moving in a simulated Mars setting [ESA, 2008], and robots operating in Urban Search and Rescue scenarios [RoboCup Federation, 2009]. It has also been used for the evaluation of domestic service robots and industrial robots in the context of the RoCKIn project[13] of the European Robotic League[14], as documented in [5], [65], and [66].



(a) HectorSLAM (b) GMapping (c) KartoSLAM (d) CoreSLAM (e) LagoSLAM

Figure 2.3: Occupancy grid maps of the same environment obtained by different SLAM algorithms through Stage simulations [3].

Two important shortcomings of robotic competitions are their limited scale and little resemblance to real application environments. This is due to

[13] `http://rockinrobotchallenge.eu/`

[14] `https://www.eu-robotics.net/robotics_league/`

the complexity and cost of setting up a mock evaluation environment, which results in most competitions being held in relatively small settings that are poor proxies for the variety of environments that characterize real-world applications.

Another significant shortcoming of competitions is the significant complexity and diversity of hardware and software settings that have an impact on the robots' performances. As each robot has a different hardware architecture, software stack, and selection of navigation and mapping parameters, it is extremely difficult to say how much of a robot's performance is due to the choice of the SLAM algorithm and how much is instead due to other factors. Moreover, the task is typically performed a limited number of times, or in some cases just once; as we will discuss in Section 3.2, however, the performance of a SLAM algorithm is not guaranteed to remain stable across multiple repetitions of the same task, so that an individual execution of the task may not be an accurate depiction of the actual average performance of the algorithm at that task.

These limitations, in addition to both hardware and software settings of the robots being tuned to the specific context of each competition, make it difficult to draw conclusions on the expected performance of a given SLAM algorithm in more realistic settings.

To overcome some of these problems, the robotics community has proposed to collect and publish collections of standard datasets for anyone to use as benchmarks. Evaluating SLAM algorithms on standardized sets of environments and robot settings is also a good experimental practice, as it ensures reproducibility, replicability, and comparability of the results similarly to what well-defined experimental methodologies guarantee in other branches of science.

Radish[15,16] [6] is one of the oldest efforts in this direction. Proposed by Howard and Roy in 2003, it offers a collection of over 40 datasets of explorations conducted by real robots in many different, although mostly indoor, environments. The datasets typically consist of records of the sensory data collected during a run of a robot, an indication of which SLAM algorithm has been used, and a visual representation of the final map produced. However, the datasets do not adhere to a common standard and have different levels of completeness. Most of them include laser scans and odometry data, while others include sonar data or monocular camera snapshots. The recorded data are sometimes offered in Carmen Log File (CLF) format, as it is commonly used in conjunction with ROS, but this is not always the

[15] `http://radish.sourceforge.net`

[16] `http://cres.usc.edu/radishrepository/view-all.php`

case. Some of the environments are explored by autonomous robots, whereas in others the robots are tele-operated. Almost none of the available datasets include ground truth data, neither of the followed trajectory nor of the floor plan of the building, preventing the evaluation of many performance metrics that are based on some form of a priori knowledge about the actual map and trajectory. Finally, although it is still currently widely used by the research community, the website has not been updated in over eight years, so it doesn't reflect the state of the art of SLAM technology anymore.

Fontana et al. [1] conducted a similar, but more ambitious, effort in 2008 called the *RAWSEEDS project*[17] [7]. It was proposed to overcome the limitations of Radish by performing multiple explorations of each environment, publishing recorded data in a defined and well-documented standard, simultaneously collecting information from a variety of sensors, and offering ground truth data of both the robot trajectory and the building floor plan of each exploration. Unfortunately, it only collected very few datasets, and it hasn't been updated since 2009.

Other publicly available datasets include runs performed at the KTH Royal Institute of Technology[18] and at the MIT Killian Court location[19], both of which were collected in the early 2000s. More recent efforts include dataset collections from the Computer Vision Group of the Technical University of Munich[20] [67,68]. Additional resources can be found on the website of the OpenSLAM initiative[21].

Having a real robot roaming a physical environment, either autonomously or in a tele-operated fashion, is also a major hurdle for the evaluation of SLAM algorithms because of the significant time and money that are necessary to perform the experiments. Moreover, it dramatically limits the scale and diversity of the tests that can be reasonably conducted to rather small environments, like specially fitted laboratory rooms, that fail to replicate the complexity of many real-world application environments. *Simulation* has been proposed as a possible solution to this problem, leading to the development of several robotic simulators.

The *Player/Stage Project*[22] [8] is a suite of robotic tools developed by Howard et al. since 2003 to simplify the design, implementation and testing of robots. While Player provides a distributed, simple and clean control

[17] `http://www.rawseeds.org/home/`
[18] urlhttp://www.nada.kth.se/ johnf/kthdata/dataset.html
[19] `http://www.ijrr.org/contents/23_12/abstract/1113.html`
[20] `https://vision.in.tum.de/data/datasets`
[21] `http://openslam.org/`
[22] `http://playerstage.sourceforge.net`

interface for a robot's sensors and actuators, Stage[23,24] is a lightweight robot simulator that can handle populations of hundreds of virtual robots in a two-dimensional bitmapped environment. Stage supports a variety of actuators and sensors, including grippers, wifi modules, blinking lights, lasers, sonars and infrared ranger sensors. Simulated robots can be differential-steer drive models, omnidirectional models, or car-like; a simple odometry model allows to simulate the effects of a uniformly distributed random error on odometry readings. The suite can also be used in conjunction with ROS.

To overcome the limitations of two-dimensional simulations, Howard et al. introduced *Gazebo*[25,26] [69], an open-source robotic simulator that recreates 3D dynamic multi-robot environments. It is built on top of the Open Dynamics Engine[27] to accurately simulate the dynamics and kinematics of articulated rigid bodies. Compared to Stage, which simulates a purely 2D environment, Gazebo enables more complex simulations that accurately mimic real-world physics and offer the possibility to simulate a broader range of actuators and sensors, which can also be extended via third party plugins.

USARSim[28] [70] is an alternative robotic simulator developed by Carpin et al. to support the virtual robots competition within the RoboCup initiative. Despite being originally conceived for Urban Search and Rescue applications, it has evolved into a full-fledged, general purpose multi-robot simulator that can be extended to model arbitrary application scenarios. USARSim leverages Unreal Engine[29], a commercially available game engine produced by Epic Games, Inc.[30], to simulate 3D environments with high fidelity. Like Gazebo, it supports a wide range of actuators and sensors that can be extended via third party plugins.

2.3.2 Performance metrics

The second step in the development of reliable SLAM evaluation approaches is the definition of a suitable performance metric. To this regard, several approaches have been proposed over the years.

In [2], Balaguer et al. assess the performance of three SLAM algorithms - GMapping, GridSLAM, and DP-SLAM - by visually estimating the fidelity of the reconstructed maps of a set of indoor environments with respect to

[23] `https://github.com/rtv/Stage`
[24] `http://wiki.ros.org/stage`
[25] `http://gazebosim.org`
[26] `http://wiki.ros.org/gazebo_ros_pkgs`
[27] `http://www.ode.org`
[28] `https://sourceforge.net/p/usarsim/wiki/Home/`
[29] `https://www.unrealengine.com`
[30] `https://www.epicgames.com`

their ground truth counterparts, both in real world and in simulation with USARSim. However, the study lacks rigour and replicability, as the chosen performance metric is inherently subjective.

In [13], Amigoni et al. propose a methodology to address these shortcomings. According to this study, in order to effectively evaluate and compare different SLAM algorithms it is necessary to: i) provide extensive information about the produced maps, ii) report the behavior of the mapping system for different values of the parameters, iii) include one or more examples of maps produced following a closed loop path, and iv) whenever a ground truth map is available, use it to assess the quality of the estimation by evaluating its distance from the produced map according to a well-defined similarity metric. Furthermore, the datasets used to perform the evaluation must be publicly available, in order to let other researchers replicate the results.

In [9], Colleens et al. argue that comparing the map produced by a SLAM algorithm to its ground truth counterpart is not an appropriate evaluation metric. In fact, they claim that the main purpose of SLAM algorithms is not to produce human-understandable maps, but rather to create an appropriate representation of the robot's surroundings to enable the execution of some other task, like navigation. In this context, the degree of accuracy of a produced map with respect to a known ground truth does not necessarily reflect its usefulness, as schematic representations of the environment could still be sufficient for the completion of the desired task while more visually faithful representations could lack details, like doors and passages, that are crucial for the successful completion of the task. For this reason, they propose to use two different metrics: the first is the degree to which the paths created in the generated map would cause the robot to collide with a structural obstacle in the real world, and are therefore invalid; the second is the degree to which the robot should be able to plan a path from one pose to another using the generated map, but cannot because such paths are invalid in the ideal map. However, this approach also suffers from two main limitations, as it can only be applied to SLAM algorithms that produce occupancy grid maps as outputs and it requires precise alignment of the produced map with the ground truth map in order to verify whether paths identified in one of the two maps are also viable in the other.

In [7] and [10], Fontana et al. propose the adoption of several methodologies for SLAM algorithms evaluation depending on the context of application. Expanding on the idea described by Colleens et al. in [9], they propose to adopt performance metrics that capture the ability of SLAM algorithms to produce useful maps in the contexts of localization and navigation. Their main contribution in this respect is the introduction of a *Self-Localization*

Error metric, computed by processing the distance errors between the estimated pose of the robot and the corresponding pose from the ground truth trajectory, and an *Integral Trajectory Error* metric, which is similar to the self-localization error but focuses on the overall distance between the reconstructed and ground truth trajectories over the whole path of the robot. It must be noted that both these metrics require ground truth trajectory information; Ceriani et al. address this problem in [1], in which they present two methodologies for collecting ground truth data for indoor localization and mapping based, respectively, on a network of fixed cameras and on a network of fixed laser scanners.

A significant drawback of metrics that rely on absolute poses is that they are strongly influenced by the timestamp at which an error occurs. This introduces a strong bias in the evaluation, as the same error may lead to very different results depending on whether it is introduced at the beginning, in the middle, or at the end of an exploration. Consider for instance a rotation error of a few degrees: if the error is introduced at the end of an exploration, its effect will be limited to a very small number of poses and its impact on the overall performance will be negligible; if however it occurs at the beginning of the exploration, the reconstructed trajectory will rapidly diverge from the ground truth one, leading to a much higher measured error despite the map still being substantially correct.

To overcome this limitation, Kuemmerle et al. in [11] and in [71] propose a metric that considers the deformation energy that is needed to transfer the estimated trajectory onto the ground truth trajectory, i.e., that is based on the relative displacements between poses. However, Kuemmerle et al. do not provide a definite criterion to choose which relative displacements should be considered to compute the metric, noting that, in absence of ground truth information, close-to-true relative displacements can be obtained by other sources of information, such as background human knowledge about the length of a corridor or the shape of a room.

In the context of SLAM evaluation through map evaluation, Birk et al. investigate in [12] and in [72] the possibility to use *topology graphs* derived from *Voronoi diagrams* to capture high-level spatial structures of indoor environments. A Voronoi diagram is a partition of the space into *cells*; each cell encloses a *site*, i.e., a point on the map that represents an obstacle, and contains all points of the map whose distance to the site is not greater than their distance to all other sites. The graph is then obtained by considering the boundaries of said cells and applying a number of post-processing steps. Figure 2.4 shows a examples of filtered topology graphs.

In [3], Santos et al. evaluate the performance of five SLAM algorithms

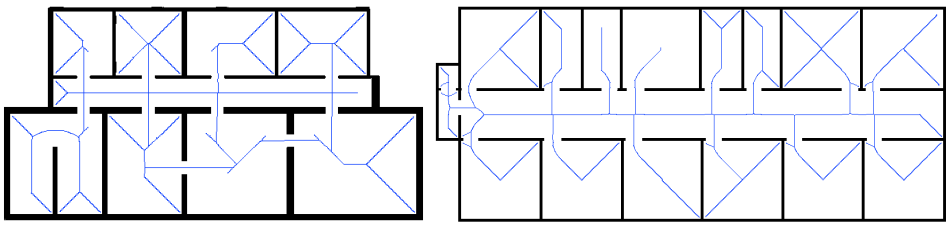


Figure 2.4: Topology graphs of two sample maps.

- HectorSLAM, GMapping, KartoSLAM, CoreSLAM and LagoSLAM - in terms of distance between the generated map and the ground truth map using a performance metric based on the k-nearest neighbor concept.

Finally, Krinkin et al. [73] propose to evaluate the quality of generated maps by measuring three different aspects: the proportion of occupied and free cells to determine blur, the amount of corners in the map, and the amount of enclosed areas.

2.4 Performance generalization and prediction

As documented in the previous section, the research community has not agreed on a single method for SLAM algorithms evaluation, proposing instead several metrics to measure different aspects of their performance. However, a common trait of all these solutions is that they are designed to conduct evaluations on already collected data, i.e., they can assess the performance of a SLAM algorithm in a certain environment only *after* the environment has already been explored. This is a significant limitation for three reasons.

The first is that ex post evaluations are typically rather difficult to perform, especially in large environments. This is due to the need of gathering both sensory information and ground truth data in order to assess how much the SLAM algorithm's results differ from reality in term of quality of map reconstruction and trajectory estimation. While reference floor plans and blueprints can usually be obtained with ease, the same doesn't hold true for ground truth trajectory data, which requires extremely accurate tracking systems in order to be measured with adequate precision. An example of such a system is *OptiTrack*[31], a positioning system developed by NaturalPoint, Inc.[32] in 2009 that uses a variable number of synchronized infrared

[31] `https://www.optitrack.com`
[32] `https://www.naturalpoint.com`

cameras, each containing a grayscale CMOS imager capturing up to 100 FPS, to triangulate the pose of an infrared reflector placed on the robot itself. As the environment becomes larger, the number of required cameras grows in the hundreds, making the system impossibly expensive and complex to operate.

Even when it is possible to have highly accurate ground truth data, ex post evaluations in real environments may be impractical to perform. All settings being equal, the performance of a SLAM algorithm in an environment may in fact vary across multiple explorations, so that an individual exploration of the environment may not accurately represent the actual average performance of the algorithm in that setting. A possible countermeasure is to repeat the measurements many times, which however turns the evaluation procedure into a quite time-consuming process. We will discuss this issue more in depth in Section 3.2.

These two limitations can be overcome by taking advantage of computer simulations, which offer perfect pose tracking and can be often executed in parallel on dedicated servers to increase the throughput, while still providing reasonably accurate results [2, 4]. However, there is a third fundamental limitation of these evaluation methods that affects both real-world and simulated explorations: their lack of generalization capabilities.

Generalization is a fundamental aspect in all branches of science, as it provides the means to make predictions on the general behavior of a system in a wide range of situations starting from a much smaller set of observations. In our context, by generalization we mean the process of identifying significant correlations between the performance of a given SLAM algorithm in a setting and its performance in a different setting. A *setting* is defined by the environment in which the algorithm is tested, the accuracy and the capabilities of the robot sensors, and the specific values of the algorithm parameters. Each of the examined evaluation methods proposes a different and potentially equally valid way to assess the a posteriori performance of a SLAM algorithm after an exploration has taken place; however, none of them provides a way to use previously collected performance data to make predictions about the performance of a SLAM algorithm in a setting *before* actually exploring the environment.

This limitation is significant, since knowing how well a SLAM algorithm performs in a certain setting does not immediately provide any information as to how well it will perform in a different setting, as the measured level of performance is strictly dependent on the characteristics of the setting itself. The problem is exacerbated by the fact that there is no single best SLAM algorithm for every scenario. As an example, a study [4] conducted by Tur-

nage using the Hausdorff distance between the ground truth map and the reconstructed map as a metric to assess the relative level of performance of HectorSLAM, CoreSLAM, and GMapping on three different environments shows that, while HectorSLAM outperforms both CoreSLAM and GMapping in two environments out of three, CoreSLAM performs best in the remaining one, with HectorSLAM and GMapping performing the same.

However, the ability to perform a prior assessment is crucial towards enabling more pervasive applications of mobile robotics, as it vastly simplifies the deployment of robots in real-world contexts. The absence of generalization in SLAM performance assessment increases the difficulty in knowing at design time which SLAM algorithm and robot configuration best fits a given application scenario, often requiring a cumbersome and expensive trial and error process.

Research has been done on the broader topic of promoting generalization in robotics. On one hand, there have been attempts to explicitly improve the generalization capabilities of algorithms for robotic applications. One such example is [74], in which Pinville et al. propose a supervised learning approach to improve the generalization capabilities of controllers in evolutionary robotics. On the other hand, researchers have focused on the problem of developing approaches to predict robotic performance in a wide variety of contexts.

In [75], the authors investigate the usage of neural networks, fuzzy systems, genetic algorithms, and other soft computing techniques to predict the performance of several industrial machining processes.

Considering autonomous wheeled robots, in [76] and in [77] Young et al. propose a model to assess the traversal cost of a natural outdoor environment for an autonomous vehicle using A* planning. The model exploits information about the complexity of the environment itself, including the slope of the terrain and the presence of vegetation, to ultimately predict the average speed of the vehicle. In [78], Regier et al. introduce a method to estimate the traversal time of a path by using its length, its smoothness, and its obstacle clearance as features of a non-linear regressor. The model, which is trained by using Gazebo simulations of an omnidirectional robot in a variety of maps, is shown to significantly outperform predictions only based on path length. Dawson et al. highlight in [79] the limits of several performance prediction systems in estimating the average coverage time of an environment in multi-robot autonomous exploration, citing the difficulty for simulations to properly take into account the slowdowns introduced by inter-robot communications, physical interferences and network latency.

In [80], Amigoni et al. argue that a major hurdle towards achieving a

higher level of generalization of experimental results in autonomous robotics is the limited representativeness of the experimental settings. This is especially true for physical experiments involving actual robots, which are often conducted in small and carefully tuned labs that are very poor proxies for the variety of environments that characterize real-world applications.

2.5 Summary

In this chapter, we discussed the state of the art in the field of autonomous exploration and mapping in mobile robotics. At first, we overviewed the main issues connected with the navigation, exploration, and mapping of environments by autonomous robots. Next, we surveyed the landscape of algorithms that have been proposed to solve the problem of Simultaneous Localization and Mapping (SLAM), focusing on the most successful approaches and implementations. Next, we discussed the state of the art of performance evaluation for SLAM algorithms, reviewing several evaluation methodologies and performance metrics that have been proposed to this end. Finally, we examined how the limited applicability and the lack of generalization capabilities of such evaluation methodologies seriously hinder their usefulness in assessing the performance of SLAM algorithms in real-world scenarios.

Chapter 3

Problem formulation

In this chapter, we present a formal definition of the problem that we address in this thesis. We start by analyzing the motivations of this research, we set the thesis goal, and we introduce the concept of predictive benchmarking. We then review some preliminary definitions and we provide a formal characterization of the learning problem that we aim to solve. Finally, we discuss the scope of our analysis and we list the assumptions underlying our work.

3.1 Motivations and goal

As we mentioned in the previous chapter, the state of the art of SLAM algorithms performance evaluation is focused on the ex post assessment of the results of robot runs in benchmark environments, either using simulation [2] or with the use of benchmark datasets [5–7]. This kind of retrospective analysis can be useful for several comparison and evaluation purposes, but results obtained with such methodology can be difficult to generalize to the task of estimating the expected level of performance of a SLAM algorithm in a yet to be tested setting, like a previously unseen environment, different operational specs and types of the robot sensors, or when alternative values of the parameters are chosen [4]. One of the consequences of this limitation is a difficulty in extending the results obtained in controlled settings to the prediction of the expected performance of a SLAM algorithm in an actual application environment, thus severely limiting the deployment of autonomous mobile robots in real world and daily usage scenarios.

Consider the development of a robot for patrolling buildings. In order to safely, efficiently, and autonomously navigate a building, the robot must be able to build an internal representation of its surroundings and to correctly track its own pose as it roams the environment, hence it must run a

SLAM algorithm. However, every building has its own characteristics due to a combination of age, architectural style, purpose, number of hosted people, and many other aspects. A kindergarten in a small mountain community is likely to have a different shape, size, and internal structure from the office of a large corporation, as a university campus is considerably different from a factory, and a hospital from a home. Different applications have different budget constraints as well as different safety and accuracy requirements, imposing restrictions on the sensors and the SLAM algorithm to be used. As the performance of a SLAM algorithm is strictly dependent on the characteristics of the environment in which it is run and on the capabilities of the sensors used to collect the data, the lack of generalization of state of the art SLAM evaluation techniques results in long and extensive phases of prototyping and field testing for each specific application scenario in order to tune the robot configuration and achieve the required level of accuracy. Furthermore, as these evaluations can only be done a posteriori, it may be difficult to get an accurate estimate of the cost associated with certain safety and accuracy requirements before building the robot and testing it in its final operational environment, as the same performance level may require significantly different hardware depending on the context. This severely impacts the ability of potential customers to plan their investments and ultimately limits the adoption of autonomous mobile robots.

The goal of this thesis is to take a first step towards overcoming the limitations of state of the art SLAM evaluation techniques by developing a software tool that employs known features of unexplored environments to predict the performance of SLAM algorithms in those environments.

To this end, we introduce the concept of *predictive benchmarking*, a method that reuses the performance results obtained by SLAM algorithms in benchmark environments to predict their performance in new ones. Compared to other SLAM evaluation techniques [3, 9–12, 73], predictive benchmarking has the advantage of providing an estimate of the expected level of performance of a given SLAM algorithm in new environments without requiring their exploration.

While existing SLAM evaluation techniques handle every new environment as a separate standalone problem, predictive benchmarking builds upon the knowledge gained with each new exploration to create a model of the relationship between an environment and its associated SLAM performance; this results in increasingly accurate predictions over new environments as the number of explorations used for training increases.

This approach significantly simplifies the development and deployment of autonomous mobile robots by allowing designers and manufacturers to

evaluate at design time the suitability of a certain choice of SLAM algorithm, parameters values and sensors for a given scenario, eliminating the need for extensive field testing sessions and therefore reducing the cost and length of the development cycle. It may also be used as a basis to perform comparisons between SLAM algorithms at a lower cost and on a broader set of environments than what is possible with traditional SLAM evaluation techniques, and to estimate the SLAM performance of an already developed robot in a real world scenario in absence of accurate ground truth positioning data.

3.2 Localization error performance metric

Before we proceed with the detailed discussion of our work, it is first necessary to introduce some preliminary definitions and concepts. In the following paragraphs, we present and motivate our choice for the metric to be used for the evaluation and the prediction of the performance of a SLAM algorithm, we review its formal definition for single robot runs, and we propose a generalization to multiple runs of a single environment.

3.2.1 Metric definition

The research community has developed a variety of metrics to assess the performance of SLAM algorithms. In the context of this thesis, we use the *localization error* performance metric proposed by Kümmerle et al. in [11]. This metric measures the performance of a SLAM algorithm as a function of its ability to accurately estimate the trajectory followed by a robot in a single run. To do so, the metric computes the deformation energy that is required to transfer the estimated trajectory onto the ground truth trajectory: the smaller the energy, the higher the accuracy of the reconstruction.

The choice of Kümmerle's metric was dictated by its generality and versatility. Compared to other evaluation strategies, such as those based on the comparison of different maps of the same environment, Kümmerle's approach is not restricted to any particular map representation format and can thus be used to measure the performance of any SLAM algorithm; it is also independent of the type of sensors mounted on the robot, and is therefore applicable to a broad range of scenarios.

We hereby recall the formal definition of the metric as given by Kümmerle et al. in [11]:

Definition 3.1. *Let $x_{1:T}$ be the poses of the robot estimated by a SLAM algorithm from time step 1 to T during an exploration of environment E, $x_t \in SE(2)$, with $SE(2)$ being the usual special Euclidean group of order 2. Let $x^*_{1:T}$ be the associated ground truth poses of the robot during mapping. Let $\delta_{i,j} = x_j \ominus x_i$ be the relative transformation that moves the pose x_i onto x_j, and let $\delta^*_{i,j}$ be the transformation based on x^*_i and x^*_j accordingly. Finally, let δ be a set of N pairs of relative transformations over the entire exploration, $\delta = \left\{ \langle \delta_{i,j}, \delta^*_{i,j} \rangle \right\}$.*
The localization error *performance metric is defined as:*

$$\begin{aligned}
\varepsilon(\delta) &= \frac{1}{N} \sum_{i,j} (\varepsilon(\delta_{i,j}))^2 = \\
&= \frac{1}{N} \sum_{i,j} (\delta_{i,j} \ominus \delta^*_{i,j})^2 = \\
&= \frac{1}{N} \sum_{i,j} [trans(\delta_{i,j} \ominus \delta^*_{i,j})^2 + rot(\delta_{i,j} \ominus \delta^*_{i,j})^2] = \\
&= \varepsilon_t(\delta) + \varepsilon_r(\delta),
\end{aligned} \tag{3.1}$$

where the sums are over the elements of δ, $\ominus$ is the inverse of the standard motion composition operator, and $trans(\cdot)$ and $rot(\cdot)$ are used to separate the translational and rotational components of the error.

3.2.2 Metric extension

As it stands, this metric is already sufficient to evaluate the performance of a SLAM algorithm in a single robot run. However, the performance of a SLAM algorithm in a certain environment may vary across multiple runs, so that computing this metric over a single run is only a rough indicator of the expected level of performance of the algorithm during the normal operation of the robot. These oscillations may be due to multiple factors, including noisy measurements, variations in the followed trajectory, and even a certain level of randomness that is inherent to the behavior of some SLAM algorithms.

For instance, consider the case of SLAM algorithms based on particle filters, like FastSLAM and GMapping. These algorithms maintain a predefined number of hypotheses about the state of the environment which are continuously updated and discarded according to the information provided by each new observation. The selection of which particles should be maintained at each update step is based on a maximum likelihood probabilistic approach, so that particles with low weights tend to be replaced as time progresses.

However, this behavior often leads to particle depletion, a phenomenon in which the variability of the hypotheses decreases to the point of compromising the ability of the algorithm to successfully perform loop-closure, i.e., to correctly identify a place in the environment that the robot has already seen. As the whole particles update and resampling process is probabilistic, there is no guarantee that two different exploration runs of the same environment will lead to the same particles being retained. Furthermore, transient errors introduced by noise in the observations may compromise the quality of the particles selected by the resampling process, leading to an unpredictable decrease of performance in an exploration run that may not be present in another.

Table 3.1: Mean and standard deviation of $\varepsilon_t(\delta)$ for 12 different exploration runs in 5 sample indoor environments (in meters).

	Building 1	Building 2	Building 3	Building 4	Building 5
$\varepsilon_t(\delta)$ mean	0.191	0.195	0.225	0.263	0.195
$\varepsilon_t(\delta)$ st.dev.	0.028	0.040	0.032	0.034	0.053

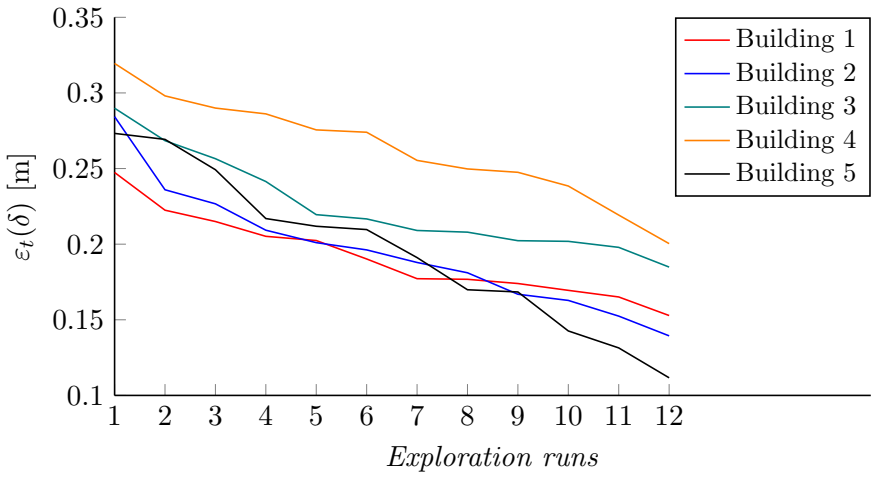


Figure 3.1: Values of $\varepsilon_t(\delta)$, in meters, for 12 different exploration runs of 5 sample indoor environments (ordered by their magnitude).

The overall conclusion is that the performance of a SLAM algorithm in an environment is subject to a certain degree of variability which cannot be captured by a single robot run. As an example, Figure 3.1 plots the values of $\varepsilon_t(\delta)$ obtained by GMapping in 5 different environments across 12 exploration runs, while Table 3.1 reports the corresponding means and standard deviations.

The variability of the measurements clearly shows that choosing the result obtained by GMapping in any individual exploration run of an environment as a measure of its expected performance in that environment would produce an approximation of the real performance that can be obtained. It is important to notice that the extent to which the aforementioned sources of uncertainty influence the exploration result may vary depending on the capabilities of the sensors, the characteristics of the environment, and the choice of the SLAM algorithm, so that it is certainly possible for some of them not to be significant in certain scenarios. As we tackle the problem from a general point of view, however, we see the necessity of using a performance metric that is able to take into account these oscillations and provide a more comprehensive picture of the range of possible behaviors a SLAM algorithm may exhibit.

To do so, we propose the following straightforward generalization of the previous definition to model the concept of *expected localization error* of a generic exploration run of an environment.

Definition 3.2. *Let p_{δ_E} be the probability of observing the δ_E set of relative pose transformations during an exploration run.*
We define the mean translational localization error *of environment E, denoted as $\mathbb{E}[\varepsilon_t(E)]$, as the expected value of the translational component of the localization error over all the possible exploration runs on E:*

$$\mathbb{E}[\varepsilon_t(E)] = \sum_{\delta_E} \varepsilon_t(\delta_E) * p_{\delta_E} \tag{3.2}$$

We therefore define the standard deviation of the translational localization error *of environment E, denoted as $\sigma[\varepsilon_t(E)]$, as:*

$$\sigma[\varepsilon_t(E)] = \sqrt{\mathbb{E}[\varepsilon_t(E)^2] - \mathbb{E}[\varepsilon_t(E)]^2} \tag{3.3}$$

Similarly, we define the mean rotational localization error *of environment E, denoted as $\mathbb{E}[\varepsilon_r(E)]$, as the expected value of the rotational component of the localization error over all the possible exploration runs on E:*

$$\mathbb{E}[\varepsilon_r(E)] = \sum_{\delta_E} \varepsilon_r(\delta_E) * p_{\delta_E} \tag{3.4}$$

We therefore define the standard deviation of the rotational localization error *of environment E, denoted as $\sigma[\varepsilon_r(E)]$, as:*

$$\sigma[\varepsilon_r(E)] = \sqrt{\mathbb{E}[\varepsilon_r(E)^2] - \mathbb{E}[\varepsilon_r(E)]^2} \tag{3.5}$$

In practice, we approximate the above quantities with their sampled versions, since the weak law of large numbers guarantees their convergence to the theoretical definitions as the number of exploration runs $|\mathcal{R}_E|$ in an environment E increases [81].

Definition 3.3. *Let $\mathcal{E}$ be a set of environments for which exploration results are available, $E \in \mathcal{E}$ be one of such environments and $\mathcal{R}_E$ the set of exploration runs performed on E.*
The sample mean and sample standard deviation of the translational localization error *of E are defined as:*

$$\overline{\varepsilon_t(E)} = \frac{\sum\limits_{R \in \mathcal{R}_E} \varepsilon_t(\delta_R)}{|\mathcal{R}_E|} \tag{3.6}$$

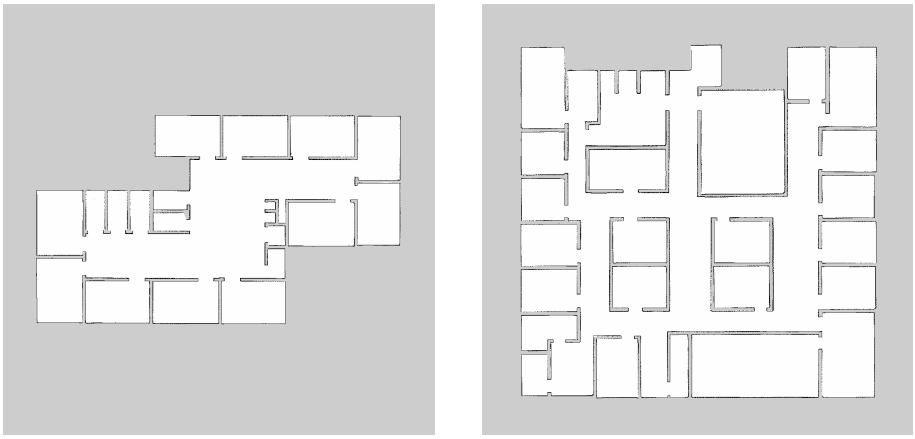
$$s(\varepsilon_t(E)) = \sqrt{\frac{\sum\limits_{R \in \mathcal{R}_E} [\varepsilon_t(\delta_R) - \overline{\varepsilon_t(E)}]^2}{|\mathcal{R}_E|}} \tag{3.7}$$

The sample mean and sample standard deviation of the rotational localization error *of E are defined as:*

$$\overline{\varepsilon_r(E)} = \frac{\sum\limits_{R \in \mathcal{R}_E} \varepsilon_r(\delta_R)}{|\mathcal{R}_E|} \tag{3.8}$$

$$s(\varepsilon_r(E)) = \sqrt{\frac{\sum\limits_{R \in \mathcal{R}_E} [\varepsilon_r(\delta_R) - \overline{\varepsilon_r(E)}]^2}{|\mathcal{R}_E|}} \tag{3.9}$$

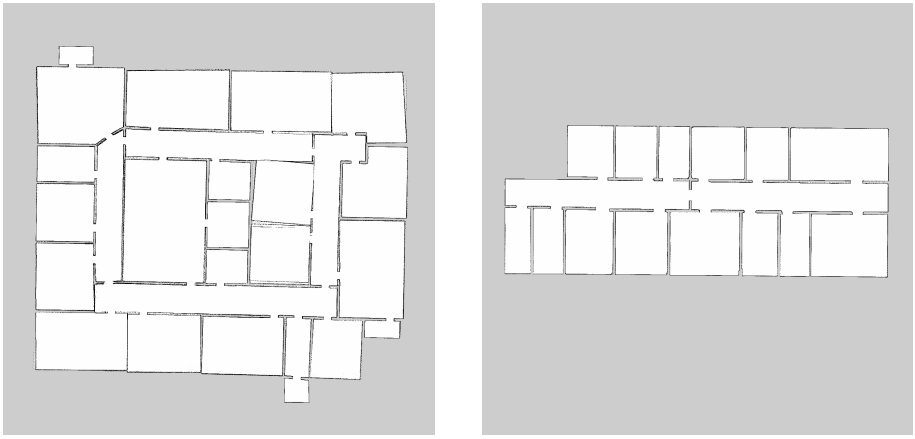
Figure 3.2 shows an example of the utility of our proposed generalization of the localization error performance metric. In environment E_a of Fig. 3.2a, GMapping has a mean translational localization error of $\overline{\varepsilon_t(E_a)} = 0.31\,\text{m}$, but the translational localization error of one of the runs is 0.43 m. In environment E_b of Fig. 3.2b, GMapping has a mean translational localization error of $\overline{\varepsilon_t(E_b)} = 0.52\,\text{m}$, but the translational localization error of one of the runs is 0.39 m. Therefore, looking only at the two single runs, one could conclude that GMapping performs better in E_b than in E_a, but, on average (and with a statistically sound number of runs), the opposite is true.



(a) E_a, $1{,}400\,\mathrm{m}^2$ *(b)* E_b, $2{,}400\,\mathrm{m}^2$

Figure 3.2: Two environments for which the performance of GMapping measured with a single run is not informative.

Finally, we want to stress the fact that the performance of a SLAM algorithm in a certain environment may also be influenced by the specs of the actual robot and of its sensors. For example, considering the environment of Figure 3.3a, the mean translational localization error made by GMapping is 0.68 m if the range of the laser range scanner is 30 m and 0.91 m if the range is 15 m; in both cases, the laser's field of view was 270° and the robot odometry error was estimated to be not greater than 0.01 m/m and 2 °/rad for the translational and rotational component respectively.



(a) The MediaCache environment. *(b) The Freiburg79 environment.*

Figure 3.3: Two environments for which changing the range of the laser range scanner significantly changes the observed localization error.

The same phenomenon can be observed for the environment of Figure 3.3b, for which the mean translational localization error made by GMapping is 0.38 m if the range of the laser range scanner is 30 m and 0.55 m if the range is 15 m. The intuitive explanation for this performance difference is that the increased laser range of the second setting allows GMapping to rely on additional information to perform scan matching and update its internal estimate of the state of the world, while simultaneously reducing the amount of distance travelled by the robot and therefore limiting the amount of error introduced by imprecise odometry readings.

Overall, these examples clearly show that the results obtained by a SLAM algorithm in a certain setting only allow for very limited conclusions on the expected level of performance of that algorithm in a different setting. In the following, we focus on the specific sub-problem of generalizing the results obtained by a SLAM algorithm in a certain environment to predict its performance in a different, and potentially unexplored, environment; however, our approach can be easily extended to encompass variations of the other aspects that define a setting, like the characteristics of the robot's sensors and the parameters that control its behavior in terms of exploration, navigation, localization, and mapping.

3.3 Problem formalization

The approach we follow to solve the problem of predicting the performance of SLAM algorithms in unexplored environments is that of supervised learning. Supervised learning is a machine learning technique that aims to infer a function from sample pairs of training data, the first element of the pair being a vector of features and the second being the desired output value. In the context of this work, the desired output value is one of the four components of the expected localization error, i.e., $\mathbb{E}[\varepsilon_t(E)]$, $\sigma[\varepsilon_t(E)]$, $\mathbb{E}[\varepsilon_r(E)]$, or $\sigma[\varepsilon_r(E)]$, depending on the specific model that is being trained.

Definition 3.4. *Let $\mathbb{E}$ be the set of possible environments in which a SLAM algorithm can be run. A* feature *is a function $f : \mathbb{E} \mapsto \mathbb{R}$ that, given any environment $E \in \mathbb{E}$, returns a real number that denotes some property of E. Let $\mathbb{F}$ be the set of all possible such functions.*

Examples of features can be the area, the perimeter or the number of walls. We discuss features more in detail in Chapter 5.

Definition 3.5. *A* model *is a function $m : \mathcal{F} \mapsto \mathbb{R}$ that, given a set of features $\mathcal{F} \subset \mathbb{F}$ evaluated over an environment, returns the predicted SLAM*

performance of the robot in that environment in terms of a component of the expected localization error. Let $\mathbb{M}$ be the set of all possible such models.

Let $\mathbb{E}[\varepsilon_t(E)]_m$ be the value of the mean translational localization error for environment E predicted by model m, and let $\sigma(\varepsilon_t(E))_m$ be the predicted value of the standard deviation of the translational localization error. Furthermore, let $\mathbb{E}[\varepsilon_r(E)]_m$ and $\sigma(\varepsilon_r(E))_m$ be the corresponding quantities of the rotational localization error.

Definition 3.6. *The* learning problem *is to find a set of subsets of features $\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4 \subset \mathbb{F}$ and a set of models $m_i : \mathcal{F}_i \mapsto \mathbb{R}$ that are solutions to the following optimization problems:*

$$
\begin{aligned}
m_1 &= \arg\min_m \sum_{E \in \mathcal{E}} [\mathbb{E}[\varepsilon_t(E)]_m - \mathbb{E}[\varepsilon_t(E)]]^2 \\
m_2 &= \arg\min_m \sum_{E \in \mathcal{E}} [\sigma(\varepsilon_t(E))_m - \sigma[\varepsilon_t(E)]]^2 \\
m_3 &= \arg\min_m \sum_{E \in \mathcal{E}} [\mathbb{E}[\varepsilon_r(E)]_m - \mathbb{E}[\varepsilon_r(E)]]^2 \\
m_4 &= \arg\min_m \sum_{E \in \mathcal{E}} [\sigma(\varepsilon_r(E))_m - \sigma[\varepsilon_r(E)]]^2
\end{aligned}
$$

It is important to highlight that this problem does not realistically admit an exact solution, because the expected localization error of a generic environment is unknown by definition. In order to actually solve the learning problem, it is therefore necessary to approximate the four components of the expected localization error, i.e., $\mathbb{E}[\varepsilon_t(E)]$, $\sigma[\varepsilon_t(E)]$, $\mathbb{E}[\varepsilon_r(E)]$, and $\sigma[\varepsilon_r(E)]$, with their sample versions, i.e., $\overline{\varepsilon_t(E)}$, $s(\varepsilon_t(E))$, $\overline{\varepsilon_r(E)}$, and $s(\varepsilon_r(E))$.

In order for the learned models to represent a meaningful approximation of reality, it is essential to gather a large amount of data concerning a wide variety of training environments. This represents a fundamental requirement to ensure that the identified solutions effectively capture the true relationship between an environment's features and observed localization error, and to limit the effect of random noise on the model. We will extensively discuss our strategy to obtain such data in Chapter 4.

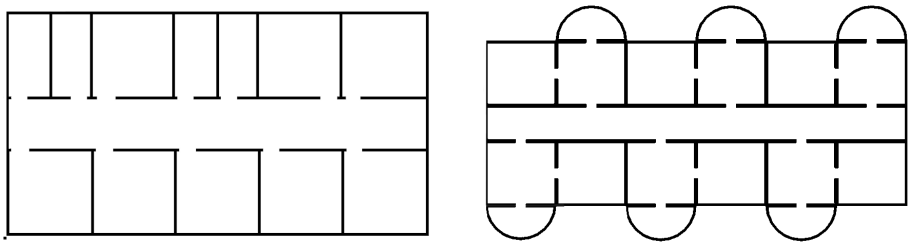
In addition, it is necessary to adopt training techniques that promote the ability of the obtained models to generalize to new samples, and to use validation techniques that are able to effectively assess such ability. We will provide an in-depth review of our strategies for model learning and model validation in Chapters 5 and 7, respectively.

3.4 Assumptions

The research community has developed over the years a wide variety of approaches to solve the SLAM problem. While the ample choice of sensors, algorithms, and parameters enriches the landscape of available solutions and broadens the range of possible applications, it also represents a significant obstacle to a comprehensive study of the field. In this work, we make some assumptions on the characteristics of the robots, of the environments, and of the data collection process.

Robots The first set of assumptions we make is about the capabilities of the robot. We assume the robot is equipped with an odometry sensor and a LIDAR sensor, a standard choice for autonomous mobile robots. We also restrict our attention to a subset of all commercially available LIDAR sensors, investigating only a limited number of range and field of view values. We also assume the robot is equipped with the necessary hardware and software stack to run the GMapping SLAM algorithm; the choice of GMapping as a representative of SLAM algorithms is dictated by its widespread usage in the mobile robotics research community, its moderate computational requirements and its robustness to noise.

Environments The second set of assumptions we make is about the characteristics of the environments. In the context of this thesis, we work exclusively with indoor, single-floor environments. We assume the environments to be sufficiently wide and uncluttered to allow the robot to explore them in their entirety. In addition, we assume all obstacles, such as walls and doors, to be opaque and made of non-transparent materials, in order to be detected by the LIDAR sensor of the robot. Moreover, we assume environments are static, i.e., they don't change during explorations, and can be represented as two-dimensional entities, i.e., $E \subset \mathbb{R}^2$. For the feature extraction methodology that leverages the work of [82] and [83] on layout reconstruction, we require the environments to follow the Manhattan world assumption, i.e., to only have orthogonal planar walls; this assumption however is not required for the feature extraction methodology based on Voronoi diagrams. Figures 3.4a and 3.4a show two environments for which the Manhattan world assumption is, respectively, satisfied and unsatisfied. Lastly, we assume to have access to an accurate representation of the environment in the form of a bitmap floor plan of known scale.



(a) A Manhattan environment. *(b) A non-Manhattan environment.*

Figure 3.4: An environment that follows the Manhattan world assumption (left) compared to one that doesn't (right).

Data collection The third set of assumptions we make is about the data collection process. In particular, we assume to be able to gather ground truth trajectory data of autonomous robotic explorations with millimeter precision, in order to accurately compute the localization error performance metric we introduced in Definition 3.3 on the environments that we use to train and validate our models. Importantly, this assumption is not required to actually use the final models to predict the performance of a SLAM algorithm in new environments, and is only necessary for model training and validation. We will further discuss the steps we take to ensure the validity of this assumption in Chapter 4.

3.5 Summary

In this chapter, we presented a formal definition of the problem addressed in this thesis. We introduced the concept of predictive benchmarking as a response to the limitations of existing SLAM evaluation techniques and we provided a mathematical characterization of the learning problem that we aim to solve. Lastly, we discussed the scope of our analysis and we listed the assumptions underlying our work.

Chapter 4

Data collection

In this chapter, we present the methodology we adopt to collect data. We start by introducing the problem of obtaining ground truth trajectory data, we discuss the methodology proposed by Kümmerle et al. in [11] and we show its limitations in terms of scalability, accuracy, and representativeness of SLAM performance. We then propose a revised approach that aims to overcome these limitations through the usage of automatized robotic simulations and we discuss its application to the execution of multiple exploration runs per environment.

4.1 The problem of ground truth trajectory data

As we mentioned in the previous chapter, the goal of this thesis is to develop a software tool that employs known features of unexplored environments to predict the performance of SLAM algorithms in those environments. To do so, we leverage the paradigm of supervised learning to determine the relationship between the value of several candidate environmental features and the corresponding observed localization error, in order to build a model that can be subsequently used for prediction. As a preliminary step, we therefore have to build a dataset of explored environments for which the localization error is known and for which the environmental features can be extracted.

In Section 3.2.1, we formally defined the localization error of a δ_E run of an environment E, $\varepsilon(\delta_E)$, as the deformation energy that is required to transfer the SLAM estimate of the robot's trajectory in that run onto the corresponding ground truth trajectory. This computation requires knowledge of both the estimate of the trajectory produced by the SLAM algorithm and of the exact trajectory followed by the robot in the run, in order

to compute the $\delta_{i,j}$ and $\delta_{i,j}*$ relative transformations that constitute the basis of the performance metric.

In general, the problem of obtaining ground truth trajectory data is not trivial. The exact knowledge of the path followed by a robot during an exploration run is obtained by accurately tracking its movements and pinpointing its pose in the environment at every time step, a task that is usually performed by SLAM algorithms. However, for SLAM evaluation to be meaningful, ground truth positioning data has to come from a different and more reliable source of information, significantly more accurate than the pose estimates obtained from any SLAM algorithm. This usually involves the usage of complex and expensive tracking technologies that need to cover the whole environment, thus requiring a setting that can be seldom deployed in real world scenarios.

To overcome this problem, Kümmerle et al. propose in [11] a methodology that leverages background human knowledge of the explored environments to manually determine the set $\delta^*_{i,j}$ of reference relative transformations, or *relations*, between pairs of poses to be used as ground truth. In this approach, the robot is equipped with a laser range scanner and records a laser scan of the environment at every time step. After the exploration, a human operator analyzes the scans to determine which observations in the dataset cover the same part of the space and manually aligns them until they match; the amount of displacement required for the alignment is then stored as the relative transformation between the two. This process only enforces local consistency of the measurements; to enforce global consistency, relative transformations between scans that don't cover the same area of the environment must also be added. However, as the latter refer to different areas of the environment, their displacement cannot be determined through scan matching; in this case, the human operator has to rely on external sources of information to obtain additional data, such as the exact distance between pairs of reference objects in the two scans.

This approach has the advantage of not requiring any special hardware setup to obtain ground truth trajectory data, which makes it potentially applicable to a variety of environments. However, it suffers from three major drawbacks that severely limit its applicability.

The first major drawback is its limited scalability. As it requires human intervention to manually align the laser scans and obtain the reference relative transformations, this method can't efficiently scale as the number of laser scans increases. This effectively prevents its employment in many real-world application scenarios, imposing a compromise between the covered area of the environment and the temporal resolution of the scans. This

issue is even more evident in the case of multiple explorations of the same environment.

The second major drawback is its limited ability to portray the actual localization accuracy of a SLAM algorithm. Relations obtained through scan matching can be used to enforce local consistency, but often result in a significant underestimation of the true localization error of the SLAM algorithm, as scans that refer to the same area of the environment are likely to have been taken at short temporal distance from one another and consequently to show little to no error. The alignment of scans taken at loop closures represents a better choice, but their presence and quantity is largely dependent on the environment's structure and on the followed path. A more realistic estimate of the localization error can be obtained by complementing the relations obtained through scan matching with additional relations derived from background knowledge of the environment to account for errors associated to pairs of poses that are very far from each other; however, the information required for this kind of corrections is seldom available with the necessary accuracy to represent a valid source of additional ground truth data. Therefore, this methodology often underestimates the actual localization error of the tested algorithms and provides optimistically biased estimates of their accuracy.

The third and perhaps most significant drawback of this approach is its level of arbitrariness in the choice of the relative transformations to be used as ground truth relations. This selection, which is left entirely to the human operator's discretion, may have a significant impact on the computation of the metric and consequently undercut the validity and replicability of the evaluation results. This issue is particularly evident when considering the usage of additional relations to enforce global consistency, as their presence, accuracy, and amount can dramatically alter the computed localization error.

4.2 Proposed methodology

The approach proposed by Kümmerle et al. is an attempt at overcoming the significant challenges related to the collection of ground truth trajectory data in real world generic application environments, where accurate continuous robot localization depends on the availability of tracking equipment covering the entire area of interest.

However, the research community has developed over the years a number of simulation approaches that can be used to reproduce mobile robots' behavior with high fidelity without requiring the physical interaction with

an environment [8, 69, 70]. These techniques offer the ability to exactly and continuously track a robot's pose throughout the entire duration of a simulation run without requiring any special equipment, and can thus be the foundation for the development of more accurate, objective, and scalable approaches to SLAM evaluation.

We therefore propose an approach that uses automatized robotic simulations to simultaneously collect both the SLAM data and the ground truth trajectory data that are necessary for the computation of the localization error performance metric. Our method has several advantages over the methodology proposed by Kümmerle et al. , including its complete independence from human interaction, the possibility to collect data in a fully automatized way, and a significantly higher level of representativeness of SLAM performance.

4.2.1 Automatized exploration

As we mentioned in Section 3.3, collecting a large amount of data concerning a wide variety of training environments is a fundamental requirement to ensure that the solution to the learning problem effectively captures the true relationship between an environment's features and observed localization error. As a consequence, the efficiency of the data collection process is paramount to ensure the actual applicability of our methodology.

To solve this issue, we propose a method that uses automatized robotic simulations in place of real-world experiments to perform the autonomous exploration of environments. The usage of simulations allows us to benefit from three major advantages that vastly simplify the data collection process.

First, simulations are significantly cheaper and more efficient than real-world experiments, as they don't require the physical availability of properly equipped testing environments. This leads to important savings in terms of both time and money, and allows us to conduct experiments on a much larger set of environments that what would be possible otherwise.

Second, simulations can be fully automatized without the need for a human researcher to constantly pay attention to the robot to ensure the safety of its surroundings. This allows the method to efficiently scale as the availability of computational resources increases, further contributing to the overall efficiency of the data collection process.

Third, simulations can be finely tuned to reproduce a specific robot configuration in terms of sensor capabilities and odometry accuracy, potentially enabling the investigation of a wide variety of different experimental scenarios and therefore improving the representativeness of the obtained results.

Among the main parameters that can be modified, the most important ones are certainly the range and field of view of the laser range scanner, as well as the amount of translational and rotational error affecting the odometry readings of the virtual robot. A much more exhaustive list of the available settings is reported in Appendix A.

According to our proposal, simulated exploration runs are conducted as follows.

Starting from an initial conventional pose, the virtual robot explores its surroundings by moving towards a series of points in the environment called *goals*. Goals are proposed individually in sequential order; whenever a goal is reached, a new one is proposed, in an iterative process that lasts until the end of the exploration.

Goals are computed according to the frontier-based exploration paradigm defined by Yamauchi in [28], where each frontier represents a region on the boundary between free known space and unexplored space; an area of the space is considered to be *explored* once it becomes part of the map produced by the SLAM algorithm. To make the exploration process more robust, points that are located at very short distance from each other are clustered in a single frontier.

In order to select which frontier should be explored after a goal has been reached, frontiers are ordered according to their Euclidean distance from the current pose of the robot; the goal is then chosen on the frontier that is nearest to the robot, with ties broken randomly.

To check if an exploration is *complete*, our approach periodically takes snapshots of the map produced by the SLAM algorithm at regular time intervals and compares every new snapshot to its immediate predecessor in the series. The comparison is done using the mean square error (MSE) metric, which is computed by averaging the squared intensity difference of each pixel in the map between the two snapshots. The exploration is stopped once the degree of similarity between the two snapshots reaches a certain predefined threshold. Once an exploration is complete, we compute its associated $\varepsilon_t(\delta_E)$ and $\varepsilon_r(\delta_E)$ and we use the latest snapshot as the final reconstructed map of the environment.

As we mentioned, the localization error of an individual exploration run is often subject to oscillations that limit its reliability as an estimator of SLAM performance. To compensate for this limitation, in Definition 3.3 we introduced the idea of computing the mean and standard deviation of the localization error of multiple exploration runs as a way to assess the expected performance of a SLAM algorithm during a generic exploration of

an environment.

We therefore repeat the described exploration process until the number of runs $|\mathcal{R}_E|$ performed on environment E reaches a critical threshold, whose value is determined as a function of the desired accuracy of the estimation, as we will discuss more in depth in Section 4.2.2. The different components of the expected localization error of that environment, i.e., $\overline{\varepsilon_t(E)}$, $s(\varepsilon_t(E))$, $\overline{\varepsilon_r(E)}$, and $s(\varepsilon_r(E))$, are then computed according to Equation 3.6, 3.7, 3.8 and 3.9 respectively.

4.2.2 Estimation of the number of runs

In order to actually enact the procedure we described, it is first necessary to establish a criterion to identify the $|\mathcal{R}_E|$ number of runs that should be performed for each environment E. In the following, we propose an approach to compute the size of such sample of runs depending on the desired level of confidence for the estimate of the localization error, under the hypothesis that two assumptions on the distribution of the sample mean are verified.

Our first assumption requires that the covariance of the localization error on different runs of the same environment tends to be zero on average. In principle, we cannot assume different explorations of a same environment to be completely independent from each other; this is because, despite them being conducted separately and with no mutual interference, they might be performed according to the same exploration criteria and thus have similar trajectories. However, Chebyshev's weak law of large numbers for potentially correlated sequences guarantees the convergence of the sample mean of the localization error $\overline{\varepsilon(E)}$ to the true mean $\mathbb{E}[\varepsilon(E)]$ under the weaker condition that the average of the covariances between all possible pairs of measurements of $\varepsilon(\delta_E)$ in the sequence tends to zero as the number of measurements increases [81]. In practice, the large amount of external factors that influence a robot's trajectory throughout an exploration run, including noisy measurements and tie-breaking mechanisms in the frontier selection process, provides a reasonable assurance that this assumption is verified.

Our second assumption is about the distribution of the sample mean, which we assume to be at least approximately normal. In principle, the lack of formal guarantees on the independence of different exploration runs implies that the central limit theorem is not guaranteed to hold, preventing us from drawing formal conclusions about the normality of the distribution of the sample mean. For the purpose of determining a viable number of sample exploration runs, however, we assume the aforementioned external factors to be sufficient for the normality condition to be at least approximately

verified.

Under these assumptions, we therefore proceed with the estimation of the number of runs to be performed for each environment using the following formula:

$$|\mathcal{R}_E| = \frac{z_{\alpha/2} * s^2}{d^2} \tag{4.1}$$

where E is the environment, s^2 is the unbiased estimator of the population variance of $\varepsilon(\delta_E)$ across multiple runs of E, d is the margin of error, α is the complement of the desired confidence level and $z_{\alpha/2}$ is its associated z-score.

It should be noted that, for Equation 4.1 to be applicable, we must first obtain an estimate of the variance of $\varepsilon(\delta_E)$ across multiple runs of E. To solve this issue, we rely on the widely used statistical practice of pilot research to obtain an initial estimate of the variance on a small sample of 10 explorations. This value is then used to compute an initial estimate of the number of required explorations $|\mathcal{R}_E|$ using Equation 4.1; afterwards, the additional exploration runs that are needed to reach $|\mathcal{R}_E|$ are performed and the newly obtained data are used to compute a new estimate of $|\mathcal{R}_E|$, iteratively repeating the process until the newly estimated sample size is smaller than or equal to the number of already performed explorations.

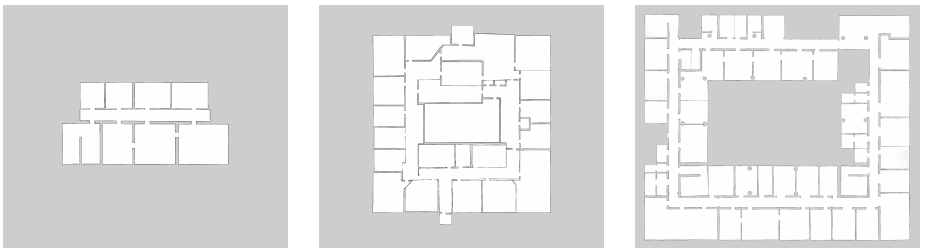
It is also important to highlight that the sample sizes required for an accurate estimation of the translational and rotational components of the localization error may differ, as the variance of $\varepsilon_t(\delta_E)$ may be different from the variance of $\varepsilon_r(\delta_E)$. Whenever this happens, the selected sample size is the maximum of the two.

4.2.3 Relations sampling

Theoretically, the availability of accurate ground truth trajectory data guaranteed by the usage of simulation tools enables the straightforward computation of both $\varepsilon_t(\delta)$ and $\varepsilon_r(\delta)$ by means of Equation 3.1 immediately after the conclusion of an exploration run. However, a direct application of such equation poses significant practical problems because of its computational complexity, which is quadratic in the number of poses on the robot's trajectory and thus makes the metric increasingly difficult to compute as the size of the environment grows.

To help framing the problem, it is useful to examine a few examples. Consider the case of an autonomous robot exploring indoor environments at speeds between 0.2 m/s and 0.4 m/s, which is a typical safety range to ensure adequate human protection. We examine the average time required

for the complete autonomous exploration of three office environments of increasing size, each explored 15 times under identical circumstances, using the data collection methodology we introduced in Section 4.2.1. In particular, we take a snapshot of the reconstructed map every 10 min and we stop the exploration when the mean square error of the latest snapshot with respect to its immediate predecessor is below a threshold of 10. To be able to represent small environments as well as large environments, we set the size of the reconstructed map to be 200 m × 200 m. We also assume a temporal resolution of 10 poses per second in order to have reasonably fine-grained measurements. A detailed report of the measured exploration times is shown in Table 4.1.



(a) A small-sized office *(b) A medium-sized office* *(c) A large-sized office*

Figure 4.1: A comparison between the floor plans of three offices of different sizes.

The first office, depicted in Figure 4.1a, consists of 9 rooms and a corridor covering an area of approximately 1,500 m^2. In this case, the simulations show that the average time required for its complete exploration is 25 min; with a resolution of 10 poses per second, there are 15,600 poses in the robot's trajectory, which result in over 243 million relations.

The second office, depicted in Figure 4.1b, is relatively larger than the first one, consisting of about 30 spaces among rooms and corridors covering an area of approximately 4,100 m^2. In this case, the average time required for its complete exploration is 75 min, resulting in 45,000 poses in the robot's trajectory and about 2 billion relations.

The third office, depicted in Figure 4.1c, is the largest of the three, comprising over 50 spaces among rooms and corridors covering an area of approximately 6,500 m^2. The average time required for its complete exploration is 102 min, which means there are 61,200 poses in the robot's trajectory, for a grand total of over 3.7 billion relations.

These examples clearly illustrate a significant scalability issue that prevents the direct application of Equation 3.1 as a measure of SLAM performance. To overcome this problem, we propose to restrict its application to a

Table 4.1: Exploration times of three different datasets over 15 runs.

	Time required for a complete autonomous exploration of the dataset (in minutes)															
Dataset	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12	r13	r14	r15	avg
Office 4.1a	30	40	20	30	20	20	20	30	30	20	30	30	20	20	30	26
Office 4.1b	100	90	50	70	80	70	70	100	80	100	50	50	70	70	80	75
Office 4.1c	90	90	90	140	110	70	80	100	140	100	80	130	120	100	90	102

limited subset of randomly sampled relations, the size of which is determined as a compromise between sampling accuracy and computational complexity.

The procedure is conceptually similar to the approach we described in Section 4.2.2 for the estimation of the number of sample runs for each environment, and is based on the usage of the central limit theorem to approximate the relations' sampling distribution with a normal distribution [84].

At first, we set the confidence level and the margin of error of the estimation. Then, we apply the following formula to determine the number of relations to be used to estimate the localization error:

$$n = \frac{z_{\alpha/2} * s^2}{d^2} \tag{4.2}$$

where s^2 is the unbiased estimator of the variance of the population of the errors $\varepsilon(\delta_{i,j})$ made by the SLAM algorithm on the individual relative transformations $\delta_{i,j}$ for the considered exploration run, d is the margin of error, α is the complement of the desired confidence level and $z_{\alpha/2}$ is its associated z-score.

In order to ensure the applicability of the central limit theorem, we must assume relative transformations to be independent and identically distributed random variables. In principle, this may not be the case for every possible pair of relations, as transformations involving pairs of poses that are close to each other will inevitably be similar and not independent; however, the number of possible relations is so large that, given any two random relations, the likelihood of them being dependent is negligible for all practical purposes. Furthermore, as the process used for data collection is the same for all poses, it is reasonable to assume the identical distribution property to hold as well.

To validate our approach, we verified the empirical soundness of these assumptions on a representative set of environments. Figure 4.2 shows the sample distribution of $\varepsilon_t(\delta_E)$ for an exploration run of two of these environments; the distributions were obtained by repeatedly applying the sampling procedure to extract 200 different samples of relations imposing a 99 % confidence level and a margin of error of $\pm\,0.02\,\mathrm{m}$. As it can be seen, the shape of the distributions is approximately normal.

Also in this case, for Equation 4.2 to be applicable, we must first obtain an estimate of the variance of $\varepsilon(\delta_{i,j})$, which in turn has to be computed on a sample for performance reasons. We rely again on the statistical practice of pilot research to compute an initial estimate of the population variance from the sample variance of a different set of 500 randomly selected relations.

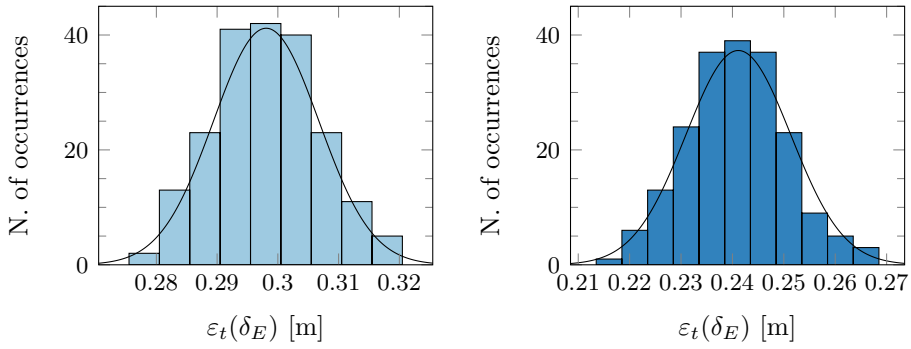


Figure 4.2: Distribution of the translational localization error of an exploration run of two test environments; the shape tends towards a normal distribution as the number of samples increases, in accordance with the central limit theorem.

Similarly to what observed for the estimation of the number of runs, the sample size required for an accurate estimation of the translational and rotational components of the localization error may differ, as the variance of the translational component of $\varepsilon(\delta_{i,j})$ may be different from that of its rotational component. Whenever this happens, the selected sample size is the maximum of the two.

4.3 Summary

In this chapter, we presented the methodology we adopt to perform data collection. We first analyzed the proposal made by Kümmerle et al. in [11] to obtain ground truth trajectory data in generic environments, highlighting its limitations in terms of scalability, accuracy, and representativeness of SLAM performance. We subsequently introduced a modified approach that overcomes these shortcomings by leveraging the possibility to obtain highly precise ground truth data through the usage of automatized robotic simulations. Compared to Kümmerle's approach, our proposal has the main advantage of providing a fully automated, objective, and scalable methodology for data collection.

Chapter 5

Proposed solution

In this chapter, we complement the content of Chapter 4 to provide an in-depth explanation of our solution to the problem we outlined in Chapter 3. We start by presenting a logical high-level overview of the different modules of which the solution is composed, after which we delve into a more extensive discussion of the functionalities covered by each component.

5.1 General overview

In order to minimize code duplication and enhance flexibility and extendability, our solution consists of a chain of several modules, each responsible for a logically distinct sub-task, as shown in Figure 5.1:

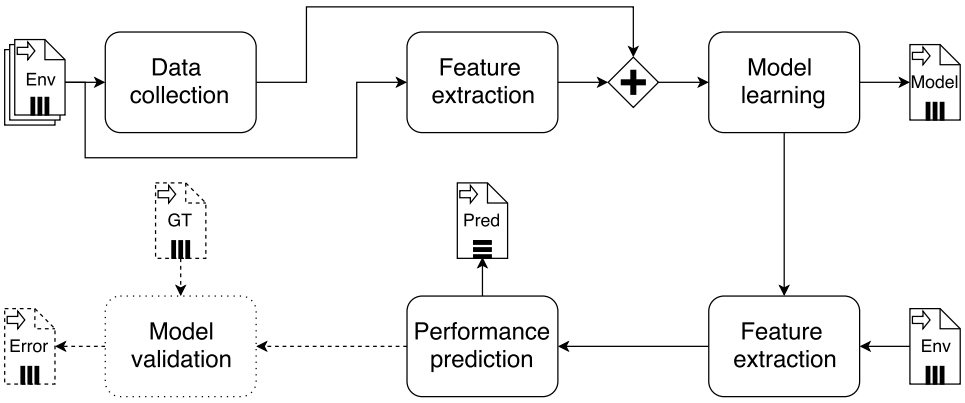


Figure 5.1: A BPMN diagram of the control flow of our solution. Dashed lines represent optional components.

- **Data collection**: this module is responsible for the collection of the true localization error data that are used to train and test the per-

formance prediction models we aim to obtain. Its design has already been extensively covered in Chapter 4.

- **Feature extraction**: this module computes a number of characterizing features of the environments starting from the analysis of their floor plans. It is itself composed of three sub-modules, each devoted to the extraction of a particular type of feature: geometrical, topological, and based on Voronoi graphs.

- **Model learning**: this module operates on a set of training environments for which the localization error is known. Its purpose is to use machine learning regression techniques to analyze the correlation between the values of the environmental features of the training environments, as computed by the feature extraction module, and the corresponding values of the localization error obtained by the data collection module. It consists of three separate sub-modules, each implementing a different machine learning approach: a simple linear regression model, a multiple linear regression model with explicit feature selection, and a regularized ElasticNet regression model with implicit feature selection. Each trained model correlates a specific subset of environmental features to a single component of the localization error.

- **Performance prediction**: this module predicts the localization error of a SLAM algorithm on an environment. It relies on the models built by the model learning module, automatically selecting the most appropriate model for the component of the localization error that the user is interested in predicting; the features of the selected model are evaluated on the input environment by the feature extraction module. It also allows the user to select a prediction model of choice regardless of its relative performance with respect to other models, thus offering the possibility to choose a compromise between model complexity and prediction accuracy.

- **Model validation**: this module verifies the experimental accuracy of the models built by the model learning module, comparing the predicted localization error of a set of testing environments against their known true localization error. Its purpose is therefore purely diagnostic and its presence is not required for the normal operation of our methodology. We will discuss the evaluation strategy used by this module in greater detail in Chapter 7.

5.2 Feature extraction

In Chapter 3, we formally defined the concept of *feature* as a function that, given an environment E, returns a real number that denotes some property of E. In this section, we provide an in-depth review of the environmental features we decided to implement in the feature extraction module of our solution.

It is important to highlight that these features only rely on the availability of a bitmap representation of the floor plan of an environment in order to be computed, and therefore can be used to characterize both the environments used for training and the environments whose localization error we want to predict. This property is fundamental, as the independence of the environmental features from the actual exploration data is the premise to allow the prediction of SLAM performance on environments for which such data are not available.

For clarity, we split the discussion in three subsections, according to the kind of environmental property each feature is based upon: geometrical, topological, or derived from Voronoi graphs.

5.2.1 Geometrical features

The first set of features we propose is based on the analysis of geometrical properties of environments. In this context, by *geometrical properties* we refer to any kind of property related to the shape or extension of the building or a part thereof.

A preliminary step towards the computation of such properties is the extraction of a clean geometrical model of the environment from a bitmap representation of its floor plan. For this purpose, we rely on the algorithm described in [82] and in [83] to perform layout reconstruction. For the sake of completeness, we hereby provide a brief overview of the main steps of the algorithm.

The method applies the *Canny edge detection* algorithm to identify the edges in the image, which are then processed by the *Hough line transform* algorithm to detect lines. The Canny edge detection algorithm uses a Sobel kernel to identify potential edges based on the intensity of the edge gradient of each pixel, defined as the square root of the sum of the squares of the first derivatives of the Sobel kernel in the horizontal and vertical direction. Afterwards, the Hough line transform algorithm takes the floor plan image containing the edges identified by the Canny edge detection algorithm and uses a probabilistic approach to detect lines. The identified lines are

then clustered together with the mean shift operator into walls, which are then used to identify the regions of the image that may belong to the same delimited space of the environment, like a room or a corridor. Finally, the identified regions are merged together to form spaces using the DBSCAN [85] clustering algorithm. Figure 5.2 shows an example of layout reconstruction for a sample environment.

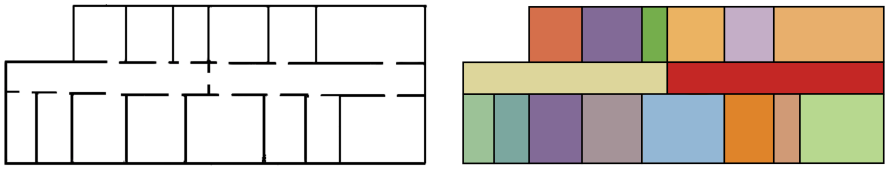


Figure 5.2: On the left, the ground truth floor plan of the Freiburg 79 environment. On the right, its reconstructed layout, with the delimited spaces highlighted in different colors.

We use the results of this analysis to compute two distinct sets of geometrical features: one based on global properties of the environment, the other based on local properties of the individual spaces. The global properties we consider are the area and the perimeter of the building, as well as the ratio between its shortest and its longest wall, the overall number of identified spaces and the total sum of their perimeters. With the exception of the last two features, the local properties of the individual spaces are identical, but are averaged to be treated as features of the environment.

5.2.2 Topological features

The second set of features we propose is based on the analysis of properties of topological graphs. In this context, by *topological graph* we mean a graph structure whose nodes correspond to delimited spaces of the environment (e.g., rooms and corridors) and whose edges indicate the existence of a direct connection between them (e.g., doors and passages). Figure 5.3 shows an example of topological graph for a sample environment.

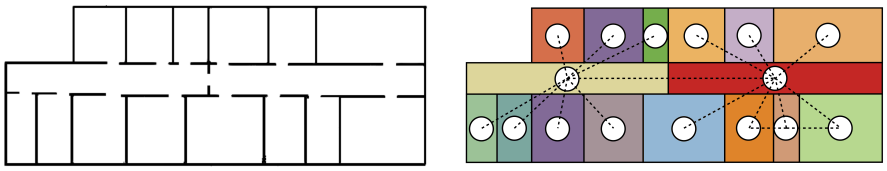


Figure 5.3: On the left, the ground truth floor plan of the Freiburg 79 environment. On the right, its associated topological graph.

Conceptually, a topological graph can be seen as a kind of skeletonized representation of the structure of an environment in which the specific geometrical properties of the spaces are ignored in favour of a more abstract representation of the relationships between them.

The computation of the topological graph of a floor plan is based on an extension of the layout reconstruction algorithm we described in the previous subsection. We provide a pseudocode of this extension in Algorithm 1.

Algorithm 1 Create topological graph

```
function createGraph(img, wallEnlargement)
    borderingSpaces, connectedSpaces = []
    spaces = reconstructLayout(img)
    for s in spaces do
        s.enlargeWalls(wallsEnlargement)
    for sA in spaces do
        for wA in sA.enlargedWalls do
            for sB in spaces do
                for wB in sB.enlargedWalls do
                    if wA.intersects(wB) then
                        borderingSpaces.append(sA, sB)
    for sA,sB in borderingSpaces do
        if areSpacesConnected(img,sA,sB) then
            connectedSpaces.append(sA, sB)
    G = createGraphFromConnections(connectedSpaces)
    return G
```

To build the graph, we first verify whether any two spaces are *bordering*, i.e., they have at least a wall in common. To minimize the risk of false negatives, we perform a preprocessing step that consists of enlarging the walls of each space in order to make them thicker.

However, physical closeness represents a necessary but not sufficient condition for connectivity: in order for a space to be directly reachable from another, it is also necessary that the two spaces are connected via a door or some other kind of passage.

To verify this condition, we propose a technique called *selective floodfill*, whose pseudocode is described in Algorithm 2. This method is based on the assumption that if two spaces are directly connected to each other via a passage, then it must be possible to perform a *floodfill* operation from any point of the first space and detect its effect on any point of the second space.

To select the two sample points for the analysis, we rely on an algorithm that iteratively manipulates the centroid of each space to find a representative point that is guaranteed to fall within the space[1].

Algorithm 2 Selective floodfill

```
function isolateSpaces(img, spaceA, spaceB)
    imgA = cutSpaceFromImage(img, spaceA.corners)
    imgB = cutSpaceFromImage(img, spaceB.corners)
    cutout = bitwiseOR(imgA, imgB)
    return cutout

function areSpacesConnected(img, spaceA, spaceB)
    cutout = isolateSpaces(spaceA, spaceB)
    reprPointA = getFreeReprPoint(img, spaceA)
    reprPointB = getFreeReprPoint(img, spaceB)
    floodImg = floodfill(cutout, reprPointA, 127)
    if cutout[reprPointB] == 127 then
        return true
    return false
```

The algorithm then iteratively verifies the connectivity condition for all pairs of bordering spaces and proceeds with the construction of the topological graph.

In the following, we review the graph metrics that we implemented for the analysis of topological graphs, dividing them according to the aspect of the graph they try to capture. A detailed explanation of these metrics can be found in [86].

5.2.2.1 Size measures

By *size measures* we refer to a set of graph metrics that are intended to capture the extension of a graph from different points of view. In particular, we discuss the concepts of *order*, *size*, *density*, *diameter* and *radius* of a graph, as well as the notions of *bifurcation point* and *terminal point*.

A first kind of measures of this type is based on counting the number of nodes or edges that compose the graph; this is the case respectively of a graph's *order* and *size*.

From these quantities, it is possible to define the idea of graph *density*, which is a measure of the degree of connectivity of a graph with respect to

[1] `http://toblerity.org/shapely/manual.html#object.representative_point`

its overall number of nodes. In this context, we are interested in providing a definition of graph density only for undirected graphs, i.e., for graphs where the existence of an edge between two nodes implies the possibility of moving freely from one node to the other and viceversa; this assumption stems from the consideration that, in the large majority of buildings, doors and passages are two-way connections between spaces. Under this assumption, the density of a graph is defined as:

$$d = \frac{2m}{n(n-1)}$$

where m is the number of edges and n is the number of nodes of the graph.

In addition to these metrics, we also consider the concept of *eccentricity* of a node, that is the maximum distance from that node to all other nodes of the graph. The *diameter* and *radius* of a graph are respectively defined as the maximum and the minimum eccentricity among all nodes of the graph.

Finally, we also measure the number of bifurcation points and terminal points of the graph. A *bifurcation point* is a node that has more than two neighbors; a *terminal point* is a node that has exactly one neighbor.

5.2.2.2 Centrality measures

In graph theory and network analysis, indicators of centrality identify the most important nodes within a graph. Each measure of centrality provides its own criterion to assess the importance of a node, depending on the specific aspect of the network it wants to capture, and its own normalization factor to obtain values that can be compared across different graphs.

In order to obtain metrics that can be computed on the graph as a whole, for each of these forms of node centrality we consider its mean and standard deviation among all nodes as a measure of centrality of the graph itself.

Betweenness centrality

The *betweenness centrality* of a node v is the sum of the fraction of all-pairs shortest paths that pass through v:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where V is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t)-paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v other than s,t. If $s = t$, $\sigma(s,t) = 1$, and if $v \in s,t$, then $\sigma(s,t|v) = 0$. In practice, it is a measure of how frequently a certain node appears on the shortest path

between any two other nodes of the graph: the more frequently it appears, the more important the node is for the connectivity of the network as a whole.

Closeness centrality

A related concept is that of *closeness centrality* of a node u, which is defined as the reciprocal of the sum of the shortest path distances from u to all the other $n-1$ nodes. Since the sum depends on the number of nodes in the graph, the measure is usually normalized by the sum of minimum possible distances; as the minimum distance between a node and every other node is 1, the correction factor amounts to $n-1$. The formula for its computation is therefore:

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$$

where $d(v,u)$ is the shortest-path distance between v and u, and n is the number of nodes in the graph. The intuition behind this metric is to give greater importance to nodes that lie at very short distance from all the other nodes.

Eigenvector centrality

Another approach to node centrality is that of *eigenvector centrality*. The eigenvector centrality of a node is a measure of its influence in a network, under the assumption that connections to high-scoring nodes contribute more to the score of the node in question than connections to low-scoring nodes. The scalar definition of this metric is the following:

$$x_v = \frac{1}{\lambda} \sum_{t \in M(v)} x_t = \frac{1}{\lambda} \sum_{t \in G} a_{v,t} x_t$$

where x_v is the eigenvector centrality of node v, G is a graph, $M(v)$ is the set of neighbors of v, λ is a constant, and $A = (a_{v,t})$ is the adjacency matrix of graph G, i.e., $a_{v,t} = 1$ if node v is linked to node t and $a_{v,t} = 0$ otherwise. However, because of the mutual dependence between the relative scores of neighboring nodes, this metric is often computed by taking advantage of the following equivalent vectorial definition:

$$Ax = \lambda x$$

where x is the eigenvector of centralities for all nodes of the network, i.e., the v^{th} component of x is the relative centrality score of node v in the

network. In general, there may be different eigenvalues λ for which a non-zero eigenvector solution exists; however, since eigenvector centrality is required to be positive, only the greatest eigenvalue results in the desired centrality measure (by the Perron-Frobenius theorem).

Katz centrality

Finally, the last kind of node centrality we consider in this thesis is *Katz centrality*. Katz centrality represents a variant of eigenvector centrality that is computed according to the following formula:

$$x_v = \gamma \sum_{t \in G} a_{v,t} x_t + \beta$$

where G is a graph, $A = (a_{v,t})$ is the adjacency matrix of graph G, $\gamma \in (0, 1)$ is an attenuation factor such that $\gamma < \frac{1}{\lambda_{max}}$, λ is the vector of eigenvalues of A, λ_{max} is the largest such eigenvalue and β is a parameter that controls the initial centrality of the nodes (in this thesis, it is set to 1).

5.2.3 Voronoi features

The third and last set of features we propose is based on the analysis of properties of Voronoi graphs.

In Chapter 2, we introduced the concept of Voronoi diagram as a partition of the plane into cells, where each cell encloses a site, i.e., a point of the plane that represents an obstacle, and contains all points of the plane whose distance to the site is not greater than their distance to all other sites. A *Voronoi graph* is a graph structure that is obtained by considering the boundaries of the cells of a Voronoi diagram as edges and the points of intersection between them as nodes.

In the following, we discuss a methodology for the computation of Voronoi graphs of environments from their floor plans and we present a number of features based on properties of Voronoi graphs for the estimation of the localization error.

5.2.3.1 Voronoi graph computation

The first step towards the identification of environmental features based on properties of Voronoi graphs is developing a methodology to compute the Voronoi graph associated to a floor plan.

To do so, we first perform a Voronoi tesselation of the floor plan by exploiting the mathematical property that states that the Voronoi diagram of

a set of points is dual to its Delaunay triangulation. A Delaunay triangulation for a given set P of discrete points in a plane is a triangulation, i.e., a subdivision of the plane into triangles $DT(P)$, such that no point in P is inside the circumcircle of any triangle in $DT(P)$. Several methods have been proposed to obtain the Delaunay triangulation of an image; in this thesis, we rely on the algorithm[2] proposed by Bowyer and Watson respectively in [87] and in [88]. The result of this operation is a tessellation of the floor plan image into Voronoi cells, whose boundaries form a graph-like structure that serves as the basis for the construction of the Voronoi graph. We call this structure *bitmap Voronoi graph*.

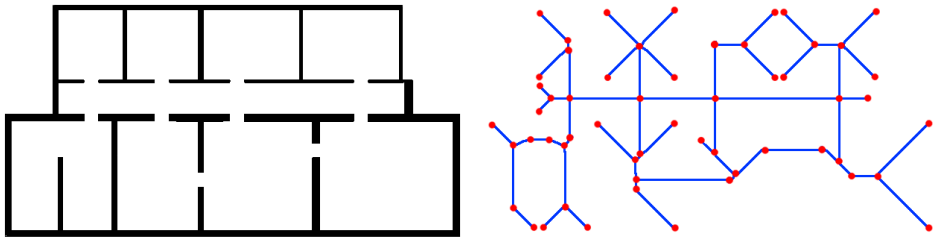


Figure 5.4: On the left, the image of a floor plan used as input for the Voronoi graph computation algorithm. On the right, a visual representation of the computed Voronoi graph (nodes in red, edges in blue).

Since the images we consider represent floor plans of buildings, it is always possible to clearly divide their content in two distinct zones: the inside and the outside of the building. For the purpose of predicting SLAM performance, we are only interested in obtaining a skeletonization for the inner part of a building, as it is the one that the robot will explore.

For this reason, we first identify all the contours in the image using the algorithm proposed by Suzuki et al. in [89]; among them, we select the longest one as the contour of the polygon that represents the perimeter of the building. We therefore check each pixel of the bitmap Voronoi graph and we retain only those pixels that fall inside the polygon.

At this point, we proceed with the conversion of the bitmap Voronoi graph into a proper graph structure.

To do so, we first apply an image dilation operator to the bitmap Voronoi graph using a 5×5 pixels kernel; afterwards, we reduce the resulting image to a 1 pixel wide representation using the skeletonization operator defined by Zhang et al. in [90]. These two steps are necessary to obtain a representation of the bitmap Voronoi graph that is as clean and uniform as possible and

[2] `https://en.wikipedia.org/wiki/BowyerWatson_algorithm`

to remove any imperfection and irregularity that may have been introduced by the tesselation algorithm.

We therefore consider each pixel as a node of the Voronoi graph, adding an edge of unitary weight between two nodes if their corresponding pixels are *close* to each other, i.e., if one sits within the 3×3 pixel area centered around the other. To reduce the size of the graph and obtain something more manageable, we then perform a graph sparsification that removes all *pass-through* nodes, i.e, nodes that have exactly two neighbors and whose corresponding pixels belong to a sufficiently long straight line; the neighbors are then connected via a new edge whose weight is the sum of the weights of the original edges it's replacing.

Figure 5.4 shows a visual representation of the Voronoi graph of the Freiburg 52 environment after the sparsification step, with nodes in red and edges in blue, together with its corresponding ground truth floor plan.

5.2.3.2 Feature computation

Besides evaluating on the Voronoi graph the same metrics that we have already introduced for the analysis of the topological graph, we propose to exploit the unique properties of Voronoi graphs to define two additional features to better capture a robot's behavior during the exploration of an environment.

A notable property of Voronoi graphs that is particularly useful in autonomous robotics is that their edges lie on the path that is furthest from all obstacles on the plane. This property has significant implications for path planning; in fact, the edges of the graph constitute the clearest path that a robot could follow to move from a place of the environment to another.

Although the actual path followed by a robot strongly depends on the exact path planning algorithm that it runs, we argue that it is possible to obtain a good approximation of such path by performing a special kind of graph traversal of the Voronoi graph that takes into account the limitations of the robot sensors and the physical characteristics of the environment that the graph refers to.

Our proposal is therefore to use the Voronoi graph of the environment to obtain an approximation of the path the robot would follow to perform a complete exploration of that environment. In particular, we define two quantities, called *Voronoi traversal distance* and *Voronoi traversal rotation*, that respectively represent the overall distance and the overall amount of rotation that a robot would have to travel to create a map of the environment while staying exclusively on paths belonging to the Voronoi graph.

Data structure
To incorporate information about the physical structure of the environment into the graph traversal process, the algorithm uses the Voronoi graph obtained as previously described in conjunction with its bitmap representation and a monochrome image of the environment's floor plan.

A pixel on the floor plan's image can be either black or white, with black pixels representing obstacles and white pixels representing free areas.

Instead, a pixel on the bitmap Voronoi graph can be black, gray, or white, with black pixels representing unexplored nodes of the graph, gray pixels representing explored nodes of the graph, and white pixels representing free areas that are not associated to any node of the graph.

The algorithm also maintains an internal model of a virtual robot whose position at each step of the traversal process coincides with the coordinates in the bitmap Voronoi graph of the node being visited by the algorithm in that step. The model holds the robot's position and orientation in the environment, as well as the range and field of view of the laser range scanner.

Algorithm description
The traversal process starts from the node of the Voronoi graph whose coordinates in the bitmap Voronoi graph are closest to the actual starting position of the robot during exploration. From there, the algorithm proceeds by alternating a *mapping* phase to an *exploration* phase.

Mapping phase During the mapping phase, the algorithm identifies which nodes of the Voronoi graph correspond to pixels of the bitmap Voronoi graph that are *visible* from the robot's pose and uses this information to update its internal representation of which portions of the environment have been mapped so far. Intuitively, visible pixels represent areas of the environment that the robot would be able to perceive with a laser scan from its current pose, and that would consequently become part of the map maintained by the SLAM algorithm in the current exploration step.

To discover such pixels, the algorithm performs a ray casting operation on the superimposition of the bitmap Voronoi graph and the floor plan's image starting from the pixel that corresponds to the current robot's position. A pixel is considered to be visible if it is within the laser's range and field of view and if there are no obstacles, i.e., black pixels on the floor plan's image, that belong to the straight line connecting it to the current robot location. The algorithm then marks all visible nodes as *seen* and turns their corresponding pixels on the bitmap Voronoi graph to grey. The pseudocode for this part of the algorithm is detailed in Algorithm 3.

Algorithm 3 Mapping phase

```
function lineOfSight(VG, bitmapVG, floorplanImg, currPxl,
prevPxl, laserRng, laserFOV)
    visiblePixels = getVisiblePixels(bitmapVG, floorplanImg,
                    currPxl, prevPxl, laserRng, laserFOV)
    visibleNodes = pixelsToNodes(visiblePixels)
    markVisiblePixelsAsVisited(bitmapVG, visiblePixels)
    seenNodes = markVisibleNodesAsVisited(VG, visibleNodes)
    return visibleNodes, seenNodes

function getVisiblePixels(bitmapVG, floorplanImg, currPxl,
prevPxl, laserRng, laserFOV)
    visiblePixels = []
    nearbyPixels = getNearbyPixels(bitmapVG, currPxl, prevPxl,
                   laserRng, laserFOV)
    for p in nearbyPixels do
        pixelsAlongLine = lineBetween(currPxl, p, floorplanImg)
        occupiedPixels = [p in pixelsAlongLine | p == 0]
        if len(occupiedPixels) == 0 then visiblePixels.append(p)
    return visiblePixels

function getNearbyPixels(bitmapVG, currPxl, prevPxl, laserRng,
laserFOV)
    occupiedPixels = [p in bitmapVG | p == 0]
    refAngle = computeAngle(currPxl, prevPxl)
    distances = euclideanDistance(occupiedPixels, currPxl)
    angles = arctan(occupiedPixels, currPxl)
    idx = [i for i in range(0, lgt) | distances[i] < laserRng
          and abs(atan2(sin(refAngle − angles[i]),
          cos(refAngle − angles[i]))) < laserFOV]
    return occupiedPixels[idx]
```

Exploration phase In the exploration phase, the algorithm identifies the node of the Voronoi graph that corresponds to the next frontier to explore and moves the virtual robot towards it, accounting for the areas of the environment that get explored while moving. As we discussed in Chapter 2, in the context of this thesis we assume to proceed using a nearest frontier exploration criterion; however, the algorithm can be modified to explore the

environment according to an arbitrary criterion.

To discover the nearest unexplored frontier, the algorithm searches for the black pixel of the bitmap Voronoi graph that is located at the shortest Euclidean distance from the current robot's position; ties are broken randomly. It then uses Dijkstra's algorithm to compute the shortest path on the Voronoi graph that connects the node of the robot's position to the node that corresponds to the selected pixel.

For each node along the path, the algorithm uses the bitmap Voronoi graph to update the virtual robot's position and orientation according to the node's relative displacement from its immediate predecessor on the path; it then adds the amount of performed translation and rotation to the overall Voronoi traversal distance and Voronoi traversal rotation respectively. Finally, the algorithm performs a mapping phase to identify which pixels would be visible from that pose and updates the *seen* status of their corresponding nodes accordingly, in order to keep track of which areas of the environment have been explored.

The Voronoi traversal distance is estimated by computing the Euclidean distance between the pixels of the bitmap Voronoi graph corresponding to the two nodes, which is a good approximation of the real distance the robot would have to travel to move between them.

On the other hand, the Voronoi traversal rotation is estimated by computing the absolute value of the minimum angular difference between the current orientation of the robot and the gradient of the straight line connecting the two nodes, but only if the two nodes are at a minimum distance from each other. This adjustment is made necessary by the consideration that the relative orientation difference between two nearby pixels does not correctly reflect the actual amount of rotation the robot performs along the trajectory, and is instead strongly influenced by imperfections in the pixels alignment; therefore, evaluating the relative orientation of two nodes only if their distance is sufficiently large represents a better approximation of the actual amount of rotation the robot would perform between them, as it only accounts for rotations in the correspondence of curves in the simulated trajectory.

Once the final node on the path has been reached, the algorithm identifies the next frontier to explore and the loop is repeated. The pseudocode for this part of the algorithm is reported in detail in Algorithm 4.

Stopping condition The algorithm stops when all nodes of the Voronoi graph have been marked as *seen*.

Algorithm 4 Exploration phase

```
function getNearestFrontier(bitmapVG, currPxl)
    occupiedPixels = (bitmapVG == 0)
    distances = euclideanDistance(occupiedPixels, currPxl)
    nearestIdx = argmin(distances)
    return occupiedPixels[nearestIdx]

function exploreVoronoiGraph(VG, bitmapVG, floorplanImage,
startPosition, laserRng, laserFOV, scale, speed, minRotDist)
    totalNodes = len(VG.nodes())
    totalDist, totalRot, numSeenNodes = 0
    partDist, partRot, robotAngle = 0
    prevPxl, currPxl, currAnglePxl = startPosition
    while numSeenNodes < totalNodes do
        nearestPxl = getNearestFrontier(bitmapVG, currPxl)
        nearestNode = pixelToNode(nearestPxl)
        currNode = pixelToNode(currPxl)
        shortestPath = dijkstra(VG, currNode, nearestNode)
        for newNode in shortestPath do
            newPixel = nodeToPixel(newNode)
            totalDist += euclideanDistance(VG, currNode, newNode)
            if partDist > (minRotDist × scale) then
                partRot += atan2(newPxl[0]-currAnglePxl[0],
                           newPxl[1]-currAnglePxl[1])
                totalRot += abs(atan2(sin(partRot − robotAngle),
                             cos(partRot − robotAngle)))
                robotAngle = partRot
                currAnglePxl = newPxl
                partDist = 0
            prevPxl, prevNode = currPxl, currNode
            currPxl, currNode = newPxl, newNode
            visibleNodes, nseen = lineOfSight(VG, bitmapVG,
                                  floorplanImg, currPxl,
                                  prevPxl, laserRng, laserFOV)
            numSeenNodes += nseen
        currPxl = nearestPxl
    return totalDist/scale, totalRot
```

5.3 Model learning

The next part of our solution focuses on solving the learning problem outlined in Section 3.3 by building a model of the relationship between the features we have extracted from the environments of the training set and the observed localization error of said environments.

In the following, we describe the three machine learning techniques that we use to solve the regression problem and we highlight the differences between their approaches to feature selection.

5.3.1 Simple linear regression

The first machine learning technique we examine is simple linear regression [91]. The simplicity of this approach lies in its usage of a single explanatory variable for the characterization of the behavior of the target variable, assuming a linear correlation between the two.

At first, the algorithm selects the component of the localization error that must be used as the target variable and the feature that must be used as the explanatory variable. Then, the algorithm computes the values of the two selected variables for all the environments of the training set and uses ordinary least squares minimization to obtain the regression line that best fits the data. The process is repeated for each of the four components of the localization error and for every feature identified in the previous section.

A first important observation is that this model is not regularized, meaning that the identified regression line is the one that best fits the data with no added constraints. In principle, the lack of regularization in a model can lead to overfitting, a situation in which the model learns the data so well that it also fits the samples' noise and therefore becomes unable to generalize to new samples. However, usually overfitting occurs only when the model has too many parameters with respect to the size of the sample on which the training is performed; since single feature linear regression models only have two parameters (slope and intercept), they do not suffer overfitting and therefore regularization is not necessary.

A second important observation is that this model assumes the dependency between the target variable and the explanatory variable to be linear, hence it performs poorly if a dependency exists but follows a more complex law. Therefore, a low performing linear model does not imply a lack of dependency between two variables, but only that such dependency (if it exists) cannot be described in linear terms.

5.3.2 Explicit feature selection

The second approach we investigate is that of multiple linear regression, i.e., a linear regression technique that uses more than one explanatory variable [91].

In order to select the best explanatory variables among the entire set of features that we have identified in Section 5.2, we use a univariate linear regression test known as *F-regression* [92]. The idea behind F-regression is to evaluate the correlation between each of many regressors and the target variable in the form of an F-score and retain the regressors that contribute the most to the explanation of the target variable's variance.

Considering an input matrix X and an array of observations of the target variable y, the test applies the following formulas for every feature $X[:, i]$:

$$\rho_i = \frac{(X[:, i] - mean(X[:, i])) * (y - mean(y))}{std(X[:, i]) * std(y)}$$

$$F_i = \frac{\rho_i^2}{1 - \rho_i^2} * (n - 2)$$

where $n = len(y)$ is the number of samples, ρ_i is the Pearson's correlation coefficient between the i-th input feature and the target variable, and F_i is the F-score of the i-th input feature.

Feature selection is then performed by choosing the K features with the highest F-scores, with K ranging from one to the overall number of features identified in Section 5.2. The F-regression test is performed multiple times for each K on different splits of the training set, keeping track of which features have the highest F-scores each time; in the end, the algorithm returns the K features that were selected most frequently and uses them as the explanatory variables of the K-features linear model.

This approach is also not regularized, therefore the final models are simply obtained with ordinary least square minimization of the prediction error on the entire training set. However, the usage of different splits of the training set for the selection of the best performing K features for each number of features K through majority voting enhances the probability of obtaining models that are robust and resilient to changes of the training set and therefore helps preventing overfitting.

It should also be noted that, despite the usage of multiple predictors, the models produced with this approach are still linear, so the predicted value is obtained through a linear combination of the input features.

5.3.3 Implicit feature selection

The third and last approach we investigate is also based on multiple linear regression, but directly performs feature selection as part of the training process. The model we use is the *ElasticNet*, a regularized regression method that linearly combines the L1 and L2 penalties of the lasso and ridge regression methods to prevent overfitting [91, 93].

Both *lasso* and *ridge* regression methods are based on the idea that, in order for a linear model to overfit the data, the coefficient of each feature has to be free to assume very large values to ensure enough flexibility for the model to faithfully follow the data points. Therefore, these methods introduce in the ordinary least squares minimization formula a regularization term that penalizes the models that rely on features with very large coefficients. These regularizers, known respectively as L1 and L2 for lasso and ridge, are computed according to the following formulas:

$$\|\beta\|_1 = \sum_j |\beta_j| \tag{5.1}$$

$$\|\beta\|_2 = \sum_j \beta_j^2 \tag{5.2}$$

where $\|\beta\|_1$ is the L1 penalty, $\|\beta\|_2$ is the L2 penalty, and β_j are the coefficients of the features employed by the model as explanatory variables.

ElasticNet extends this approach by combining both regularizers in a single cost function:

$$\hat{\beta} = \arg\min_{\beta}(\|y - X\beta\|^2 + \lambda_1\|\beta\|_1 + \lambda_2\|\beta\|_2)$$

where λ_1, λ_2 are the hyperparameters regulating the relative importance of the L1 and L2 penalties and $\hat{\beta}$ is the set of coefficients that optimizes the regularized cost function.

While the ElasticNet is trained using the entire set of environmental features we identified in Section 5.2 as regressors, the resulting model has non-zero coefficients only for a restricted subset of features, effectively embedding feature selection directly in the training process. Furthermore, the addition of the penalty terms to the overall cost function constrains the approximation power of the model and therefore increases the likelihood of producing models with good generalization capabilities.

5.4 Performance prediction

The last part of our solution focuses on the prediction of the localization error of a SLAM algorithm on an generic environment. More specifically, it

allows the user to predict the localization error that would be made on an environment by the same SLAM algorithm that has been used to gather the training data.

Depending on the specific application, some of the components of the localization error may be more or less relevant than others. In addition, different tasks may require different tradeoffs between prediction accuracy and model complexity. For these reasons, the user is able to specify which components of the localization error should be predicted for a specific environment and which models should be used to predict them. The selection is completely orthogonal: for instance, a single feature linear model may be used to predict the standard deviation of the translational localization error, while an ElasticNet regularized model may be simultaneously used for the prediction of the mean of the rotational localization error.

In addition, it is also possible to let the prediction module automatically choose the best performing model for each component of the localization error the user is interested in predicting. In both cases, the desired models must have already been computed by the model learning module for the prediction to be actually possible.

In order to predict the localization error of the SLAM algorithm in an environment, we process the bitmap representation of its floor plan to compute the characterizing features that are needed by the selected regression models and that we described in Section 5.2. Once the values have been obtained, we use them as the input variables of the models to look up the predicted value of the selected components of the localization error.

5.5 Summary

In this chapter, we complemented the content of Chapter 4 to provide an in-depth explanation of our solution to the problem we outlined in Chapter 3. We first presented a logical high-level overview of the different modules of which the solution is composed; then, we extensively discussed the functionalities covered by each component and we delved into the details of their design.

Chapter 6

System architecture

In this chapter, we present the architecture of the system we conceptually described in Chapters 4 and 5 and we provide a thorough explanation of the main software components that implement the modules of our solution. We first describe the implementation of our data collection methodology, providing an overview of the technological solutions we adopt to perform the automatic exploration of simulated environments and the evaluation of their localization error. Then, we delve into the details of our feature extraction and model learning modules, reviewing the third party libraries we use for their implementation and discussing the technical aspects of their design. Finally, we briefly illustrate the functioning of the performance prediction module.

6.1 Data collection

In this section, we describe the implementation of the data collection methodology that we outlined in Chapter 4. First, we provide a brief overview of the main aspects of the ROS middleware that constitutes the foundation of our implementation. Then, we discuss our choices for the selection of the robotic simulator and the exploration, navigation, and mapping packages that we use to perform simulations. Finally, we discuss the scripts developed to perform automatic exploration and we present the tool that we use to compute the localization error performance metric. A high level diagram of our data collection solution is shown in Figure 6.1.

6.1.1 ROS

Robot Operating System (ROS) [94] is an open-source, meta-operating system for robots that provides high-level features such as hardware abstrac-

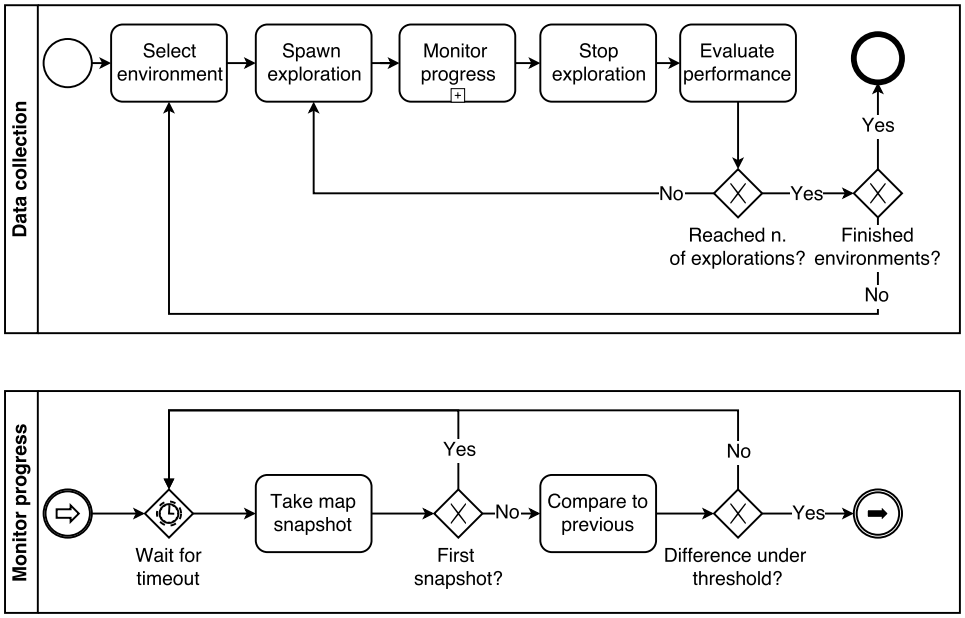


Figure 6.1: A BPMN diagram of the control flow of the data collection process.

tion, low-level device control, package management, standard implementations of frequently-used functionalities, and inter-process message passing. It also provides tools and libraries to write, build, and run distributed code across multiple computers.

ROS acts as a middleware built around the concept of *computation graph*, a peer-to-peer network of ROS processes that are processing data together. There are several types of components that may be part of the graph; in the following, we describe only the ones that we use in this thesis.

- **Nodes**: nodes are processes that perform computations. A complete robot control system typically comprises several nodes, each implementing a different function. These functionalities may be both hardware related, such as those in charge of sensors and actuators control, or purely software related, such as those performing path planning and mapping.

- **Messages**: messages are data structures used by nodes to exchange information with each other. Similarly to C structs, messages can contain any number of typed fields; each field may be a scalar, an array, or a nested structure.

- **Topics**: topics provide a publish/subscribe message passing mechanism to allow inter-node communication. Each topic is identified by

a name which usually describes the content of the messages that are exchanged through it. A node that produces content of a certain kind will publish appropriately formatted messages to the corresponding topic; similarly, a node that is interested in receiving content of that kind will subscribe to it. A single topic may receive content from any number of concurrent publishers and deliver it to any number of concurrent subscribers; similarly, a single node may listen or publish to multiple topics at the same time.

- **Services**: services provide a request/reply communication model that allows two nodes to directly interact with each other. A provider node offers a service by exposing its name to the computation graph; client nodes use the service by sending a request message and awaiting the reply.

6.1.2 Stage

The tool that we use to perform all our robotic simulations is Stage[1,2], a lightweight robot simulator that can handle populations of hundreds of virtual robots in a two-dimensional bitmapped environment. The choice of Stage as a simulator is justified by its simplicity, adaptability, proved robustness, and reasonably good simulation accuracy for 2D environments.

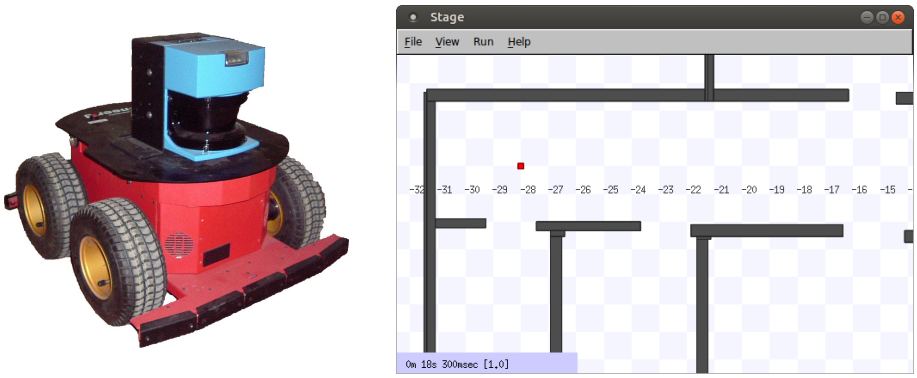


Figure 6.2: On the left, a picture of a Pioneer 3-AT robot equipped with a laser range scanner from SICK. On the right, a snapshot of a Stage simulation, with the virtual robot denoted by the red square.

The input of the simulator is a *world* file providing a description of the setting that Stage should simulate. The world file specifies the path of the

[1] `https://github.com/rtv/Stage`
[2] `http://wiki.ros.org/stage`

file representing the floor plan that should be used to create the simulated environment, the characteristics and the starting positions of the virtual robots, the speed and fidelity of the simulation, and other parameters.

The simulated environment is described by a binary image, with white pixels representing areas of the environment that are free from obstacles, and black pixels representing obstacles.

Stage supports a variety of actuators and sensors, including grippers, wifi modules, blinking lights, laser range scanners, sonars, and infrared ranger sensors. It also provides a simple odometry model to simulate the effect of a uniformly distributed random error on odometry readings; all other simulated sensors are noise-free.

In this thesis, we use it in conjunction with the ROS middleware to perform simulated explorations of indoor environments with a differential-steer drive robot based on the Pioneer 3-AT robotic platform[3] equipped with a laser range scanner. Figure 6.2 shows a picture of the simulated robot and a snapshot of a Stage simulation of a test indoor environment.

6.1.3 Exploration package

The explorer package for ROS[4] provides a suite of algorithms to perform frontier-based exploration. In addition to frontier detection, it offers a number of exploration strategies both for single agent and multiple agents exploration. In our work, this package is used to continuously generate new exploration goals for the robot according to a nearest frontier strategy based on Euclidean distance. In order to increase the exploration efficiency, frontiers that are very close to each other are clustered together and treated as a single frontier. In addition, we modified the package to introduce a recovery mechanism that prevents the robot from getting stuck while trying to reach unreachable frontiers: if the proposed frontier is not reached within a maximum amount of time, the goal is aborted and a random frontier is selected as the next goal.

6.1.4 Navigation package

In order to reach the goals proposed by explorer, our software stack relies on the functionalities offered by the ROS navigation package[5] to perform path planning, obstacle avoidance, and navigation. Internally, these functionali-

[3] `http://www.mobilerobots.com/ResearchRobots/P3AT.aspx`

[4] `http://wiki.ros.org/explorer`

[5] `http://wiki.ros.org/navigation`

ties are implemented by the coordinated effort of several nodes, including a costmap, a local planner, a global planner, and several recovery mechanisms.

Navigation is performed under the assumption that each cell of the occupancy grid map representing the environment is characterized by a traversal cost, with values ranging from 0 (completely free cell) to 254 (certainly occupied cell) and the special value 255 being reserved for cells whose cost is unknown. Each cell's cost is kept updated using the sensory information acquired by the laser scanner; each cell can be either *letal* (when an actual obstacle is present, value 254), *inscribed* (if the cell is less than the robot's inscribed radius away from an actual obstacle, value 253), *possibly circumscribed* (if the cell is less than the robot's circumscribed radius away from an actual obstacle), *freespace* (if its cost is zero), *potentially reachable* (if its cost is between zero and the value associated to the possibly circumscribed state), or *unknown* (if no information is available). The actual cost of a potentially reachable or possibly circumscribed cell is computed according to the following formula:

$$cost = 252 * e^{(-f*(d-r))}$$

where f is a cost scaling factor, d is the distance of the cell from its closest certainly occupied cell, and r is the robot's inscribed radius. Two costmaps are simultaneously maintained during navigation: a global costmap, which covers the entire area being mapped by the SLAM algorithm, and a local costmap, that covers a $6\,\mathrm{m} \times 6\,\mathrm{m}$ area centered around the robot. In our thesis, both costmaps are implemented by the costmap_2d[6] ROS package.

On top of the data provided by the costmap_2d package, the actual navigation process is carried out by a *global planner* and a *local planner*.

The task of the global planner is to propose a viable path for the robot to follow in order to get from its current pose to its next exploration goal while simultaneously avoiding any obstacle that may be present in the environment. Among the several planners offered by the navigation package, we use the navfn behavior of the global_planner[7] because of its fast performance and proved robustness.

On the contrary, the local planner provides the controller that actually drives the robot around the environment. Its job is therefore to use the traversal cost of each cell estimated by the costmap package to determine the `dx`, `dy`, and `dtheta` velocities to send to the robot. For this purpose, we rely on the *trajectory rollout* approach provided by the base_local_planner[8]

[6] `http://wiki.ros.org/costmap_2d`

[7] `http://wiki.ros.org/global_planner`

[8] `http://wiki.ros.org/base_local_planner`

ROS package. Under this approach, the controller discretely samples the robot's velocities space and, for each sampled velocity, performs a forward simulation from the robot's current pose to predict the trajectory the robot would follow if the sampled velocity were applied for a short period of time; each trajectory is then evaluated according to its proximity to obstacles, to the goal, and to the global path, with the highest-scoring trajectory being used to actually drive the robot forward.

Inter-node communication between the global and the local planner is provided by the move_base[9] ROS package. This package provides an action-based mechanism that allows the global planner to send a goal to the local planner and receive feedback on its status (e.g., if the goal is currently being pursued by the local planner, if it has been reached, or if the navigation was aborted); a complete list of all possible goal statuses can be found in the package's documentation[10]. In addition, move_base also maintains the two costmaps used by the local and the global planner and executes the recovery mechanisms that are activated whenever the robot gets too close to an obstacle. In this thesis, we only consider the recovery mechanism provided by the clear_costmap_recovery[11] ROS package that attempts to clear out space in the navigation stack's costmaps by reverting to the static map provided by the mapping package outside of a given radius away from the robot.

6.1.5 Mapping package

Localization and mapping tasks are performed by the GMapping package[12], a ROS wrapper for the OpenSlam's implementation of the GMapping SLAM algorithm[13] originally proposed by Grisetti et al. in [42].

As we mentioned in Chapter 2, GMapping is a particle filter SLAM algorithm that makes use of both odometry data and raw laser range data. Compared to other SLAM algorithms based on particle filters, the usage of Rao-Blackwellization and adaptive resampling makes GMapping particularly efficient while still being robust to noise in the data. These properties, in addition to its widespread usage in the mobile robotics research community, led to our choice of GMapping as a good representative of SLAM algorithms.

For this particular implementation, odometry data is encoded using

[9] `http://wiki.ros.org/move_base`
[10] `http://docs.ros.org/api/actionlib_msgs/html/msg/GoalStatus.html`
[11] `http://wiki.ros.org/clear_costmap_recovery`
[12] `http://wiki.ros.org/GMapping`
[13] `https://www.openslam.org/GMapping.html`

transform messages of the `tfMessage` type. Each message is characterized by a unique numerical identifier and a timestamp and represents the position and orientation of the robot in the space with respect to the fixed map reference frame. The position is encoded in terms of `x`, `y`, and `z` coordinates, while the orientation is encoded as a quaternion.

Laser data is instead encoded by ROS in the LaserScan message type, which stores the information of a single scan from a planar laser range scanner. Each message has a unique numerical identifier and is timestamped in order to be associated with the most appropriate odometry reading. It also contains the actual range measurements of the scan, as well as the characteristics of the sensor that took it in terms of field of view, angular resolution and range.

It should be noted that the GMapping implementation we use is optimized for long-range laser scanners, like the ones of the SICK LMS family; shorter range lasers like the Hokuyo URG scanner are likely to achieve lower performance due to the limited amount of information they provide.

6.1.6 OptiTrack node

Ground truth trajectory data is automatically provided by Stage whenever the exploration is performed in a simulated environment. However, our methodology can also be applied in real world laboratory environments, provided that they are equipped with accurate localization technologies to continuously track the robot's pose throughout the exploration.

In our implementation, we offer native support for the *OptiTrack*[14] positioning system, a tracking technology developed by NaturalPoint, Inc.[15] that uses a variable number of synchronized infrared cameras, each containing a grayscale CMOS imager capturing up to 100 FPS, to triangulate the pose of an infrared reflector placed on the robot itself.

The mocap-optitrack ROS node[16] is therefore used as a data relay between the tracking system and the exploration module, converting the localization data streamed through the network by OptiTrack's Motive proprietary software into messages that are published to a ROS topic.

6.1.7 Automatic exploration script

The automatic exploration script is a custom Python component that orchestrates the sequential automatic exploration of batches of environments.

[14] `https://www.optitrack.com`

[15] `https://www.naturalpoint.com`

[16] `https://github.com/Enri2077/rsbb/tree/master/rsbb_mocap_optitrack`

Given a directory of environments in Stage world format, the script schedules a predefined number of explorations for each of them and starts the first one by means of the roslaunch command of ROS. Once an exploration is underway, it takes snapshots of the map produced by the SLAM algorithm at regular time intervals; each snapshot is compared to the previous one using the mean square error metric. When the difference between two subsequent snapshots falls under a certain threshold, the current exploration is stopped, its localization error is computed and the latest snapshot is saved as the final reconstructed map of the environment. All sensory and ground truth trajectory data are saved to disk in a ROS bag file, while the reconstructed SLAM trajectory is saved in a separate log file in CARMEN format. The script then launches a new exploration of the environment until the number of explorations per environment is reached, at which point it proceeds with the next environment in the dataset. When all environments in the dataset have been explored, the script quits.

6.1.8 Adjustment tools

Because of the way the exploration process is performed, the robot may temporarily stop moving after reaching a frontier while it considers which frontier should be explored next; this implies that all poses on the robot's trajectory that correspond to a pause are repeated multiple times in the logs, which can potentially impact the relations sampling process by increasing their probability of being extracted. Also, since trajectory data are continuously recorded to disk until the very end of an exploration run, the last pose in the trajectory may be saved incorrectly if the recording process is terminated before the writing buffer could be completely flushed.

To mitigate these issues, we have developed a pair of additional support tools that can be invoked after an exploration is completed to adjust the obtained log files. More specifically, the adjust_metric.py Python script is designed to mitigate the effect of pauses during the exploration by analyzing the trajectory logs and keeping a single instance of each pose that is consecutively repeated multiple times. The adjust_output.py Python script instead checks the logs for improper terminations and deletes the entry relative to the last pose if it has been saved incorrectly.

6.1.9 Metric evaluation

In order to evaluate the performance of a SLAM algorithm in an exploration run, we rely on the sampling methodology we outlined in Section 4.2.3 for

the identification of the representative set of ground truth relations on which to compute the localization error performance metric.

First, we extract an initial set of 500 randomly sampled relations from the complete ground truth trajectory file to get an estimate of the localization error's variance, according to the widely used statistical practice of pilot research. For convention, relations are considered to follow the usual progression of time, i.e., we consider the relative transformation that is necessary to transform the earliest of the two randomly sampled timestamped poses into the other. This set of relations is then saved to a ground truth relations file for evaluation.

The actual computation of the localization error is performed by a standalone tool called *metric evaluator*[17], a software developed by Kümmerle et al. at the University of Freiburg as part of the research presented in [11]. The metric evaluator uses the timestamps of the reference relations provided by the ground truth relations file to reconstruct the corresponding relations of the SLAM trajectory log file and computes the translational and rotational components of the localization error according to Equation 3.1. It also computes several statistics about the relations' distribution, including the maximum and minimum observed errors on a single relation, the errors' standard deviations, and the number of relations that were used for processing.

The computed standard deviations are then used to establish the size of a new sample of relations on which to compute a more accurate estimate of the localization error. The intuitive idea behind this process is that if the majority of the relations show a similar amount of translational and rotational error, then a good estimate of both components of the localization error only requires a relatively small sample of relations; on the contrary, if the observed variability is significant, the sample has to grow proportionally in size to ensure an accurate estimation. As we mentioned in Chapter 4, the sample sizes required for an accurate estimation of the translational and rotational components of the localization error may differ, in which case the selected sample size is the maximum of the two.

Once the sample size has been estimated, we repeat the sampling process and we use the metric evaluator again to get the definitive estimate of the localization error of the run.

[17] `http://ais.informatik.uni-freiburg.de/slamevaluation/software.php`

6.2 Feature extraction

In this section, we provide a detailed description of the components that constitute our implementation of the feature extraction methodology that we outlined in Chapter 5. In particular, we discuss the rationale behind our technical choices and we highlight the third party libraries that we use to implement them.

6.2.1 Voronoi graph extraction

The Voronoi extractor is the component that performs the computation of the bitmap Voronoi graph. It is a lightweight standalone C++ utility that represents an extension of the work of Bormann et al. [95] on the usage of Voronoi graphs for room segmentation, as implemented by the ipa_room_segmentation ROS package[18].

The tool processes environments in batch. Each environment is characterized by a Stage world file and a floor plan, which is expected to be in `png` format and to be a grayscale 8-bit image.

The tool uses the OpenCV[19] image processing library to compute the floor plan's Voronoi diagram according to the methodology we described in Section 5.2.3. First, it uses OpenCV's `findContours` method to identify the contours of all the shapes contained in the floor plan; for this operation, the `CV_CHAIN_APPROX_NONE` option is used to preserve every point of each contour. Then, the floor plan's Delaunay triangulation is computed using OpenCV's `Subdiv2D` class and adding to the initial empty instance all the points belonging to the identified contours. Finally, the Voronoi facets of the dual representation are obtained using the `getVoronoiFacetList` method of `Subdiv2D` and their contours are drawn in a new image, together with the previously identified floor plan's contours.

At this point, the tool filters from the obtained bitmap Voronoi graph all the points lying outside the perimeter of the floor plan's building. First, the tool uses OpenCV's `threshold` operator to obtain a binary representation of the floor plan, turning every pixel with a value between 0 and 250 to 0. The obtained image is then inverted and processed with OpenCV's `findContours` operator to identify the contours of every shape in the floor plan, using the `CV_CHAIN_APPROX_SIMPLE` option to compress horizontal, vertical, and diagonal segments and only retain their end points. The longest contour is then assumed to represent the perimeter of the building, its length

[18] `http://wiki.ros.org/ipa_room_segmentation`
[19] `https://opencv.org`

is computed with OpenCV's `arcLength` function, and its shape is approximated with OpenCV's `approxPolyDP` function with a margin of error up to 0.02% of the perimeter's length. The tool then uses the `Geometry` class of the GEOS[20] library to create a polygon that closely approximates the building's shape. Finally, the membership of each point of the bitmap Voronoi graph to the approximating polygon is verified using the `contains` method of the `Geometry` class on the polygon's instance; the points that fail the membership test are removed from the graph.

If the `clear contours` command line option is used, the floor plan's contours are removed from the final image, leaving a clear representation of just the bitmap Voronoi graph. Otherwise, the floor plan's contours are retained, which can be a useful option for visualization purposes.

6.2.2 Voronoi graph construction

The Voronoi graph construction module is responsible for the conversion of the bitmap Voronoi graph produced by the Voronoi extractor to the graph structure that is used by the feature computation module, according to the methodology we presented in Section 5.2.3.1.

At first, the bitmap Voronoi graph is subject to an image dilation operation, which is immediately followed by an image skeletonization operation. These two steps are necessary to obtain a representation of the bitmap Voronoi graph that is as clean and uniform as possible and to remove any imperfection and irregularity that may have been introduced by the tesselation algorithm. The dilation operator is implemented by OpenCV's `dilate` function; instead, the skeletonization operator is implemented by the `skeletonize` function provided by the scikit-image image processing library. It should be noted that the two libraries use a different convention to represent binarized images, with OpenCV expressing values between 0 and 255 and scikit-image using the $[0, 1]$ range. For this reason, it is necessary to perform a format conversion from OpenCV's convention to scikit-image's convention before the application of the skeletonization operator, and to perform the inverse conversion afterwards.

The module then proceeds with the construction of the node-to-pixel and pixel-to-node mapping tables, both implemented as Python dictionaries, that are necessary to keep track of the correspondence between the nodes of the Voronoi graph and the pixels of the bitmap Voronoi graph.

Initially, the image of the bitmap Voronoi graph is scanned in a left-to-right, top-to-bottom order, and a new node is created for each black pixel.

[20] `https://trac.osgeo.org/geos`

Each node is assigned a unique progressive numerical identifier, which is then stored in the pixel-to-node mapping table using the coordinates of the corresponding pixel as key; the node-to-pixel mapping table is then updated with the opposite assignment.

To identify which nodes of the Voronoi graph should be directly linked together, the module iterates on all the elements of the pixel-to-node mapping table and checks the 3×3 pixels area centered around the pixel corresponding to each node: if a pixel in that area also belongs to the bitmap Voronoi graph, its node identifier and that of the current node are stored together in a tuple within the list of all neighboring nodes.

The complete version of the Voronoi graph is then built using the NetworkX library using the list of neighboring nodes to create edges of unitary weight and is subsequently sparsified according to the procedure we described in Section 5.2.3.1.

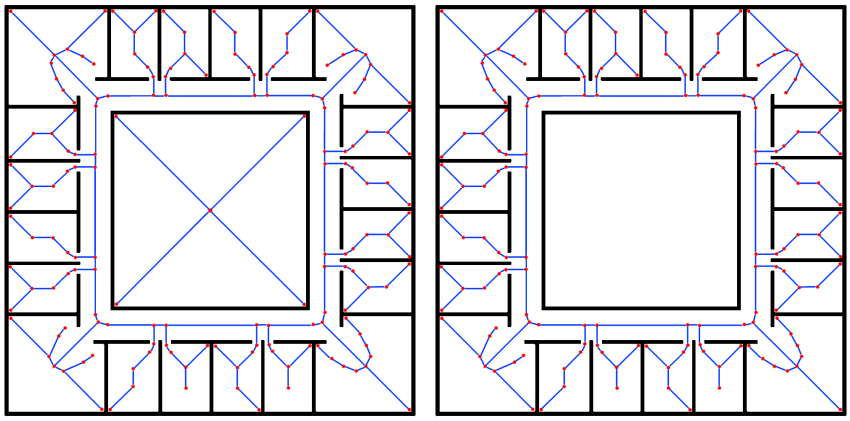


Figure 6.3: An example of Voronoi graph before (left) and after (right) pruning the nodes and edges that do not belong to its largest connected component subgraph.

Lastly, the module prunes from the Voronoi graph all nodes and edges that do not belong to its largest connected component subgraph. This operation is performed to remove nodes and edges that are associated with unreachable parts of the graph, which may be present if one or more inner areas of the environment are completely detached from the rest of the environment, like the case of inner gardens. Figure 6.3 shows an example of environment for which this operation is necessary, in order to exclude the unreachable area in the center.

The obtained graph is serialized to a file on disk using the `pickle` Python serialization format.

6.2.3 Layout reconstruction

The layout reconstruction module is the component that identifies the geometrical and topological structure of an environment, in terms of rooms, corridors and connections between them, and builds the corresponding topological graph. It is developed in Python and is composed of two distinct submodules: the room segmentation module and the topological graph module.

6.2.3.1 Room segmentation module

The room segmentation module is built upon the work of [82] and [83] on the identification of delimited functional spaces, like rooms and corridors, in indoor environments. Its purpose is to implement the geometrical aspects of the layout reconstruction process, according to the algorithm we described in Section 5.2.1. To do so, it relies on the implementations of the Canny edge detection algorithm and of the Hough line transform algorithm provided by OpenCV, applying them to a filtered and binarized representation of the environment's floor plan obtained with OpenCV's threshold operator. To cluster the identified lines into walls, the module uses the implementation of the mean shift operator provided by the scikit-learn[21] machine learning library, which also provides the implementation of the DBSCAN clustering algorithm used to merge together the regions of the image that belong to the same delimited space of the environment, like the same room or corridor. Each space is characterized by a unique alphanumeric identifier and a polygonal representation of its surface implemented with the Polygon class of the Shapely[22] geometric processing library.

6.2.3.2 Topological graph module

The topological graph module is responsible for the implementation of Algorithms 1 and 2 of Section 5.2.2 for the identification of the connections between the spaces detected by the room segmentation module and the subsequent construction of the topological graph. To maximize code reusability, the module makes significant usage of functionalities provided by third party libraries, with the most significant ones being OpenCV, Shapely, and NetworkX[23].

In particular, the OpenCV library is extensively used to perform all the image processing operations related to the analysis of connectivity between

[21] urlhttp://scikit-learn.org/

[22] `https://toblerity.org/shapely/`

[23] `https://networkx.github.io`

spaces that are at the basis of the selective floodfill method described in Algorithm 2.

Similarly, the Shapely library has an important role in the identification of representative points. In fact, to identify a free representative point for each space, the module uses the `representative_point` function of Shapely to obtain a point that is guaranteed to be inside the space's polygon and verifies its suitability by checking if its color in the binarized floor plan's image is white; if it's not, a new candidate point is selected by iteratively verifying the suitability of the pixels belonging to progressively larger squares centered around the original representative point, until a valid point is found. It should be noted that, under the reasonable assumption that spaces represent areas of the environment that are actually walkable, this process is guaranteed to eventually find a suitable representative point.

6.2.3.3 Serialization module

The serialization module has the task of saving the results of the layout reconstruction process, both in terms of geometrical properties of the identified spaces and in terms of the topological structure of the environment, for their subsequent analysis. The results are encoded using XML for maximum flexibility. There are three major components to be saved: the characteristics of the identified spaces, the connections between them, and the pixels-to-meters scale used to encode the environment's floor plan.

Each space is characterized by an alphanumerical identifier, a bounding box, a bounding polygon, and a set of line segments representing the walls delimiting the space's borders. The polygons and the line segments are described by the pixel coordinates of their vertices; in addition, each line segment has a unique alphanumerical identifier. Since vertices are expressed in pixels, it is necessary to keep track of the pixels-to-meters conversion ratio in order to allow the comparison between the geometrical features of different environments, since each environment's floor plan can be expressed with its own scale. This information is therefore stored in the XML document with a *scale* element.

Connections between spaces are represented through *portals*. Each portal indicates the existence of a passage between the two spaces whose alphanumerical identifiers are reported in the portal itself. Optionally, it is possible to specify whether the portal represents a one-way connection or a bidirectional passage; in this thesis, however, we assume that all portals are bidirectional.

6.2.4 Feature computation

The feature computation module is the component that uses the results of the preprocessing operations performed by the Voronoi extractor and by the layout reconstruction module to compute the environmental features described in Section 5.2. It is organized in three sub-modules, one for every type of feature we are interested in analyzing: geometrical, topological, and based on Voronoi graphs.

The geometrical sub-module uses the Shapely library to compute the perimeter and the area of both the environment as a whole, using the bounding polygon as reference, and of the individual spaces identified by the layout reconstruction module. The aggregated measures are then simply computed either by averaging the obtained results, in the case of the average room perimeter and room area, or by summing them, in the case of the total room perimeter.

The topological sub-module leverages the richness of the built-in functionalities of the NetworkX library to efficiently compute all the graph-related features of the topological graph, from the simplest ones like the overall number of nodes and edges, to the most complex ones like the different centrality measures. The only exception to this pattern is represented by the number of terminal and bifurcation points, which are not directly provided by a NetworkX function; however, they are easily computed by counting the number of nodes having only one neighbor and more than two neighbors respectively.

Finally, the Voronoi sub-module is responsible for the computation of all the features related to properties of the Voronoi graph. In particular, it implements Algorithms 3 and 4 of Section 5.2.3.2 for the computation of the Voronoi traversal distance and Voronoi traversal rotation features. In addition, it also uses the NetworkX library to compute on the Voronoi graph the same metrics that the topological sub-module computes on the topological graph.

6.3 Model learning

The model learning module is a straightforward implementation of the machine learning approaches we discussed in Section 5.3. Its core purpose is to verify the existence of correlations between the features computed by the feature extraction module and the true values of the four components of the localization error on the training environments in order to build models that can subsequently be used for SLAM performance prediction.

The first operation it performs is the retrieval of several pieces of information from the logs of the experimental runs performed on the training environments. Most notably, the module has the task of computing the aggregated localization error of each environment starting from the individual localization errors of its runs; in addition, it also computes the average true trajectory length and true trajectory rotation of each environment, so that they can be used as predictors for reference purposes.

It then proceeds with the model learning phase, executing one or more of the three implemented model learning methodologies according to the user's selection. For this purpose, the module makes extensive usage of the scikit-learn machine learning library for Python, which provides built-in functions for both model training and model validation.

In the case of the explicit feature selection approach, the sub-module uses the feature extraction module to retrieve the values of all possible predictors for each of the environments of the training set. Then, for each component of the localization error and for every number of features K from 1 to the overall number of features, it executes the feature selection procedure to identify the predictors that should be used for the K-features model. The selection is done using the `SelectKBest` function of the scikit-learn library using the `f_regression` scoring function to compute the F-score of each predictor on a randomly extracted 80% subset of the training set. Each of the best K performing features gets a vote; the procedure is repeated multiple times for each K on different splits of the training set, keeping track of which features have the highest F-scores each time, and finally selecting the K features that were voted most frequently. The K features are then used to train a K-features model on the entire training set. At the end of the process, the sub-module saves a summary of the feature selection procedure indicating for each component of the localization error the features used by each model.

The behavior is similar for the implicit feature selection approach that we described in Section 5.3.3, with the difference that in this case all features are used simultaneously to train a single regularized model for each component of the localization error using the implementation of the ElasticNet regression technique provided by the scikit-learn library. For this ElasticNet implementation, the hyperparameters are the *L1_ratio* and the *alpha* value, which are defined respectively as:

$$L1\_ratio = \frac{\lambda_1}{\lambda_1 + \lambda_1}$$
$$alpha = \lambda_1 + \lambda_2$$

where λ_1, λ_2 are the coefficients regulating the impact of the L1 and the

L2 penalties of the ElasticNet model that we respectively defined in Equation 5.1 and 5.2. The values of the hyperparameters are chosen with k-fold cross-validation, a technique that consists in dividing the training set in k folds of equal size and iteratively training the model on $k-1$ parts while evaluating it on the remaining one. By keeping a different portion of the dataset out of the training process at each iteration, k-fold cross-validation allows us to select the hyperparameters that represent a good compromise between the model's accuracy and its generalization capability. In our context, we use a number of folds equal to 5 as a balance between the need of leaving out of the training process enough data to perform a meaningful evaluation and the necessity of training the model on a significant and representative portion of the original dataset. The identified hyperparameters are then used to train a model on the entire training set; at the end of the process, the sub-module saves a summary indicating for each component of the localization error the features used by the identified ElasticNet model.

In the simple linear regression case, instead, the sub-module verifies the individual suitability of each of the features we presented in Section 5.2 for the prediction of the four components of the localization error, producing a different model for each possible combination of $\langle$*feature, localization error component*$\rangle$. Because of the simplicity of the model, the risk of overfitting is negligible, and therefore the models can be directly trained on the entire training set without regularization. In addition, it is also possible to directly train just the model associated with a specific feature, an option that can be useful to re-train a previously selected model on a larger training set.

Regardless of the adopted regression technique, the obtained models are saved to disk in pickle format using the `dump` method of the joblib[24] library.

6.4 Performance prediction

The last module of our architecture is the prediction module, whose purpose is to allow the prediction of the localization error made by a SLAM algorithm on a generic environment.

The prediction module requires in input the world file and the floor plan of the environment for which the localization error must be predicted; it also requires the directory containing the models that have been computed by the model learning module and that are available for prediction.

The user is able to select which component of the localization error should be predicted and to choose the preferred prediction model from a list

[24] `https://pypi.python.org/pypi/joblib`

of all the available models for that component. Alternatively, the user can simply specify the component of the localization error that should be predicted and the prediction module automatically selects the best performing model for the selected component.

The prediction module then proceeds with the evaluation of the features defined in Section 5.2 on the input environment using the feature extraction module and uses the selected model to predict the chosen component of the localization error.

6.5 Summary

In this chapter, we discussed the architecture of the solution we conceptually proposed in Chapters 4 and 5. We first described the implementation of our data collection methodology, providing an overview of the technological solutions we adopted to perform the automatic exploration of simulated environments and the evaluation of the corresponding localization error. Then, we delved into the details of our feature extraction and model learning modules, reviewing the third party libraries we used for their implementation and discussing the technical aspects of their design. Finally, we briefly illustrated the functioning of the performance prediction module.

Chapter 7

Experimental results

In this chapter, we present the setup and the results of the experiments that we conducted to evaluate the validity of our approach.

We start by discussing the evaluation procedure and the metrics that we adopted to assess the quality of our models. Then, we review their performance in terms of explanation of the localization error's variance and average prediction accuracy on simulation data. Afterwards, we verify their ability to predict the true localization error of GMapping in three different evaluation scenarios: some simulated environments, a publicly available dataset collected by a real robot, and a set of real robot explorations of our own laboratory at Politecnico di Milano. Finally, we evaluate the computational efficiency of our approach for SLAM performance evaluation and we compare it with that of simulations.

7.1 Evaluation procedure

The performance evaluation of a prediction model is a complicated matter in many branches of science. The exact definition of the qualities that a prediction model should possess to be considered adequate for a certain task is in fact subject to many factors, including the task's criticality, the required level of precision, and the strictness of resource and time constraints. In addition, the research community has not agreed on a single performance metric to assess the accuracy of a model's predictions, proposing instead several measures that address the problem from different points of view.

In the context of this thesis, we restrict our attention to the evaluation of three key aspects of prediction model's performance: the ability to capture a significant portion of the variance of the predicted variable, the average level of accuracy of the predictions, and the power to generalize the results

obtained on simulation data to predict the performance of GMapping in real-world application scenarios. We present the results of the evaluation of the first two and of the last performance measures in Sections 7.2 and 7.3, respectively.

7.1.1 Explained variance

To determine the amount of variance of the localization error that our models are able to explain, we evaluate each model according to its R^2 coefficient of determination, which is a measure of the proportion of the variance in the dependent variable that can be predicted from the independent variables. In its most general form, the formula for its computation is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \overline{y})^2}{\sum_i (y_i - \hat{y}_i)}$$

where n is the number of samples, y_i is the true value of the target variable for the i-th sample, $\hat{y}_i$ is the predicted value of the target variable for the i-th sample, and $\overline{y}$ is the sample mean of the target variable. A R^2 value of 1 implies perfect correlation, while a R^2 value of 0 implies that the model performs no better than the mean prediction, and a negative R^2 implies that the model performs worse than the mean prediction.

However, the R^2 coefficient of a model trained on the entire dataset of available samples may not be representative of the ability of that model to generalize to new samples; in particular, the addition of new features to a model always leads to an increase of the R^2 coefficient due to a greater possibility of overfitting the data, reducing its usefulness as a measure of a model's performance on unseen items.

To overcome this limitation, we therefore randomly divide the original dataset in two distinct parts using an 80% training - 20% testing split and we evaluate the performance of each model on the test set only.

7.1.2 Average prediction accuracy

The average prediction accuracy performance metric is a particularly significant measure in assessing the quality of a prediction model, since a model's usefulness strongly depends on its ability to accurately approximate the true value of the predicted variable.

The first measure that we adopt to evaluate this performance metric is the *root mean square error*, or *RMSE*, which is defined as the sample

standard deviation of the differences between the observed values and their corresponding predictions. It can be computed as:

$$RMSE = \sqrt{\frac{\sum_{k=1}^{n} [\hat{y_k} - y_k]^2}{n}}$$

where n is the number of analyzed environments, $\hat{y_k}$ is the value of the target variable y predicted by the model for environment k, and y_k is the true value of the target variable y for environment k.

Also in this case, the RMSE of a model trained on the entire dataset may not be representative of the ability of that model to generalize to new samples, due to the possibility of the model overfitting the data. We therefore adopt the same approach we previously described to randomly divide the original dataset in a 80% training - 20% testing split and we evaluate the performance of each model on the test set only.

A downside of RMSE is that it is not scale-independent. This does not represent a problem for the evaluation of the relative performance of several models on the same dataset, but it prevents the direct comparison of the performance obtained by a single model on two different datasets. For this reason, in our experiments we also consider the following normalized version of the RMSE:

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}}$$

where y_{max} and y_{min} are respectively the maximum and the minimum observed values of the target variable y in each test set.

7.1.3 Generalization ability

The last condition that we aim to verify is the extent to which our methodology is able to use the results of training sessions on simulated data to predict the localization error made by GMapping in real-world application scenarios.

For this purpose, we compare the true localization errors made by GMapping in a set of real-world experimental scenarios to the corresponding predictions made by our models according to the analysis of the environments' floor plans. We do so by means of the percentage error performance metric, which is defined as:

$$PE = \frac{|\hat{y} - y|}{y}$$

where $\hat{y}$ is the value of the target variable predicted by the model, and y is the target variable's true value.

For reference, this metric is also used to compare the true localization errors made by GMapping in a set of simulated environments to our models' predictions for those environments, in order to evaluate the difference in performance of our method on simulation data and on real-world data.

7.2 Model learning

In this section, we present the experimental results of our model learning approach on simulation data.

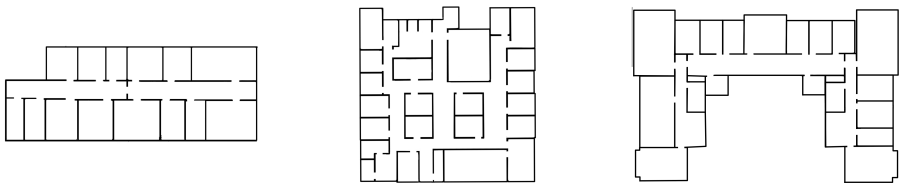
We start by discussing the characteristics of the environments that we simulate as part of our data collection process. We then review the models produced by our methodology and we extensively discuss their performance according to their average prediction accuracy and the percentage of the localization error's variance they are able to explain.

7.2.1 Training and testing environments

To obtain models that can successfully capture the characteristics of real-world applications, simulations have to be performed on a representative set of indoor environments covering a wide range of possible floor plan configurations. For this reason, we consider a total of 100 environments belonging to three different datasets and differing from each other in terms of size, shape, and building type (schools, offices, university campuses, and others). Figure 7.1 shows a sample environment from each of the three datasets.

The first dataset comprises environments originally proposed by Bormann et al. in [95] in the context of their work on room segmentation algorithms. This dataset is available as part of the ipa_room_segmentation ROS package[1] and contains both real and fictional environments, some of which can also be found in the Radish repository. The environments have an area between $100\,\mathrm{m}^2$ and $1{,}000\,\mathrm{m}^2$ and are equally divided between offices and research laboratories at both public and private organizations. The original dataset includes 20 environments in three configurations: empty, with furniture, and with both furniture and trash bins; however, the number of environments that also include a ground truth floor plan is lower. In our experiments, we select a subset of 11 environments for which the ground truth floor plan is available and we limit our analysis to their empty configuration.

[1] `http://wiki.ros.org/ipa_room_segmentation`



(a) Bormann dataset (b) MIT dataset (c) Our dataset

Figure 7.1: A sample environment from each of the three datasets used for data collection (not in the same scale).

The second dataset comprises 25 floor plans of several buildings belonging to the Massachusetts Institute of Technology (MIT) university campus in Cambridge, Massachusetts, USA. These floor plans are a subset of the dataset published by Whiting et al. as part of their work on the generation of topological graphs of multi-building environments [96] and are available upon request on the website of the KTH Royal Institute of Technology[2]. They comprise research laboratories, offices and teaching areas ranging from about 1,000 m^2 to over 30,000 m^2.

Finally, we also use our own dataset of 64 floor plans representing real world buildings [97], of which 26 are offices and 38 are schools, ranging from about 100 m^2 to over 10,000 m^2.

7.2.2 Experimental results

In order to assess the viability of our approach and examine its flexibility in adapting to a variety of robot configurations, we evaluate our models' performance in two different simulation settings, executing an average of 36 runs for each of the 100 available environments in each scenario. Simulations are performed according to the methodology we described in Section 4.2.1, assuming a 95 % confidence level, a margin of error for the estimation of the mean translational localization error of $\pm\,0.03$ m, and a margin of error for the estimation of the mean rotational localization error of $\pm\,0.002$ rad.

Both settings assume the virtual robot to have a translational odometry error of 0.01 m/m and to be equipped with a laser range scanner having a field of view of 270°, an angular resolution of 0.5°, a $[0, 60]$ m range, and a maximum usable mapping range of 30 m. In addition, they assume that mapping is performed with GMapping using 40 particles and processing a new scan whenever the robot travels 1 m, rotates 0.25 rad, or 5 s have passed since the last update of the map. To be able to represent small environments

[2] `http://www.csc.kth.se/~aydemir/floorplans.html`

as well as large environments, we set the size of the reconstructed maps to be $200\,\text{m} \times 200\,\text{m}$, with each cell being $5\,\text{cm} \times 5\,\text{cm}$. Finally, map snapshots are taken at regular intervals of 600 s and the exploration is considered complete once the difference between two subsequent snapshots, as measured by the mean square error metric, is below the value of 10.

However, the two settings differ in the amount of rotational odometry error the virtual robot is expected to show. In the *realistic* setting, the virtual robot is assumed to have a non-perfect rotational odometry and to exhibit a rotational odometry error up to $2\,^\circ/\text{rad}$, which is considered a reasonable approximation of the odometry accuracy of a real wheeled robot. In the *optimistic* setting, the virtual robot is instead assumed to have perfect rotational odometry and therefore to have zero rotational odometry error. Clearly, the optimistic setting is unrealistic from an application-oriented point of view; nevertheless, we think it provides some valuable insights about the resilience of our method to changes in the robot's characteristics.

For both settings, we discuss the fitness of each of the environmental features we introduced in Chapter 5 for the prediction of the localization error, we review the performance of the models obtained with the three machine learning approaches we presented in Section 5.3 and we analyze their benefits and limitations in terms of prediction accuracy, variance explainability, and model complexity.

7.2.2.1 Realistic setting

Linear regression

We start our analysis by reviewing the performance of the simple linear regression method we described in Section 5.3.1, as it is the simplest of the three machine learning techniques we tested. The simplicity of this approach lies in its usage of a single explanatory variable for the characterization of the behavior of the target variable, assuming a linear correlation between the two.

For our evaluation, we consider 5 different random 80% training - 20% testing splits of the original dataset. For each split, we train a linear regression model for each of the features we identified in Section 5.2, and we evaluate its performance on the test data in terms of R^2 coefficient, RMSE, and normalized RMSE. We then consider the average performance of each feature on the 5 test sets as a proxy of its expected performance on unseen data, choosing as the best predictor the one with the highest average R^2 coefficient. Importantly, our experiments show that the model with the highest average R^2 coefficient is also the one offering the best average pre-

diction accuracy, i.e., the lowest average RMSE (and normalized RMSE) on test data, and is therefore optimal on all the considered metrics.

Table 7.1 shows the performance of the best performing feature, in terms of average RMSE, normalized RMSE and R^2 coefficient on the 5 test sets, for each of the four components of the sample localization error, i.e., $\overline{\varepsilon_t(E)}$, $s(\varepsilon_t(E))$, $\overline{\varepsilon_r(E)}$, and $s(\varepsilon_r(E))$. For additional context, Table 7.2 shows the performance of the second best performing feature.

Table 7.1: Average performance on test data in the realistic simulation setting of the best single feature linear model for each component of the localization error. NRMSE is expressed in percentage, while RMSE is expressed in meters for the translational component and in radians for the rotational component.

	Used feature	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	Voronoi traversal distance	0.812	0.145	7.92%
$s(\varepsilon_t(E))$	Voronoi traversal distance	0.677	0.012	6.78%
$\overline{\varepsilon_r(E)}$	Voronoi traversal distance	0.725	0.004	11.30%
$s(\varepsilon_r(E))$	Wall ratio	0.069	0.002	17.04%

Table 7.2: Average performance on test data in the realistic simulation setting of the second best single feature linear model for each component of the localization error. NRMSE is expressed in percentage, while RMSE is expressed in meters for the translational component and in radians for the rotational component.

	Used feature	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	Voronoi eigenvector centr. std.dev	0.400	0.288	15.77%
$s(\varepsilon_t(E))$	Voronoi traversal rotation	0.393	0.019	10.50%
$\overline{\varepsilon_r(E)}$	Voronoi eigenvector centr. std.dev.	0.458	0.006	16.23%
$s(\varepsilon_r(E))$	Voronoi closeness centr. std.dev.	0.012	0.002	17.64%

Looking at Table 7.1, it is immediately evident that the Voronoi traversal distance feature exhibits a significant level of correlation with three out of the four components of the localization error, ranking first in terms of highest average R^2 coefficient and lowest average RMSE. This level of correlation is also noticeable by looking at the regression lines between the Voronoi traversal distance feature and the two components of the translational localization error on the entire set of available environments, as shown in Figure 7.2. The usage of the Voronoi traversal distance feature to predict the mean and standard deviation of the translational localization error is particularly fitting, showing an average normalized RMSE below 8% and

7% respectively. For comparison, the second-best performing feature for the mean of the translational localization error has a significantly lower average R^2 score of 0.400 and a much higher average normalized RMSE of 15.77%, as shown in Table 7.2.

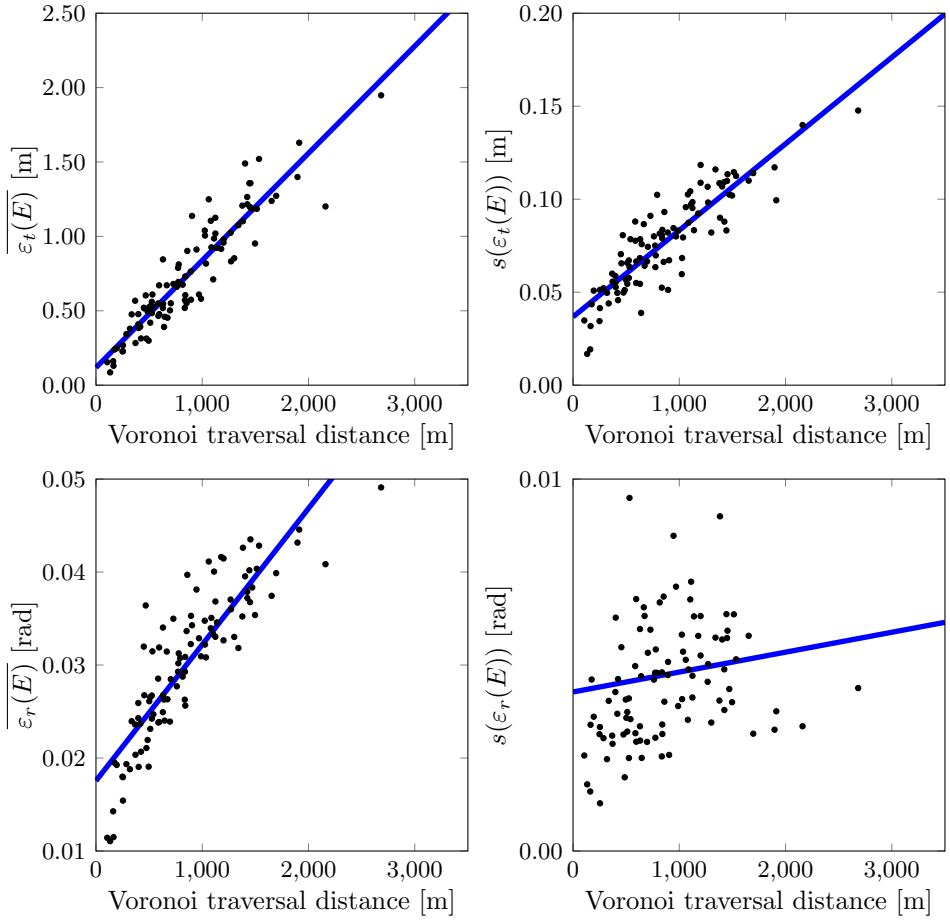


Figure 7.2: The regression line of the Voronoi traversal distance model for the four components of the localization error on the entire set of available environments. The x axis represents the Voronoi traversal distance of the environments, the y axis represents the value of the localization error, and the black dots show the true performance of GMapping as measured with simulations performed with the realistic simulation setting.

A first important observation is that these results were obtained as an average of the performances of the features on 5 different sets of testing environments, none of which were used for model training. Therefore, they are quite representative of the actual prediction performance of each feature on new samples, and suggest that the Voronoi traversal distance feature is capable of predicting the localization error made by GMapping with signifi-

cant accuracy. Furthermore, as single feature linear models are the simplest among all regression models, these results suggest that the usage of more features could result in a further increase of the prediction accuracy.

A second interesting consideration stems from a closer observation of the results relative to the prediction of the standard deviation of the rotational localization error. The low average R^2 coefficients of the first and second best performing feature suggest that none of the predictors we have considered fits the data particularly well, as their performance is not significantly better than that of the mean predictor.

However, a closer inspection of their performance on the 5 test sets reveals that their prediction accuracy is actually quite reasonable, as they both achieve an average RMSE of about 0.002 rad. The same holds true for the Voronoi traversal distance predictor that, despite not ranking in the top 5 predictors in terms of average R^2 score, also has an average RMSE of 0.002 rad. In fact, almost *all* the single feature linear regression models for the prediction of the standard deviation of the rotational localization error have an average RMSE on test data of 0.002 rad.

To illustrate the reason behind this seemingly contradictory result, the bottom right quadrant of Figure 7.2 shows the trend of the standard deviation of the rotational localization error with respect to the Voronoi traversal distance predictor. As it can be seen from the plot, the scale of the measurements is so small that even large percentage errors are practically negligible in absolute terms, effectively making the prediction of this particular component of the localization error almost insignificant for all practical purposes. In fact, the measurements' scale and their apparently erratic behavior suggest that we may be approaching the limit at which the effects of random noise become visible, making it difficult to capture any trace of regularity.

Table 7.3: Average performance on test data in the realistic simulation setting of the single feature linear model based on true trajectory data for each component of the localization error. NRMSE is expressed in percentage, while RMSE is expressed in meters for the translational component and in radians for the rotational component.

	Used feature	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	True trajectory length	0.885	0.125	6.84%
$s(\varepsilon_t(E))$	True trajectory length	0.700	0.014	7.72%
$\overline{\varepsilon_r(E)}$	True trajectory rotation	0.820	0.003	9.44%
$s(\varepsilon_r(E))$	True trajectory rotation	0.014	0.002	17.72%

A third interesting conclusion comes from the comparison between the

level of performance achieved by the Voronoi traversal distance predictor, which allows an ex ante evaluation of the localization error, and the prediction accuracy of models based on the true average amount of distance and rotation travelled by the virtual robot during the simulated explorations, whose results and regression lines are shown in Table 7.3 and in Figures 7.3 and 7.4, respectively. It should be noted that these measures can only be used for the a posteriori analysis of our models' results and cannot be employed for prediction, as their values become known only after the simulated explorations have taken place.

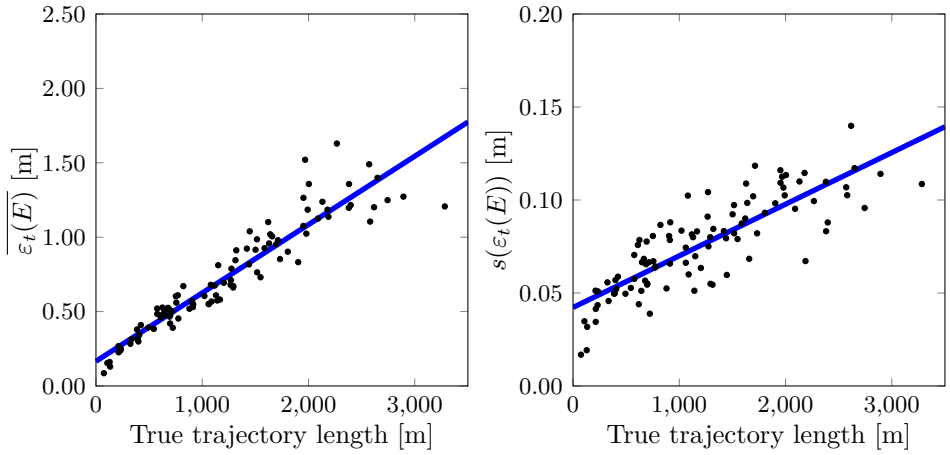


Figure 7.3: The regression line of the average true trajectory length model for the mean and the standard deviation of the translational localization error on the entire set of available environments. The x axis represents the average true trajectory length of the environments, the y axis represents the value of the localization error, and the black dots show the true performance of GMapping as measured with simulations performed with the realistic simulation setting.

The results show that the average measurements on the actual trajectory data are strongly correlated with the true values of the localization error, with the exception of the standard deviation of the rotational component that is substantially uncorrelated as before. This suggests that the amount of localization error made by GMapping on a generic environment is mostly dependent on just the overall amount of travelled distance and rotation and is not significantly related to other characteristics of the environment, a property that considerably simplifies the prediction problem and provides strong empirical evidence in support of using a Voronoi approximation of the actual trajectory as a basis for SLAM performance prediction.

We believe one of the reasons behind this result is that, perhaps in-

tuitively, longer trajectories in indoor environments typically involve the traversal of a higher number of long corridors or large spaces, which are relatively featureless areas that may result in a diminished capability of GMapping to exploit the characteristics of the environment to perform localization and mapping.

This correlation between travelled distance and SLAM error is informally well known in the SLAM community; our observations confirm such intuition and provide a quantitative evaluation of its magnitude. Similar considerations may hold for the increase of the overall amount of rotation, which is likely associated with the robot having to visit a large number of rooms, a factor that may eventually lead to a decrease of the scan matching capability of GMapping.

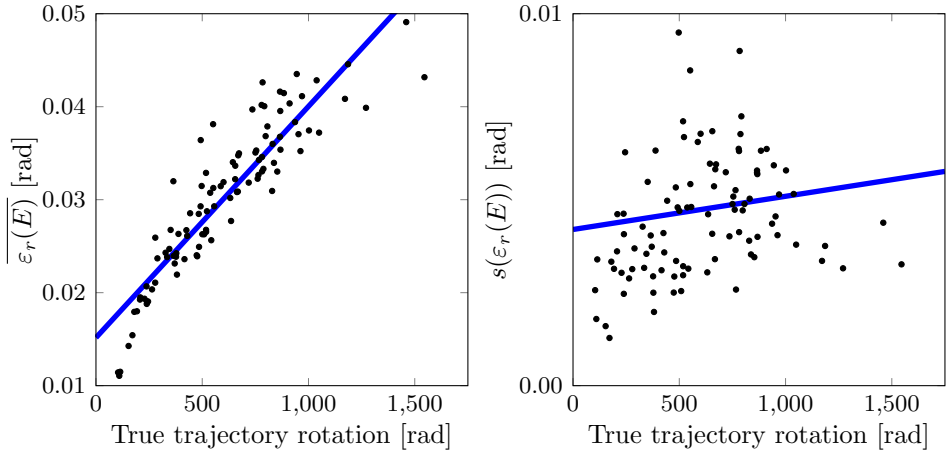


Figure 7.4: The regression line of the average true trajectory rotation model for the mean and the standard deviation of the rotational localization error on the entire set of available environments. The x axis represents the average true trajectory rotation of the environments, the y axis represents the value of the localization error, and the black dots show the true performance of GMapping as measured with simulations performed with the realistic simulation setting.

Finally, if we compare the performance of the true trajectory length feature with that of the Voronoi traversal distance feature for the prediction of the mean of the translational localization error, we can see that their average R^2 coefficients on test data are very similar. This suggests that the Voronoi traversal distance feature is a good predictor of the true distance travelled by the robot; indeed, the R^2 coefficient of determination between the two features on the entire dataset of available environments is 0.828, hinting at a strong level of correlation that can also be visually appreciated

by looking at the left plot in Figure 7.5.

However, the relationship between the Voronoi traversal rotation predictor and the true trajectory rotation predictor is less certain. Looking at the right plot in Figure 7.5, we can see that there is a linear trend between the two; indeed, the R^2 coefficient of determination between the two features on the entire set of available environments is 0.684. Despite this, the points appear more scattered around the regression line, hinting that the Voronoi traversal rotation feature is not fully able to capture the actual rotation performed by the robot. This lower correlation is reflected by the two features' different ability to effectively predict the rotational component of the localization error, with the Voronoi predictor achieving an average R^2 coefficient and RMSE on test data of 0.390 and 0.006 rad respectively, compared to the 0.820 and 0.003 rad achieved by the true trajectory rotation model.

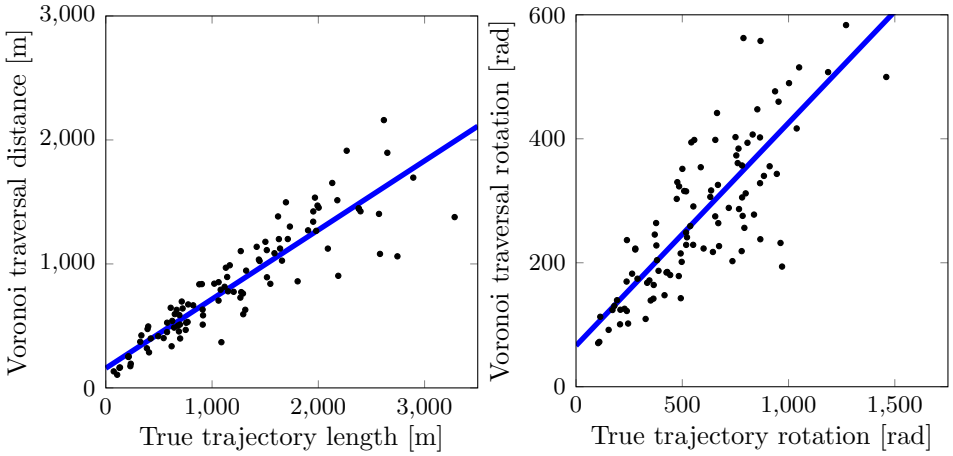


Figure 7.5: On the left, the regression line between the average true trajectory length and the Voronoi traversal distance of the 100 *environments of the realistic simulation set. On the right, the regression line between the average true trajectory rotation and the Voronoi traversal rotation of the same set of environments.*

One of the reasons behind this discrepancy is most likely related to the relatively low level of abstraction of our analysis. Intuitively, the distance between any two pixels on the bitmap Voronoi graph is a reasonable scaled approximation of the true distance travelled by the robot between the corresponding two points of the space; however, their relative orientation is a less faithful representation of the actual amount of rotation the robot performs while moving between the two, which should instead be computed at a higher level of abstraction, i.e., by considering more realistic geometrical primitives than individual pixels. The amount of performed rotation may also have a

stronger dependance on the choice of the path planner than the amount of travelled distance; however, this aspect requires further investigation.

In conclusion, these results suggest that our approach to the estimation of the Voronoi traversal distance feature is a good approximation of the true distance travelled by the virtual robot in the simulation runs, and that it can successfully produce reasonable and usable predictions of all components of the localization error but the standard deviation of the rotational component. At the same time, they also suggest that, while our attempt to link the computation of the Voronoi traversal rotation to a minimum amount of travelled distance seems promising, more sophisticated approaches are needed to fully capture the true rotational behavior of a wheeled robot.

Explicit feature selection

We now proceed with the analysis of the results obtained with the explicit feature selection approach we described in Section 5.3.2. The goal of this analysis is to verify whether more complex linear models based on a combination of multiple features provide any significant advantage over the simpler single feature models we analyzed in the previous section. For the sake of clarity, we briefly recall the main steps and parameters of the adopted methodology.

We first perform an initial 80% training - 20% testing split of the original dataset of environments for which we have the true values of the sample localization error. For any given number of features K, we then repeatedly perform the F-regression test on all the available features for a 100 different 80% training - 20% validation splits of the training set, each time selecting the K features exhibiting the highest F-score on the 80% training data; we then choose the top K predictors from the list of the most frequently selected features as the basis of a multiple-variable linear regression model, which is then trained on the entire training set and whose performance in terms of RMSE and R^2 coefficient is evaluated on the test data.

Figure 7.6 shows the trend of the RMSE on test data for each of the four components of the localization error as the number of features used by the models increases. Looking at the plots, we can see that, perhaps counterintuitively, the usage of additional features doesn't bring significant improvements to the average prediction error for any of the predicted measures but the mean of the translational localization error. On the contrary, the RMSE slightly decreases at best and significantly increases at worst as more features are incorporated into the models.

A notable exception to this pattern is represented by the mean of the translational localization error. In this case, the RMSE on test data has

a global minimum at 5 features, which corresponds to a model using the Voronoi traversal distance, the number of edges and of bifurcation points of the Voronoi graph, and the standard deviation of both the eigenvector and the Katz centrality of the Voronoi graph as predictors. This 5 features model has a RMSE on test data of 0.124 m and a R^2 score on test data of 0.869, which represent a 12% decrease and a 7% increase with respect to the performance achieved by the single feature Voronoi traversal distance model on the same data.

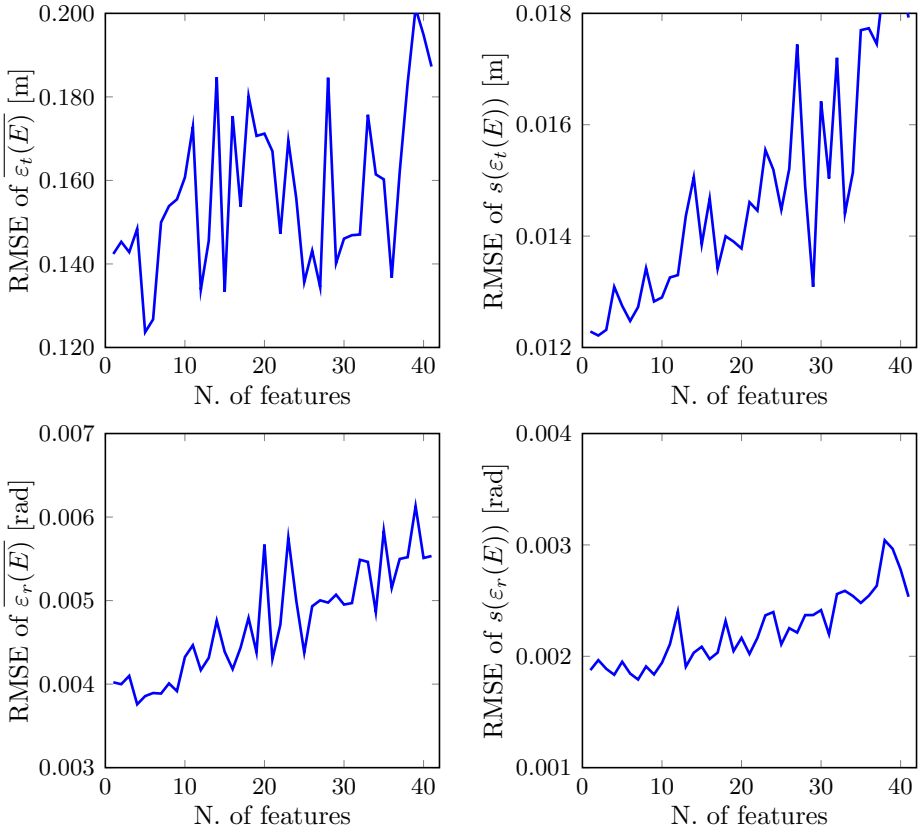


Figure 7.6: The trend of the RMSE on test data of multiple-features linear models as the number of used features increases. Features are selected in decreasing order of individual F-score according to the methodology we presented in Section 5.3.2.

In order to evaluate the stability of these results, we use the same approach presented for the assessment of the performance of the single feature linear models to obtain 4 additional 80% training - 20% testing splits on which to perform the feature selection process. For the sake of conciseness, we hereby report the results of just one of these additional evaluations, but the following observations also apply to the other 3.

Looking at Figure 7.7, we can see that the overall trend of the RMSE on this new test set is not significantly different from that observed in the first evaluation, with the number of features of the best model for each component of the localization error being lower than 10. In particular, the model associated with the best average prediction accuracy for the translational localization error is now composed of 9 features, which are a superset of the 5 features of the previously identified model that also includes the mean and standard deviation of a few additional measures of Voronoi graph centrality.

However, the magnitude of the RMSE on this test set is much smaller than what previously observed; as an example, the mean of the translational localization error now shows a much smaller RMSE of 0.008 m and a slightly higher R^2 coefficient of 0.94.

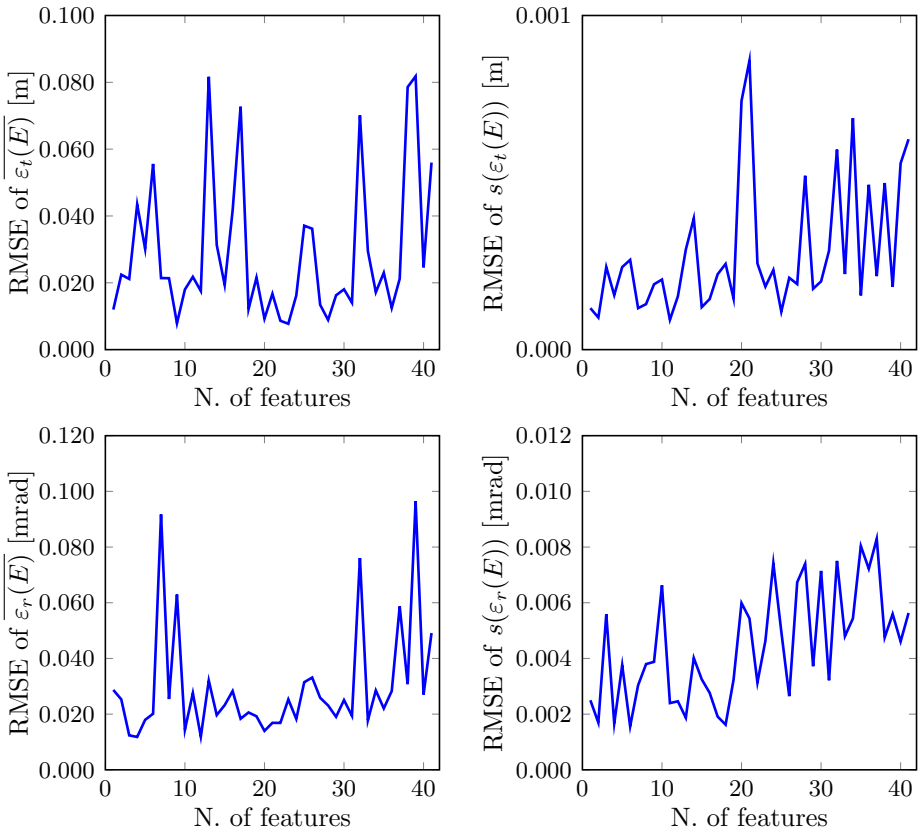


Figure 7.7: The trend of the RMSE on test data of multiple-features linear models as the number of used features increases. Features are selected in decreasing order of individual F-score according to the methodology we presented in Section 5.3.2. Note that, due to the very small scale of the measurements, the RMSE for the rotational components of the localization error is expressed in milliradians.

This dramatic difference in the magnitude of the results can be explained by considering the different composition of the two test sets used in the two evaluations. In the first evaluation, the test set comprised several environments having a Voronoi traversal distance greater than 1,500 m, which we have already determined to be associated with both a high mean and a high variance of the translational localization error and which therefore led to an increase of the RMSE of the model. In the second evaluation, the test set only comprised environments having a Voronoi traversal distance lower than 500 m, which have a much smaller and much more consistent translational localization error and therefore induce a significantly lower RMSE. Moreover, the RMSE on the second test set of the Voronoi traversal distance single feature linear model is 0.012 m, which is itself an order of magnitude smaller than the average one observed as part of our evaluation of the performance of single feature linear models, but that is also remarkably close to that associated with the best model identified with feature selection.

It should be noted that, due to limitations of the set of environments we used for data collection, the fraction of environments that have very large Voronoi traversal distances is relatively small (about 10%), and therefore the extraction of test sets that do not comprise any such environment is a relatively likely scenario; this consideration also justifies the importance of evaluating the performance of our models on multiple test sets, to increase the likelihood of spanning the entire range of environments, from the very small to the very large.

Overall, the results of this evaluation suggest that, at least with respect to the selection of features we consider in this thesis, the usage of additional predictors only leads to relatively small increases in prediction performance, as the average accuracy on unseen samples of the best models identified by the feature selection process is not dissimilar from the one associated with single feature linear model based on Voronoi traversal distance. At the same time, they suggest that the optimal set of features for the prediction of all components of the localization error is substantially stable, with the only exception of the standard deviation of the rotational component which we have already determined to be of little significance; however, they also clearly indicate that the limited amount of data at our disposal requires the evaluation of multiple training and testing sets in order to obtain a reasonable approximation of the models' expected performance on unseen environments.

Implicit feature selection

Finally, we now analyze the prediction accuracy of the models produced

with the implicit feature selection approach we described in Section 5.3.3.

Also in this case, the original dataset of environments is randomly divided with an initial 80% training - 20% testing split. Since the ElasticNet is a regularized regression method, the hyperparameters of the best model for each component of the localization error must be selected through the usage of some kind of validation technique. For our experiments, we use 5-fold cross-validation to evaluate the performance of each combination of the hyperparameters on 5 different random 80% training - 20% validation splits of the complete training set, selecting the values of the hyperparameters that guarantee the minimum cross-validated RMSE. It is worth recalling that, for the specific implementation of the ElasticNet method that we use, the hyperparameters are the *L1_ratio* and the *alpha* value, which are defined respectively as:

$$L1\_ratio = \frac{\lambda_1}{\lambda_1 + \lambda_1}$$
$$alpha = \lambda_1 + \lambda_2$$

where λ_1, λ_2 are the coefficients regulating the impact of the L1 and the L2 penalty of the ElasticNet model. For our experiments, `L1_ratio` is chosen from $[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1]$, while `alpha` is selected within $[0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 1, 3, 5, 10, 30, 50, 100]$. These values are common choices for the `L1_ratio` and the `alpha` hyperparameters; in particular, `L1_ratio` is in the $[0, 1]$ range by construction.

The best performing hyperparameters for each component of the localization error are then used to train a new model for that component on the entire training set. Table 7.4 shows the performance of such models in terms of R^2 value, RMSE, and normalized RMSE on the test set, as well as the number of features they use for prediction and the values of the `L1_ratio` and `alpha` hyperparameters employed for regularization.

Also in this case, the data suggests that the availability of additional features does not lead to a significant increase in the models' performance. On the contrary, the best performing regularized models only use a limited number of features, with the extreme case of the model for the rotational standard deviation of the localization error being identical to its best single feature counterpart.

It is also interesting to notice that the best models identified by the ElasticNet use a number of features comparable to that of the models identified by explicit feature selection. The average prediction accuracy is also similar to both that of the Voronoi traversal distance predictor, as well as that of the feature selection model evaluated on the first 80% training - 20% testing

split. In particular, the ElasticNet model for the prediction of the mean of the translational localization error is a linear combination of the following features: the Voronoi traversal distance and rotation, the sum of the perimeters of all rooms, the average room perimeter and area, the environment's perimeter and area, and the number of bifurcation points and nodes of the Voronoi graph.

Table 7.4: A summary of the characteristics of the best performing models identified by the ElasticNet regression technique for each component of the localization error in one of the 5 *evaluations performed on the realistic simulation setting.*

	N. of features	Alpha	L1 ratio	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	9	1	0.5	0.884	0.134	7.35%
$s(\varepsilon_t(E))$	5	5	0.1	0.759	0.012	6.66%
$\overline{\varepsilon_r(E)}$	5	0.03	0.99	0.730	0.004	10.95%
$s(\varepsilon_r(E))$	1	0.3	0.5	0.014	0.002	17.08%

Similarly to what we did for the evaluation of the explicit feature selection regression technique, we also evaluate the stability of the observed results on 4 additional 80% training - 20% testing splits of the original dataset.

The results of these additional evaluations closely mimic the ones we have observed in the case of the explicit feature selection regression technique. For conciseness, we hereby report the results obtained in just one of such splits, but the same considerations apply to the other 3 as well.

Table 7.5: A summary of the characteristics of the best performing models identified by the ElasticNet regression technique for each component of the localization error in one of the 5 *evaluations performed on the realistic simulation setting. The absence of a value means that its modulus is lower than* 0.001*.*

	N. of features	Alpha	L1 ratio	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	7	10	0.1	0.880	0.015	6.82%
$s(\varepsilon_t(E))$	5	0.3	0.5	0.728	-	7.23%
$\overline{\varepsilon_r(E)}$	3	0.1	0.9	0.732	-	9.87%
$s(\varepsilon_r(E))$	1	0.5	0.5	0.114	-	15.46%

Looking at the results in Table 7.5, we can see that the order of magnitude of the RMSE on this second test set is much smaller than that on the first test set. The reason behind this discrepancy is again the different composition of the two test sets, with the first one comprising environments

with a Voronoi traversal distance of over 1,600 m and the second one exclusively made of environments with a Voronoi traversal distance lower than 530 m. We also point out that the RMSE of the Voronoi traversal distance predictor for the mean of the translational localization error on this test set is 0.018 m, which is very close to that of the optimal regularized model.

Despite the difference in the magnitude of the RMSE, the features selected in the two evaluations are very similar, with those of the second evaluation being a subset of those of the first. The values of the selected hyperparameters are also quite similar, suggesting again that the set of the optimal features is relatively stable with respect to changes of the training and testing sets.

Overall, these results are in line with our expectations, since the Elastic-Net regularization procedure is designed to balance the number of support variables with the complexity of the proposed model. They also confirm our conclusion that, at least with respect to the selection of features we consider in this thesis, the usefulness of employing several features for the prediction of the localization error is limited.

7.2.2.2 Optimistic setting

Linear regression

Also in this case, we start our analysis by discussing the performance of the simple linear regression method we described in Section 5.3.1.

For our evaluation, we follow the same procedure we used for the assessment of our models' performance in the realistic simulation setting. We consider 5 different random 80% training - 20% testing splits of the entire set of 100 environments for which we have simulation data; for each split, we train a linear regression model for each of the features we identified in Section 5.2, and we evaluate its performance on the test data in terms of R^2 coefficient, RMSE, and normalized RMSE. We then consider the average performance of each feature on the 5 test sets as a proxy of its expected performance on unseen data, choosing as the best predictor the one with the highest average R^2 coefficient. Our experiments show that the model with the highest average R^2 coefficient is also the one offering the best average prediction accuracy, i.e., the lowest average RMSE (and normalized RMSE) on test data, and is therefore optimal on all the considered metrics.

Table 7.6 shows the performance of the best performing feature, in terms of average RMSE, normalized RMSE, and R^2 coefficient on the 5 test sets, for each of the four components of the sample localization error, i.e., $\overline{\varepsilon_t(E)}$, $s(\varepsilon_t(E))$, $\overline{\varepsilon_r(E)}$, and $s(\varepsilon_r(E))$. For additional context, Table 7.7 shows the

performance of the second best performing feature.

Table 7.6: Average performance on test data in the optimistic simulation setting of the best single feature linear model for each component of the localization error. NRMSE is expressed in percentage, while RMSE is expressed in meters for the translational component and in radians for the rotational component.

	Used feature	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	Voronoi traversal distance	0.731	0.067	9.93%
$s(\varepsilon_t(E))$	Voronoi traversal distance	0.660	0.015	11.86%
$\overline{\varepsilon_r(E)}$	Voronoi traversal rotation	0.276	0.002	18.72%
$s(\varepsilon_r(E))$	Voronoi eigenvector centr. std.dev.	0.031	0.002	15.85%

Table 7.7: Average performance on test data in the optimistic simulation setting of the second best single feature linear model for each component of the localization error. NRMSE is expressed in percentage, while RMSE is expressed in meters for the translational component and in radians for the rotational component.

	Used feature	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	Voronoi Katz centr. std.dev	0.408	0.107	15.93%
$s(\varepsilon_t(E))$	Voronoi eigenvector centr. std.dev.	0.362	0.021	16.46%
$\overline{\varepsilon_r(E)}$	Voronoi Katz centr. std.dev	0.275	0.002	19.03%
$s(\varepsilon_r(E))$	Voronoi diameter	0.028	0.002	15.93%

The results reported in Table 7.6 confirm that, similarly to what we observed in the realistic simulation setting, the Voronoi traversal distance feature exhibits a significant level of correlation with both the mean and the standard deviation of the translational localization error. A visual confirmation of this property comes from Figure 7.8, which shows the regression lines between the Voronoi traversal distance feature and the two components of the translational localization error on the entire set of available environments. For comparison, the second best performing feature for the mean of the translational localization error has a significantly lower average R^2 score of 0.408 and a much higher average normalized RMSE of 15.93%.

The results for the prediction of the mean and standard deviation of the rotational localization error are much less significant. In particular, all the single feature models for the standard deviation of the rotational localization error have an average RMSE on test data lower than 0.002 rad and an average R^2 score on test data of approximately zero. However, these numbers are an expected consequence of the fact that the optimistic simulation

scenario assumes zero rotational odometry error, and therefore the models are essentially fitting the noise in the measurements.

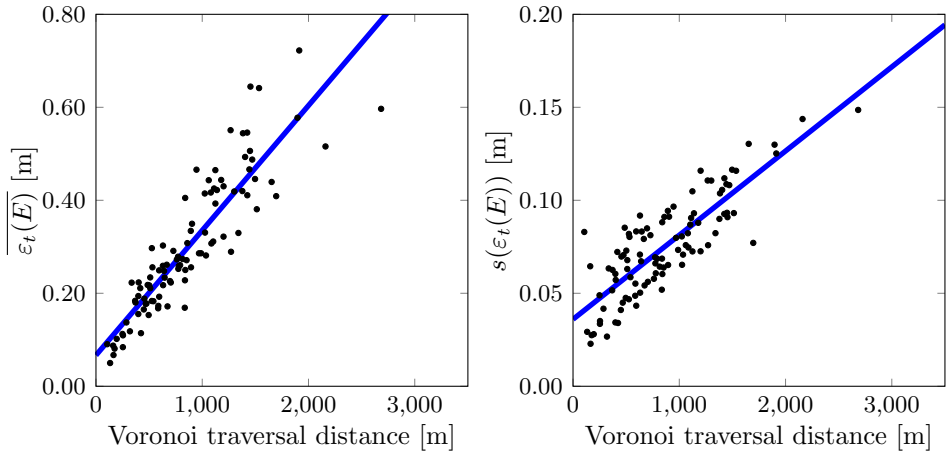


Figure 7.8: The regression line of the Voronoi traversal distance model for the mean and the standard deviation of the translational localization error on the entire set of available environments. The x axis represents the Voronoi traversal distance of the environments, the y axis represents the value of the localization error, and the black dots show the true performance of GMapping as measured with simulations performed with the optimistic simulation setting.

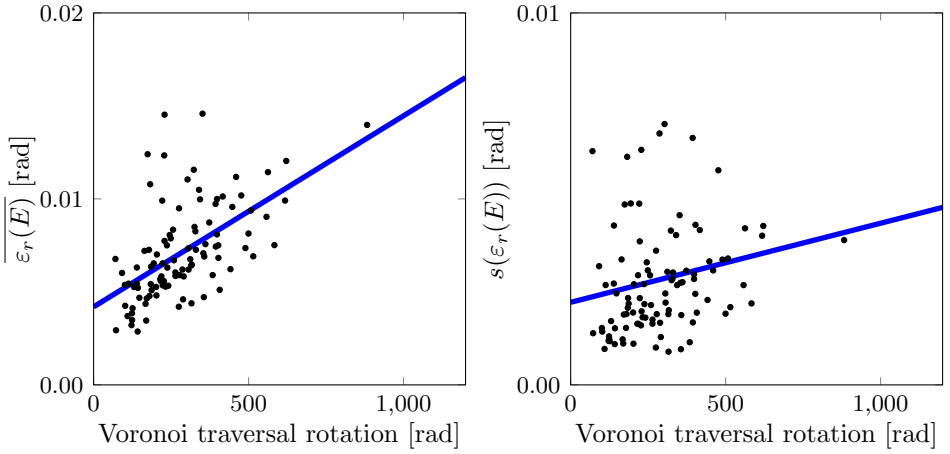


Figure 7.9: The regression line of the Voronoi traversal rotation model for the mean and the standard deviation of the rotational localization error on the entire set of available environments. The x axis represents the Voronoi traversal rotation of the environments, the y axis represents the value of the localization error, and the black dots show the true performance of GMapping as measured with simulations performed with the optimistic simulation setting.

A visual confirmation of this phenomenon can be found by looking at Figure 7.9, which shows the trend of the mean and standard deviation of the rotational localization error with respect to the Voronoi traversal rotation predictor. It is worth observing that the mean of the rotational localization error is not constant for all environments, instead showing a slight increase as the amount of Voronoi traversal rotation increases. We believe this is due to slight numerical imperfections in the way Stage simulates the rotational odometry error, which is never completely zero even when explicitly configured to be so in the simulation settings[3].

A similar pattern can be observed by looking at the performance of the models based on the true average amount of distance and rotation travelled by the virtual robot during the simulated explorations, whose results and regression lines are shown in Table 7.8 and in Figures 7.10 and 7.11, respectively.

Table 7.8: Average performance on test data in the optimistic simulation setting of the best single feature linear model based on true trajectory data for each component of the localization error. NRMSE is expressed in percentage, while RMSE is expressed in meters for the translational component and in radians for the rotational component.

	Used feature	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	True trajectory length	0.812	0.060	8.90%
$s(\varepsilon_t(E))$	True trajectory length	0.615	0.017	13.33%
$\overline{\varepsilon_r(E)}$	True trajectory rotation	0.006	0.002	20.53%
$s(\varepsilon_r(E))$	True trajectory rotation	-0.042	0.002	16.07%

Also in this case, the results show that the average true trajectory length of the simulation runs is significantly correlated with the mean and the standard deviation of the translational localization error, confirming the trend that we have observed for the realistic simulation setting.

On the contrary, the average true trajectory rotation shows little to no correlation with the mean and the standard deviation of the rotational localization error, to the point that the average R^2 value on test data of the model for the prediction of the rotational standard deviation of the localization error is negative. This behavior is a direct consequence of the zero rotational odometry error assumption we adopted for the optimistic simulation setting.

[3] `http://rtv.github.io/Stage/group__model__position.html`

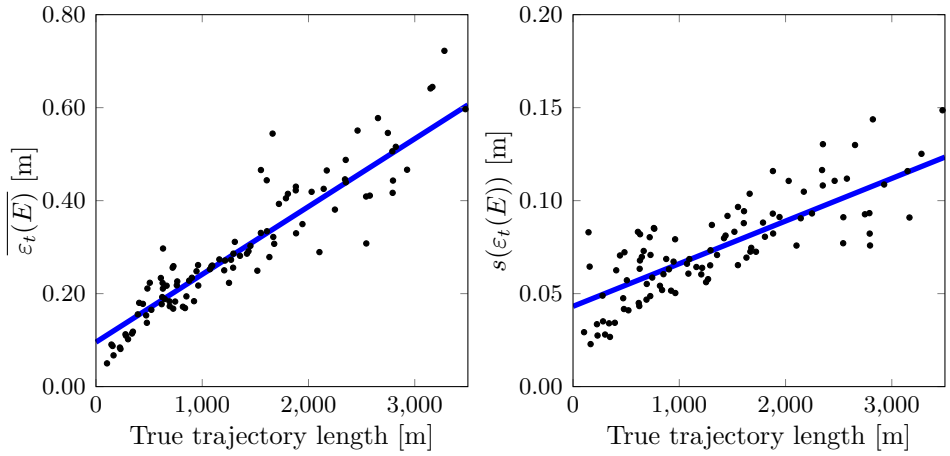


Figure 7.10: The regression line of the average true trajectory length model for the mean and the standard deviation of the translational localization error on the entire set of available environments. The x axis represents the average true trajectory length of the environments, the y axis represents the value of the localization error, and the black dots show the true performance of GMapping as measured with simulations performed with the optimistic simulation setting.

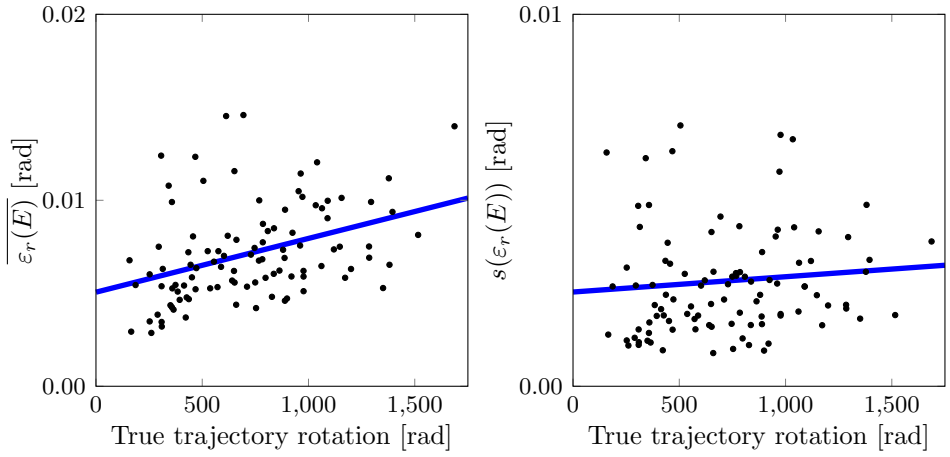


Figure 7.11: The regression line of the average true trajectory rotation model for the mean and the standard deviation of the rotational localization error on the entire set of available environments. The x axis represents the average true trajectory rotation of the environments, the y axis represents the value of the localization error, and the black dots show the true performance of GMapping as measured with simulations performed with the optimistic simulation setting.

Finally, if we compare the performance of the true trajectory length feature with that of the Voronoi traversal distance feature for the prediction

of the mean of the translational localization error, we can see that, also in this case, their average R^2 coefficients on test data are very similar. This confirms our previous intuition that the Voronoi traversal distance feature is a good predictor of the true distance travelled by the robot; indeed, the R^2 coefficient of determination between the two features on the entire dataset of available environments is 0.803, hinting at a strong level of correlation that can also be visually appreciated by looking at the left plot in Figure 7.12.

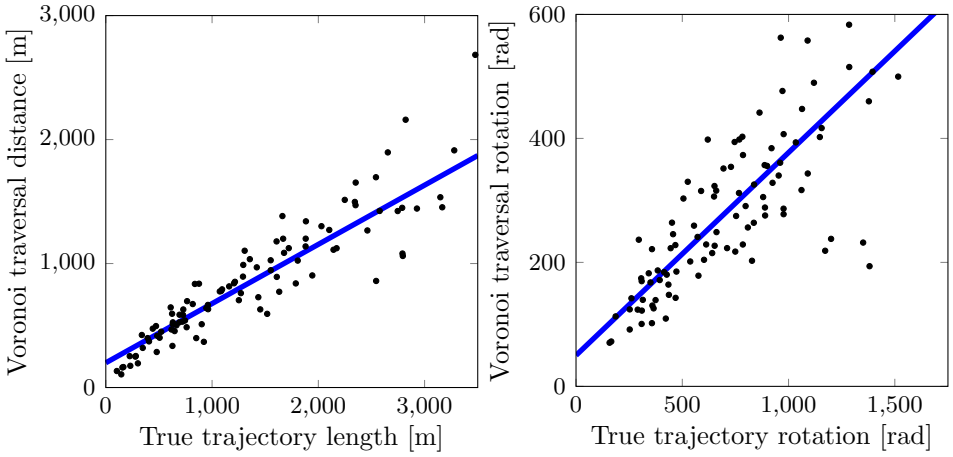


Figure 7.12: On the left, the regression line between the average true trajectory length and the Voronoi traversal distance of the 100 *environments of the optimistic simulation set. On the right, the regression line between the average true trajectory rotation and the Voronoi traversal rotation of the same set of environments.*

Instead, the correlation between the Voronoi traversal rotation predictor and the true trajectory rotation predictor is less significant, similarly to what we observed in the realistic simulation setting. Looking at the right plot in Figure 7.12, we can see that there is a linear trend between the two; indeed, the R^2 coefficient of determination between the two features on the entire set of available environments is 0.621, which is not dissimilar from the one we observed in the realistic simulation setting. However, the points are more scattered around the regression line, confirming our intuition that the Voronoi traversal rotation feature is not fully able to capture the actual behavior of the robot and that more sophisticated predictors are necessary.

Explicit feature selection

We now proceed with the analysis of the results obtained with the explicit feature selection approach we described in Section 5.3.2. For our evaluation, we adopt the same 80% training - 20% testing approach that we followed

for the assessment of the performance of this regression technique in the realistic simulation setting. For conciseness, we hereby report the best models identified in just one of the 5 different evaluations we performed, the general observations about the stability of the observed results are identical to those of the realistic simulation scenario.

Figure 7.13 shows the trend of the RMSE on test data for each of the four components of the localization error as the number of features used by the models increases. Looking at the plots, we can see that, also in this case, the usage of additional features doesn't bring significant improvements to the average prediction error for any of the predicted measures but the mean of the translational localization error. In fact, the RMSE on test data slightly decreases at best and significantly increases at worst as more features are incorporated into the models.

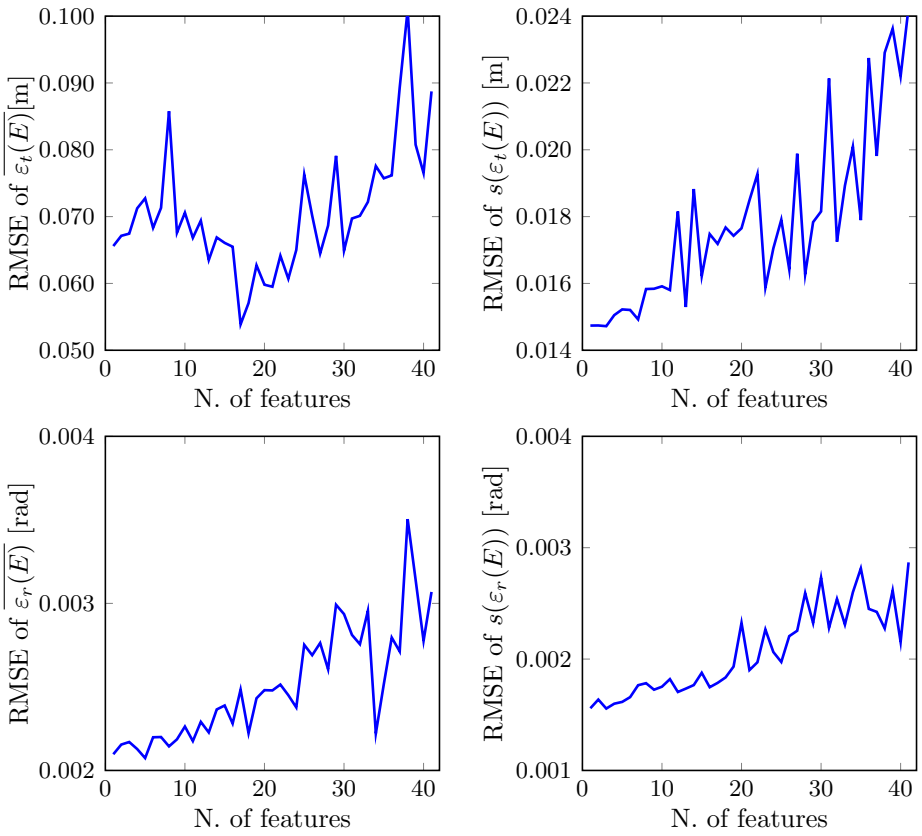


Figure 7.13: The trend of the RMSE on test data of multiple-features linear models as the number of used features increases. Features are selected in decreasing order of individual F-score according to the methodology we presented in Section 5.3.2.

Similarly to what we observed for the realistic simulation scenario, the

only exception to this pattern is the mean of the translational localization error. In this case, however, the RMSE on test data has a minimum at 17 features, which corresponds to a more complex model than the ones found in the realistic simulation scenario; this may be due to the smaller scale of the measurements making the effect of random noise more prominent, but the phenomenon requires further investigation. The 17 features model has a RMSE on test data of 0.054 m and a R^2 score on test data of 0.867, which are a 18% decrease and a 18% increase with respect to the performance achieved by the single feature Voronoi traversal distance model on the same test set. In addition to the Voronoi traversal distance and rotation predictors, this model also uses the number of nodes, edges, bifurcation and terminal points of the Voronoi graph, the number of rooms and the area of the environment, the sum of the perimeters of all rooms, the average shortest path length on the Voronoi graph, and several centrality features of the Voronoi graph.

Implicit feature selection

Finally, we now analyze the prediction accuracy of the models produced with the implicit feature selection approach we described in Section 5.3.3. The `L1_ratio` and the `alpha` hyperparameters are chosen from the same ranges and using the same cross-validation technique that we used for the realistic simulation setting.

Table 7.9 shows the performance of the models associated with the best selection of the hyperparameters for each component of the localization error in one of the 5 different 80% training - 20% testing splits of the original dataset that we consider, in terms of R^2 value, RMSE, and normalized RMSE on the test set. The number of features used for prediction and the values of the `L1_ratio` and `alpha` hyperparameters employed for regularization by each model are also reported.

Table 7.9: A summary of the characteristics of the best performing models identified by the ElasticNet regression technique for each component of the localization error in one of the 5 evaluations performed on the optimistic simulation setting.

	N. of features	Alpha	L1 ratio	R^2	RMSE	NRMSE
$\overline{\varepsilon_t(E)}$	12	0.1	0.99	0.817	0.058	9.09%
$s(\varepsilon_t(E))$	3	0.3	0.9	0.624	0.015	11.59%
$\overline{\varepsilon_r(E)}$	7	0.01	0.99	0.298	0.002	17.57%
$s(\varepsilon_r(E))$	1	10	0.5	-0.402	0.002	15.12%

Overall, these results confirm our intuition that the usage of regularized

regression techniques can be beneficial in finding a compromise between a model's performance and complexity, but also that the usefulness of additional features for the prediction of the localization error is limited.

7.3 Prediction validation

In this section, we further investigate the generalization capabilities of the Voronoi traversal distance linear model, which the results of the previous section identified as a promising tradeoff between model complexity and prediction accuracy, in three different evaluation scenarios: a pair of simulated environments, a publicly available dataset collected by a real robot, and a set of real robot explorations of our own laboratory at Politecnico di Milano. All the results that follow were obtained after training the model on the entire set of 100 environments that we used to perform data collection, in order to fully make use of all the information at our disposal.

7.3.1 Simulation data

In the first scenario, we evaluate the model's performance on a pair of additional simulated environments to provide a more detailed example of its ability to accurately predict the performance of GMapping in simulated settings. We hereby provide a brief description of the characteristics of the two environments and of the simulation settings that have been used for the test, after which we delve into a more detailed discussion of the results.

7.3.1.1 Environments

The first test environment is the Freiburg 52 building of the Bormann room segmentation dataset [95], whose floor plan and a sample SLAM map are shown in Figure 7.14. It consists of 9 rooms and a corridor covering an area of approximately 1,500 m^2.

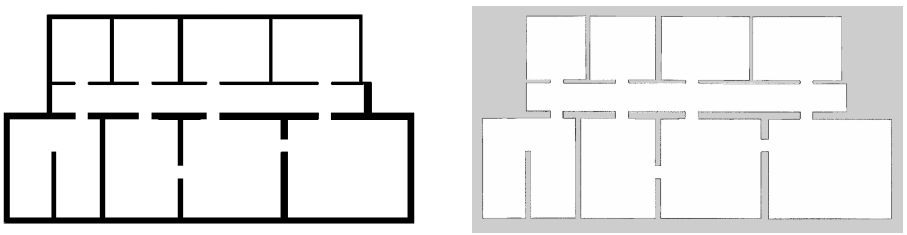


Figure 7.14: The Freiburg 52 test environment. On the left, the ground truth floor plan; on the right, the reconstructed SLAM map of one of the simulation runs.

The second test environment is the Cartesium building of the University of Bremen, whose floor plan and a sample SLAM map are shown in Figure 7.15. It consists of 17 rooms, a corridor, and two access spaces covering an area of approximately 2,200 m^2.

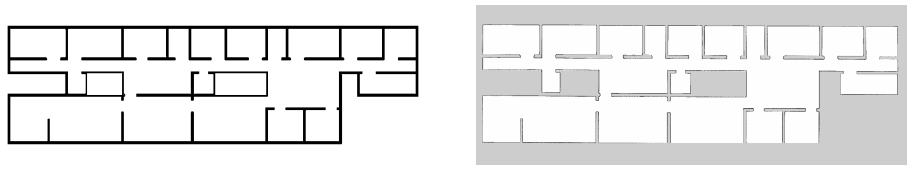


Figure 7.15: The Cartesium building test environment. On the left, the ground truth floor plan; on the right, the reconstructed SLAM map of one of the simulation runs.

The two environments have been explored using both the optimistic and the realistic simulation scenarios with an overall number of 42 exploration runs in each setting for each environment.

7.3.1.2 Results

Tables 7.10 and 7.11 show the comparison between the values predicted by our models and the ground truth localization error obtained on simulation data for the first and the second environment respectively.

Table 7.10: Values of the translational and rotational components of the localization error, in meters and radians respectively, for the Freiburg 52 test environment.

	$\overline{\varepsilon_t(E)}$	$\overline{\varepsilon_r(E)}$	$\overline{\varepsilon_t(E)}$	$\overline{\varepsilon_r(E)}$
Realistic simulation	0.349	0.052	0.019	0.003
Realistic prediction	0.292	0.048	0.021	0.004
Optimistic simulation	0.121	0.038	0.004	0.001
Optimistic prediction	0.131	0.047	0.007	0.003

Table 7.11: Values of the translational and rotational components of the localization error, in meters and radians respectively, for the Cartesium building test environment.

	$\overline{\varepsilon_t(E)}$	$\overline{\varepsilon_r(E)}$	$\overline{\varepsilon_t(E)}$	$\overline{\varepsilon_r(E)}$
Realistic simulation	0.534	0.059	0.024	0.004
Realistic prediction	0.471	0.060	0.025	0.005
Optimistic simulation	0.175	0.049	0.003	0.001
Optimistic prediction	0.198	0.057	0.009	0.003

Looking at the results, we can see that the models' predictions are able to closely approximate the true values of the localization error for both environments.

For the Freiburg 52 environment, in the realistic setting the prediction error on the translational component is below 17% and 8% for the mean and the standard deviation, respectively, while the estimate of the mean of the rotational localization error is off by less than 10% with respect to the simulation data. As expected, the standard deviation of the rotational component shows a less satisfactory prediction error of about 26%, which however amounts to an absolute error of less than 0.05°. In the optimistic setting, the prediction error on the translational component is below 9% and 24% for the mean and the standard deviation, respectively. The estimates of the mean and the standard deviation of the rotational localization error are, as expected, significantly less correct, as the true values of those measures are essentially noise.

For the Cartesium environment, in the realistic setting the prediction error on the translational component is below 12% and 2% for the mean and the standard deviation, respectively, while the estimate of the mean of the rotational localization error is off by less than 5% with respect to the simulation data. In this case, the standard deviation of the rotational component shows a more reasonable prediction error of about 18%, which amounts to an absolute error of about 0.04°. In the optimistic setting, the prediction error on the translational component is below 13% and 16% for the mean and the standard deviation respectively. As before, the estimates of the mean and the standard deviation of the rotational localization error are less accurate, but they are expected to be, as the true values of those measures are essentially noise.

Finally, it is interesting to notice that the models are able to successfully predict the fact that the SLAM performance of GMapping in the Cartesium setting is significantly worse than that in the Freiburg 52 setting, as it can be seen by comparing the models' predictions and the true localization error data of the two environments. This represents a further confirmation of the validity of our approach for SLAM performance prediction and suggests that our methodology could indeed be useful for both benchmarking and robot design purposes.

7.3.2 Real robot dataset

The second evaluation scenario we examine is the application of our performance prediction methodology to a publicly available dataset of four real-

robot runs in one of the buildings of the University of Freiburg.

We hereby provide a brief description of the characteristics of the dataset and of the experimental settings that have been used to perform the data collection, after which we summarize the main preprocessing steps that were necessary to perform our evaluation and we delve into a more detailed discussion of the results.

7.3.2.1 Dataset description

The dataset is provided by the Computer Vision Group of the Faculty of Informatics at the Technical University of Munich[4] and has been collected by Sturm et al. at the University of Freiburg as part of their work on the benchmarking of RGB-D SLAM solutions [67].

The four runs are collected in the same environment, which consists of an L-shaped large industrial hall of about 1,000 m^2 that contains several office containers, boxes, and other feature-poor objects in three runs out of four, and is empty in the fourth one.

The hall is also equipped with a motion capture system from MotionAnalysis[5] consisting of eight Raptor-E cameras with a camera resolution of 1280×1024 pixels and a 300 Hz frame rate. The system is able to continuously track the pose of the robot in the space by performing triangulation on a passive marker attached to the robot itself. This setup was therefore used to capture accurate ground truth data of the robot's trajectory during navigation with millimeter precision. However, as the motion capture system is only able to perform accurate tracking inside a limited area of 10×12 m^2, all the trajectories performed by the robot were confined to this smaller area.

The runs were performed using an ActivMedia Pioneer 3 robot, which follows the specifications of the Pioneer 3-AT research robotic platform[6]. The robot moved through the environment in a non-autonomous fashion and was remotely controlled by the researchers with a joystick.

The robot was equipped with a Microsoft Xbox Kinect sensor mounted horizontally, looking towards the driving direction of the robot and recording RGB-D data at a 640×480 pixels resolution and a 30 Hz frame rate. In addition, it was also equipped with a front-facing laser range scanner, whose characteristics are unfortunately not mentioned in the documentation of the dataset; we therefore infer its capabilities from the analysis of the raw scans

[4] `https://vision.in.tum.de`

[5] `http://www.motionanalysis.com/html/industrial/raptore.html`

[6] `http://www.mobilerobots.com/ResearchRobots/P3AT.aspx`

recorded throughout the runs, obtaining a field of view of 80°, an angular resolution of 1°, a range of about 30 m, and a frequency of 10 Hz.

Similarly, the exact amount of systematic translational and rotational error affecting the odometry readings is not disclosed in the dataset's documentation. It should be noted that the odometry readings of a real robot are typically affected by different kinds of errors, due to a combination of the state of the tires, the slipperiness of the floor, the speed at which the robot's travelling, the intensity of acceleration and deceleration, and other factors. For the purposes of our analysis, we disregard these non idealities and we only consider the amount of systematic error introduced by the odometry sensors. Considering our prior experience with another robot based on the same Pioneer 3-AT research robotic platform, we make an educated guess and assume the translational and rotational odometry error to be not greater than 0.01 m/m and 2.0 °/rad respectively.

7.3.2.2 Preprocessing

In order to apply our SLAM performance prediction methodology to this dataset, it is first necessary to perform some preprocessing operations.

The first operation consists of converting the ground truth trajectory data obtained from the motion capture system to a format that is compatible with our toolkit. The variant of the standard CARMEN log file format used by the authors encodes poses according to their timestamp, 3D position along the `x`, `y`, and `z` axes, and quaternion orientation with respect to the absolute reference frame of the motion capture system. However, our toolkit expects the orientation to be expressed in Euler angles and the positioning data to consists of `x`, `y`, and `yaw` information, as it assumes the robot to be perfectly flat and not experience any variation in pitch or roll. A conversion between the two representations is therefore necessary for a correct interpretation of the trajectory data.

The second operation consists of obtaining the GMapping SLAM estimate of the trajectory followed by the robot. To do so, we first remove from the bag file of each run the raw measurements of the Kinect sensor, as they are not useful in this context and significantly increase the computational effort of our analysis. We then rely on a simplified version of our data collection methodology that uses the `rosbag` command of the ROS suite to replay the recorded sensory information and we use the stored raw laser scans and odometry data as the inputs of the ROS implementation of the GMapping SLAM algorithm. Afterwards, the estimated SLAM trajectory is realigned and reoriented to be expressed in the same global reference frame used for

the ground truth trajectory data. To compensate for the possible variability of the estimates produced by GMapping, the estimation is repeated multiple times. Since the dataset does not include a ground truth floor plan of the environment, we also use the maps reconstructed by GMapping from the recorded navigation data as a basis to obtain a clean floor plan of the environment to use as reference.

Finally, each SLAM estimate of the robot's trajectory is compared to the ground truth trajectory data using the metric evaluator tool that we introduced in Section 6.1.9 to obtain the amount of the localization error made by GMapping on that estimate; the values are then used to compute the average localization error in the environment.

7.3.2.3 Results

As the robot used to collect the dataset has different characteristics, in terms of sensor capabilities and accuracy of the odometry readings, from the two configurations that we used to train our models, a direct comparison between the predictions of our methodology and the true values of the localization error on this benchmark environment only provides limited information on the validity of our approach. In fact, since the performance of a SLAM algorithm in an environment is strongly influenced by the characteristics of the sensory information used to perform the estimation, even small differences in the experimental setup can lead to significant variations of the localization error, potentially undermining the validity of the predictions.

For this reason, we evaluate our models' accuracy on the benchmark environment in an indirect fashion.

First, we compare the true localization error made by GMapping on the recorded sensory information to the average localization error of 10 simulations of the same environment performed with the odometry error and sensors' characteristics of the real robot, with the purpose of evaluating the extent to which our simulation results represent a valid approximation of the attainable level of performance in real-world scenarios. The results of the empty hall scenario refer to the single run that was performed in that setting, while those of the furniture scenario represent an average of the performance achieved by GMapping in the three runs that were performed in that setting.

We then perform 10 additional simulations of the same environment for each of the two simulation settings that we used to train our models. For each component of the localization error, its average value in each simulation setting is then compared to the prediction of the corresponding best

performing single feature linear model.

Each simulation is performed according to the same methodology we described in Chapter 4, with the difference that the time between two subsequent snapshots is reduced from 10 min to 2 min due to the small size of the environment. Figure 7.16 shows a side by side comparison of the maps produced by GMapping in one of the simulation runs and in one of the recorded explorations.

The results of the evaluation are shown in Table 7.12. Note that only the mean values of the localization error can be compared, due to the limited number of runs performed in the real-robot experiments.

A first consideration is that the difference between our predictions and the true values of the localization error, in both the empty hall setting and the furniture setting, is significant. As we discussed earlier, this is an expected consequence of the difference in capabilities, in terms of odometry accuracy and sensors' characteristics, between the real robot used to collect the dataset and the virtual robot employed to gather the training data. In particular, we believe the limited field of view of the laser range scanner used by the real robot to have a negative impact on the effectiveness of scan matching, making it more difficult for GMapping to use laser information to compensate for inaccuracies in the odometry estimate and therefore leading to a higher localization error.

Table 7.12: Values of the translational and rotational components of the localization error, in meters and radians respectively, for the benchmark dataset defined by Sturm et al. in [67].

	Empty hall		With furniture	
	$\overline{\varepsilon_t(E)}$	$\overline{\varepsilon_r(E)}$	$\overline{\varepsilon_t(E)}$	$\overline{\varepsilon_r(E)}$
Dataset real robot	0.189	0.058	0.267	0.070
Simulated real robot	0.223	0.031	0.245	0.045
Optimistic simulation	0.073	0.004	0.089	0.006
Optimistic prediction	0.074	0.004	0.077	0.005
Realistic simulation	0.146	0.023	0.164	0.024
Realistic prediction	0.136	0.018	0.146	0.018

On the contrary, the performance of simulations conducted with the same robot configuration used by the dataset's authors to collect their data is significantly more consistent with the true localization error. This is particularly true for the translational component of the localization error, while the rotational component exhibits a lower level of fidelity. We believe this dif-

Figure 7.16: On the left, the map produced by GMapping using the recorded sensory information of the dataset. On the right, the map produced by GMapping on a simulation performed with the same robot setting used to gather the original dataset.

ference may be due to several factors, including the fact that our estimation of the rotational odometry error disregards many non-idealities that might have affected the performance of the real robot, like the state of the tires and the slipperiness of the floor. At least for the translational localization error, though, these results confirm the hypothesis that Stage simulations can offer an accurate and cost-effective approximation of the behavior of a real robot, which provides an interesting albeit limited validation of our approach to data collection.

The results also show that, when simulations are performed with the same setting used to gather the training data, our methodology is able to provide predictions that closely approximate the measured a posteriori SLAM performance. This is particularly true for the translational component of the localization error, where our method exhibits an error below 1% in the empty hall scenario and below 14% in the furniture scenario under the *optimistic* simulation setting; similar results are obtained in the *realistic* simulation setting, with our predictions being off by less than 7% in the empty hall scenario and less than 11% in the furniture scenario. Instead, the results for the rotational component of the localization error show a slightly lower degree of accuracy, with our predictions being off by about 23% on average.

7.3.3 Laboratory experiments

The third and last evaluation scenario we examine is the application of our performance prediction methodology to our own set of real-robot laboratory

experiments performed at Politecnico di Milano.

We hereby provide a brief description of the characteristics of the dataset and of the experimental settings that have been used to perform the data collection, after which we delve into a more detailed discussion of the results.

7.3.3.1 Experimental setup

The environment we use for our experiments is one of the facilities of the Artificial Intelligence and Robotics Laboratory (AIRLab) at Politecnico di Milano.

The space consists of a square 9×9 m^2 hall, approximately divided in two zones: an L-shaped area hosting tables, desks and shelves with experimental materials, and an empty square 6×6 m^2 area that is used for robot testing and robotic competitions. Figure 7.17 shows the floor plan and an actual picture of the testing environment taken during a robot competition.

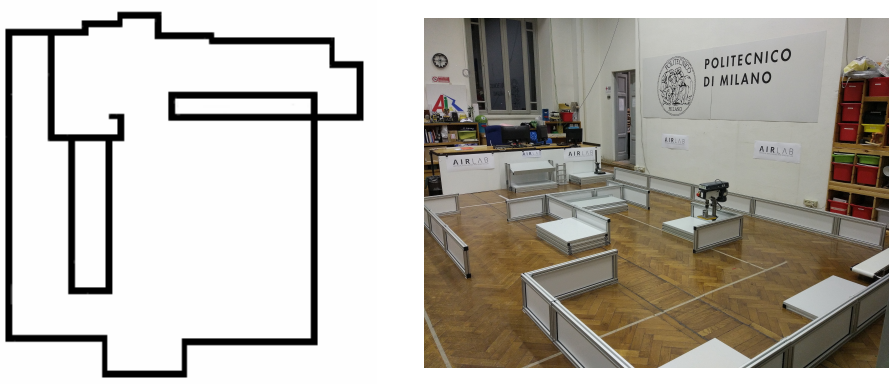


Figure 7.17: The AIRLab facility. On the left, a schematic floor plan of the entire facility; on the right, a picture of the robot testing area during a robot competition.

The testing area is equipped with an OptiTrack[7] motion capture system consisting of 12 grayscale CMOS infrared cameras, i.e., 3 at each corner of the square, operating at 100 Hz; the system uses infrared reflectors to continuously track the position and orientation of objects in space with sub-millimeter precision. In our experiments, this system is used to obtain accurate ground truth trajectory data of the robot's movements, which is then recorded in CARMEN log format.

All our experiments are conducted with a three-wheeled differential drive robot called Robocom[8]. The three wheels are positioned at the corners of

[7] `https://www.optitrack.com`

[8] `http://airlab.elet.polimi.it/index.php/RoboCom\%2BR2P`

an isosceles triangle, with each of the two front-facing wheels being powered by a separate motor and the third wheel being powerless.

The robot is powered by control boards provided by NovaLabs[9] and can both perform autonomous exploration of the environment or be teleoperated with a joypad. To perform autonomous exploration, we use the same implementation we also employ to perform data collection, substituting the Stage robotic simulator with the Robocom[10] ROS package to provide the interface with the robot's sensors and actuators.

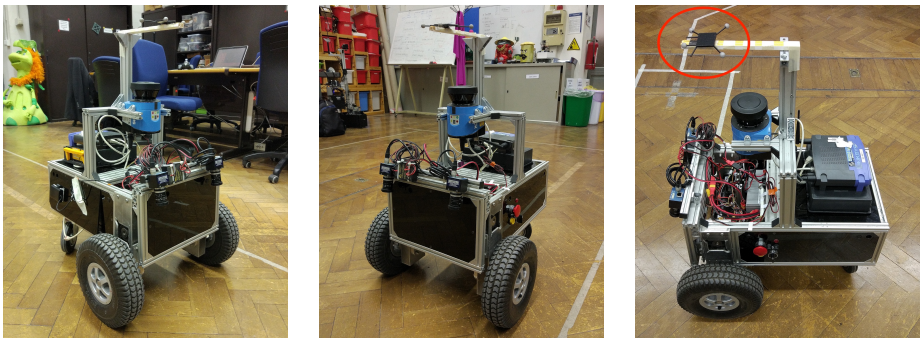


Figure 7.18: The Robocom robotic platform. The OptiTrack marker is highlighted in red in the rightmost image.

The robot is equipped with a front-facing SICK LMS100 laser range scanner for indoor applications, which offers a field of view of 270°, an angular resolution of up to 0.25°, a range of 20 m and a frequency of 50 Hz. The marker used for the OptiTrack motion capture system is mounted on an arm at the top of the robot to avoid any source of interference. Pictures of the actual robot are shown in Figure 7.18.

As for the accuracy of the odometry sensor, we had the robot travel for a known reference distance and perform a fixed number of full in-place rotations to estimate the amount of systematic translational and rotational error affecting the readings, which we evaluated to be not greater than 0.01 m/m and 4.0 °/rad respectively.

All the odometry information from the robot's wheels and the raw laser scans taken by the laser range scanner were recorded in ROS bag format[11].

[9] `http://www.novalabs.io`
[10] `https://github.com/AIRLab-POLIMI/Robocom2`
[11] `http://wiki.ros.org/Bags`

7.3.3.2 Results

We performed a total of 10 real-world experiments, each involving the autonomous exploration of the area of the laboratory covered by the OptiTrack motion capture system. In addition, we also performed 10 simulated explorations of the laboratory in Stage. In both cases, explorations were performed according to the same methodology we described in Chapter 4, with the difference that the time between two subsequent snapshots is reduced from 10 min to 2 min due to the small size of the environment. Figure 7.19 shows a side by side comparison of the maps produced by GMapping in one of the simulation runs and in one of the real-world laboratory explorations. The results of the evaluation are shown in Table 7.13.

Table 7.13: Values of the translational and rotational components of the localization error, in meters and radians respectively, for the experiments conducted in the AIRlab facility at Politecnico di Milano.

	$\overline{\varepsilon_t(E)}$	$s(\varepsilon_t(E))$	$\overline{\varepsilon_r(E)}$	$s(\varepsilon_r(E))$
Robocom	0.088	0.026	0.066	0.010
Realistic simulation	0.101	0.019	0.022	0.004
Realistic prediction	0.120	0.037	0.018	0.004

A first consideration is that the difference between the translational component of the localization error of the simulations performed with the *realistic* simulation setting and that of the explorations performed by the real robot in the laboratory setting is very small. Looking at the data, we can see that the mean and the standard deviation of the translational localization error of the simulations are respectively 15% higher and 26% lower than those of the laboratory runs. This difference may seem significant in percentage, but in absolute terms it represents an error of less than 1.4 cm on the mean and less than 1 cm on the standard deviation, which is negligible for most practical purposes.

The rotational component of the localization error, however, is significantly less consistent than expected. In hindsight, we believe this discrepancy to be mostly due to limitations of our own Robocom robotic platform. While the estimated 0.01 m/m translational odometry error of Robocom coincides with the value of the simulation setting and represents a realistic estimate of the performance of a generic wheeled robot, the estimated 4.0 °/rad rotational odometry error is effectively twice as high as the value of the simulation setting and is probably close to the maximum rotational error that can be tolerated while still achieving acceptable SLAM performance.

Figure 7.19: On the left, the map produced by GMapping on an autonomous exploration of the AIRLab laboratory. On the right, the map produced by GMapping on a simulation performed with the realistic simulation setting on the laboratory floor plan.

As for the assessment of our methodology, the results show that the predictions of our models are substantially consistent with the localization error of the simulations. In particular, the predicted means of the translational and rotational components of the localization error are remarkably close to the corresponding simulation values, achieving a prediction error of less than 19% in both cases.

Also in this case, these differences may seem significant in percentage, but in absolute terms they amount to about 2 cm on the translational component and less than 0.3° on the rotational component. The prediction error made on the standard deviation of the rotational localization error is even smaller, with the prediction being off by about 9%.

The most significant exception is represented by the standard deviation of the translational localization error, which is closer to the one made by the real robot than the one measured on the simulation runs. However, in absolute terms the prediction error is still smaller than 1.8 cm.

Finally, if we compare the predicted mean of the translational localization error with its corresponding laboratory value, we can see that the prediction is overestimating the true error by about 35%. In absolute terms, this corresponds to a prediction error of 3.2 cm, which is almost twice the one made on simulation data. We believe this difference to be at least partly caused by the area explored by Robocom in the laboratory experiments being slightly smaller than the full size of the laboratory, a restriction that is due to the limited coverage of the OptiTrack motion capture system that was used to record the ground truth trajectory data.

7.4 Computational analysis

To conclude the analysis of our method's performance, we spend a few words about its computational efficiency with respect to the usage of simulations for the evaluation of the performances of SLAM algorithms. All measurements refer to computations performed on a computer equipped with a Core 2 Quad Q8300 processor and 4GB of DDR2 RAM.

Table 7.14 compares the time required to predict the four components of the localization error with their best single feature linear regression models to the time required to get an accurate estimate of their values using simulations in the realistic simulation setting. The number of simulations is estimated according to the methodology we presented in Section 4.2.2, assuming a 95 % confidence level, a margin of error for the estimation of the mean translational localization error of $\pm\,0.03$ m, and a margin of error for the estimation of the mean rotational localization error of $\pm\,0.002$ rad. The average simulation time is computed a posteriori on the actual simulations.

Table 7.14: Comparison between prediction and simulation times of 5 *different environments in the realistic simulation scenario.*

	Freiburg 52	Cartesium	Henderson	Bronxville	Cunningham
Prediction time [min]	1	1	3	2	3
N. of simulations	18	25	38	41	44
Avg. simulation time [min]	23	35	58	53	62
Total simulation time [min]	414	875	2204	2173	2728

Looking at the data, it is immediately clear that our performance prediction approach has significant advantages over the usage of simulations. In fact, despite producing estimates of the localization error with a margin of error lower than 20% in most cases, and frequently even lower than 10%, our approach is up to 27 times faster than a single simulation run, and up to $1,000$ times faster than the entire simulation process.

As for the training process, our experiments show that the time required to train our models on the set of 100 environments we used for simulations is in the order of a few hours, depending on the desired number of features. A reasonable estimate when using all the 41 features we consider in this thesis is of 4 to 5 minutes per added environment on average, with smaller, simpler environments requiring less time than larger and more complex ones. Since the training process is mainly CPU-bound, we believe the availability of more computational power could easily bring this estimate down to 1 minute per environment.

Of course, the main bottleneck of our approach is the necessity to gather

a sufficient amount of simulation data to train the models in the first place. However, it should be noted that this step is not required once a suitable model to do performance prediction has been found, and is only necessary if one aims to increase the models reliability by training them on a larger set of environments.

7.5 Summary

In this chapter, we extensively reviewed the validity of our SLAM performance prediction approach in terms of average prediction accuracy, localization error's variance explainability, generalization to new simulation environments and real-world application scenarios, and computational efficiency.

We surveyed the landscape of the features we introduced in Chapter 5 and we identified the Voronoi traversal distance feature as the best performing predictor for the mean and the standard deviation of the translational localization error, as well as for the mean of the rotational localization error. On the contrary, we determined that the Voronoi traversal rotation feature, albeit promising, is not yet sufficiently sophisticated to fully describe the behavior of an actual wheeled robot.

We also verified that the usage of additional predictors, at least with respect to the set of features we consider in this thesis, is generally not helpful, and that single feature linear models are almost always the best compromise between model complexity and prediction accuracy.

Finally, we successfully used our models to predict the performance of the GMapping SLAM algorithm in two real-world application scenarios, obtaining reasonably accurate approximations of the true values of the localization error in both cases.

Chapter 8

Conclusions and future research directions

In this thesis, we addressed the problem of the performance prediction of SLAM algorithms in the context of autonomous mobile robotics.

We highlighted how state of the art techniques for SLAM evaluation cannot be used in unexplored scenarios, making it difficult to draw conclusions on general SLAM performance and posing a significant obstacle towards widespread adoption of autonomous mobile robots.

We therefore proposed an approach to overcome these limitations by using machine learning regression techniques to learn the relationship between an environment's structure and the expected performance of a SLAM algorithm in it, in order to allow an ex ante evaluation of SLAM performance in unseen environments. Compared to existing SLAM evaluation techniques, our method has the main advantage of offering an estimation of the expected performance of a SLAM algorithm in an environment prior to its exploration, potentially allowing robot designers and manufacturers to assess the suitability of a SLAM algorithm for a given application scenario at design time.

We evaluated the validity of our approach on both simulated environments and real robot experiments, considering several types of environmental features and regression methods in different evaluation scenarios. The evaluation has shown that our approach is able to adequately capture the relationship between an environment's structure and the expected performance of a SLAM algorithm in it, and to predict the performance of a SLAM algorithm in an unseen environment with high accuracy.

In conclusion, we believe that our work represents a promising first step towards the achievement of more reliable, general, and useful methods for

SLAM algorithms performance evaluation. Among the many possible extensions of our work, we hereby list those that we deem most significant and well worthy of future research.

More SLAM algorithms. In this research, we limited the scope of our analysis to the behavior of the GMapping SLAM algorithm. Determining the extent to which our results are transferrable to other SLAM algorithms, both when using the same sensory information and when considering different sensor types, is a crucial step towards enabling an effective usage of the proposed approach for robot design and SLAM algorithms comparison.

More robot configurations. Another important step towards strengthening the applicability of our method is the extension of the set of robot configurations used to collect the training data. Albeit in a very limited fashion, the results we obtained in the *optimistic* and *realistic* simulation settings already suggest that variations in the accuracy of the odometry readings have a strong impact on the expected localization error of GMapping; however, the impact of other factors, like the field of view and the range of the laser range scanner, still remains to be precisely determined. A more comprehensive study is therefore required to establish the nature of their impact over the performance of SLAM algorithms and embed this knowledge into a more powerful and versatile prediction model. In addition, the results we obtained in the realistic simulation setting suggest that the choice of the global and local path planners could have a significant impact on the amount of rotational localization error, posing the need for the evaluation of a richer set of planning algorithms.

Larger, more varied, and more realistic training set. The datasets we used to train our models comprise 100 environments of different sizes, shapes, and building types, and already constitute a solid base to draw conclusions about the validity of our approach; however, all environments were assumed to be uncluttered, static, and made of non-transparent materials to allow a proper functioning of the LIDAR sensor. A primal goal of future research is to extend our methodology to environments that more closely replicate the operational conditions in which mobile autonomous robots are expected to operate and that cover an even broader set of possible application scenarios.

Other features and models. Finally, a significant expansion of our research revolves around the investigation of a broader set of environmental

features to be used as predictors and of more complex and sophisticated machine learning methodologies. In particular, we believe that the usage of non-linear models, generalized linear models, and automatic feature extraction methods based on deep learning could bring a sizeable improvement in the accuracy of our models. Naturally, the higher complexity of these approaches also requires a much larger training set to draw meaningful conclusions, making this research opportunity both remarkably interesting and particularly challenging.

Bibliography

[1] S. Ceriani, G. F. Fontana, A. Giusti, D. Marzorati, M. Matteucci, D. Migliore, D. Rizzi, D. G. Sorrenti, and P. Taddei, "RAWSEEDS: ground truth collection systems for indoor self-localization and mapping," *Autonomous Robots*, vol. 27, no. 4, pp. 353–372, Sept. 2009.

[2] B. Balaguer, S. Carpin, and S. Balakirsky, "Towards quantitative comparisons of robot algorithms: Experiences with SLAM in simulation and real world systems," in *Proceedings of the Workshop on Performance Evaluation and Benchmarking for Intelligent Robots and Systems at International Conference on Intelligent Robots and Systems (IROS) 2007*, 2007.

[3] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2D SLAM techniques available in robot operating system," in *Proceedings of the 2013 International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct. 2013, pp. 1–6.

[4] D. M. Turnage, "Simulation results for localization and mapping algorithms," in *Proceedings of the 2016 Winter Simulation Conference (WSC)*, Dec. 2016, pp. 3040–3051.

[5] F. Amigoni, E. Bastianelli, J. Berghofer, A. Bonarini, G. Fontana, N. Hochgeschwender, L. Iocchi, G. Kraetzschmar, P. Lima, M. Matteucci, P. Miraldo, D. Nardi, and V. Schiaffonati, "Competitions for benchmarking: Task and functionality scoring complete performance assessment," *IEEE Robotics Automation Magazine*, vol. 22, no. 3, pp. 53–61, Sept. 2015.

[6] A. Howard and N. Roy, "The robotics data set repository (Radish)," http://radish.sourceforge.net/, 2003.

[7] G. Fontana, M. Matteucci, and D. G. Sorrenti, "The RAWSEEDS proposal for representation-independent benchmarking of SLAM," in

Proceedings of the Workshop on Experimental Methodology and Benchmarking in Robotics Research at the 2008 Robotics: Science and Systems (RSS) Conference, 2008.

[8] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR)*, 2003, pp. 317–323.

[9] T. Colleens, J. J. Colleens, and D. Ryan, "Occupancy grid mapping: An empirical evaluation," in *Proceedings of the 2007 Mediterranean Conference on Control Automation (MED)*, Jun. 2007, pp. 1–6.

[10] G. Fontana, M. Matteucci, and D. G. Sorrenti, "RAWSEEDS: Building a benchmarking toolkit for autonomous robotics," in *Methods and Experimental Techniques in Computer Engineering*, F. Amigoni and V. Schiaffonati, Eds. Springer International Publishing, 2014, pp. 55–68.

[11] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "On measuring the accuracy of SLAM algorithms," *Autonomous Robots*, vol. 27, no. 4, pp. 387–407, Sept. 2009.

[12] S. Schwertfeger and A. Birk, "Map evaluation using matched topology graphs," *Autonomous Robots*, vol. 40, no. 5, pp. 761–787, Jun. 2016.

[13] F. Amigoni, S. Gasparini, and M. Gini, "Good experimental methodologies for robotic mapping: A proposal," in *Proceedings of the 2007 International Conference on Robotics and Automation (ICRA)*, Apr. 2007, pp. 4176–4181.

[14] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, Feb. 2007.

[15] H. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI Magazine*, vol. 9, no. 2, pp. 61–74, Jul. 1988.

[16] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[17] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, Feb. 1998.

[18] A. P. Gee and W. Mayol-Cuevas, "Real-time model-based SLAM using line segments," in *Proceedings of the Second International Symposium on the Advances in Visual Computing, Part II*, pp. 354–363.

[19] T. Lemaire and S. Lacroix, "Monocular-vision based SLAM using line segments," in *Proceedings of the 2007 International Conference on Robotics and Automation (ICRA)*, Apr. 2007, pp. 2791–2796.

[20] V. Nguyen, A. Harati, A. Martinelli, R. Siegwart, and N. Tomatis, "Orthogonal SLAM: a step toward lightweight indoor autonomous navigation," in *Proceedings of the 2006 International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2006, pp. 5007–5012.

[21] R. Lakaemper, "Simultaneous multi-line-segment merging for robot mapping using mean shift clustering," in *Proceedings of the 2009 International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2009, pp. 1654–1660.

[22] H. Zender, O. M. Mozos, P. Jensfelt, G.-J. Kruijff, and W. Burgard, "Conceptual spatial representations for indoor mobile robots," *Robotics and Autonomous Systems*, vol. 56, no. 6, pp. 493–502, 2008.

[23] S. Thrun and A. Bü, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI), volume 2*, ser. AAAI'96, 1996, pp. 944–950.

[24] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (SLAM): toward exact localization without explicit localization," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 2, pp. 125–137, Apr. 2001.

[25] H. J. Chang, C. S. G. Lee, Y. C. Hu, and Y.-H. Lu, "Multi-robot SLAM with topological/metric maps," in *Proceedings of the 2007 International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2007, pp. 1467–1472.

[26] J. L. Blanco, J. A. Fernandez-Madrigal, and J. Gonzalez, "A new approach for large-scale localization and mapping: hybrid metric-topological SLAM," in *Proceedings of the 2007 International Conference on Robotics and Automation (ICRA)*, Apr. 2007, pp. 2061–2067.

[27] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 304–312, Jun. 1992.

[28] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings of the 1997 Computational Intelligence in Robotics and Automation (CIRA)*, Jul. 1997, pp. 146–151.

[29] W. Burgard, M. Moors, C. Stachniss, and F. E. Schneider, "Coordinated multi-robot exploration," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 376–386, Jun. 2005.

[30] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *Proceedings of the 2000 International Conference on Robotics and Automation (ICRA)*, vol. 1, 2000, pp. 476–481.

[31] L. Freda and G. Oriolo, "Frontier-based probabilistic strategies for sensor-based exploration," in *Proceedings of the 2005 International Conference on Robotics and Automation (ICRA)*, Apr. 2005, pp. 3881–3887.

[32] C. Stachniss and W. Burgard, "Exploring unknown environments with mobile robots using coverage maps," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, ser. IJCAI'03, Aug. 2003, pp. 1127–1132.

[33] R. Sim and N. Roy, "Global A-optimal robot exploration in SLAM," in *Proceedings of the 2005 International Conference on Robotics and Automation (ICRA)*, Apr. 2005, pp. 661–666.

[34] A. A. Amanatiadis, S. A. Chatzichristofis, K. Charalampous, L. Doitsidis, E. B. Kosmatopoulos, P. Tsalides, A. Gasteratos, and S. I. Roumeliotis, "A multi-objective exploration strategy for mobile robots under operational constraints," *IEEE Access*, vol. 1, pp. 691–702, 2013.

[35] D. Lee and M. Recce, "Quantitative evaluation of the exploration strategies of an intelligent vehicle," in *Proceedings of the 1994 Symposium on Intelligent Vehicles*, Oct. 1994, pp. 538–543.

[36] F. Amigoni, "Experimental evaluation of some exploration strategies for mobile robots," in *Proceedings of the 2008 International Conference on Robotics and Automation (ICRA)*, May 2008, pp. 2818–2823.

[37] D. Holz, N. Basilico, F. Amigoni, and S. Behnke, "Evaluating the efficiency of frontier-based exploration strategies," in *Proceedings of the 41st International Symposium on Robotics (ISR) and of the 6th German Conference on Robotics (ROBOTIK)*, Jun. 2010, pp. 1–8.

[38] A. Elfes, "A sonar-based mapping and navigation system," in *Proceedings of the 1986 International Conference on Robotics and Automation (ICRA)*, vol. 3, Apr. 1986, pp. 1151–1156.

[39] L. Kleeman, "Advanced sonar and odometry error modeling for simultaneous localisation and map building," in *Proceedings of the 2003 International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, Oct. 2003, pp. 699–704.

[40] A. Diosi, G. Taylor, and L. Kleeman, "Interactive SLAM using laser and advanced sonar," in *Proceedings of the 2005 International Conference on Robotics and Automation (ICRA)*, Apr. 2005, pp. 1103–1108.

[41] G. Nützi, S. Weiss, D. Scaramuzza, and R. Siegwart, "Fusion of IMU and vision for absolute scale estimation in monocular SLAM," *Journal of Intelligent and Robotic Systems*, vol. 61, no. 1, pp. 287–299, Jan. 2011.

[42] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 International Conference on Robotics and Automation (ICRA)*, Apr. 2005, pp. 2432–2437.

[43] B. Steux and O. E. Hamzaoui, "TinySLAM: a SLAM algorithm in less than 200 lines C-language program," in *Proceedings of the 11th International Conference on Control Automation Robotics Vision*, Dec. 2010, pp. 1975–1979.

[44] D. M. Cole and P. M. Newman, "Using laser range data for 3D SLAM in outdoor environments," in *Proceedings of the 2006 International Conference on Robotics and Automation (ICRA)*, May 2006, pp. 1556–1563.

[45] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *Proceedings of the International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Nov. 2011, pp. 155–160.

[46] J. Civera, A. J. Davison, and J. M. M. Montiel, "Dimensionless monocular SLAM," in *Proceedings of the 3rd Iberian Conference on Pattern Recognition and Image Analysis, Part II*, Jun. 2007, pp. 412–419.

[47] D. Pangercic, R. B. Rusu, and M. Beetz, "3D-based monocular SLAM for mobile agents navigating in indoor environments," in *Proceedings of the 2008 International Conference on Emerging Technologies and Factory Automation*, Sept. 2008, pp. 839–845.

[48] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007.

[49] G. Hu, S. Huang, L. Zhao, A. Alempijevic, and G. Dissanayake, "A robust RGB-D SLAM algorithm," *Proceedings of the 2012 International Conference on Intelligent Robots and Systems (IROS)*, pp. 1714–1719, Oct. 2012.

[50] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using kinect-style depth cameras for dense 3D modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[51] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millennium*, G. Lakemeyer and B. Nebel, Eds., 2003, pp. 1–35.

[52] R. Smith, M. Self, and P. Cheeseman, *Estimating Uncertain Spatial Relationships in Robotics.* Springer New York, 1990.

[53] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *International Journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, Dec. 1986.

[54] P. Moutarlier and R. Chatila, "An experimental system for incremental environment modelling by an autonomous mobile robot," in *Proceedings of the 1st International Symposium on Experimental Robotics*, 1990, pp. 327–346.

[55] J. Castellanos, R. Martinez-Cantin, J. Tardós, and J. Neira, "Robocentric map joining: Improving the consistency of EKF-SLAM," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 21–29, 2007.

[56] L. M. Paz, J. D. Tardós, and J. Neira, "Divide and conquer: EKF SLAM in $o(n)$," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107–1120, Oct. 2008.

[57] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 376–382, Jun. 1991.

[58] S. Thrun, D. Koller, Z. Ghahramani, H. F. Durrant-Whyte, and A. Y. Ng, "Simultaneous mapping and localization with sparse extended information filters: Theory and initial results," in *Algorithmic Foundations of Robotics V*, J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, Eds. Springer Berlin Heidelberg, 2004, pp. 363–380.

[59] R. M. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters," in *Proceedings of the 2005 International Conference on Robotics and Automation (ICRA)*, Apr. 2005, pp. 2417–2424.

[60] S. Thrun, D. Fox, W. Burgard, and F. Dallaert, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence*, vol. 128, no. 1-2, pp. 99–141, May 2001.

[61] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 2002, pp. 593–598.

[62] A. Eliazar and R. Parr, "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, ser. IJCAI'03, 2003, pp. 1135–1142.

[63] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, Oct. 1997.

[64] L. Carlone, R. . Aragues, J. A. Castellanos, and B. Bona, "A linear approximation for graph-based simultaneous localization and mapping," in *Proceedings of the 2011 Robotics: Science and Systems (RSS) Conference*.

[65] S. Schneider, F. Hegger, A. Ahmad, I. Awaad, F. Amigoni, J. Berghofer, R. Bischoff, A. Bonarini, R. Dwiputra, G. Fontana, N. Hochgeschwender, L. Iocchi, G. Kraetzschmar, P. Lima, M. Matteucci, D. Nardi, and

V. Schiaffonati, "The rockin@home challenge," in *Proceedings of the 41st International Symposium on Robotics (ISR) and of the 6th German Conference on Robotics (ROBOTIK)*, Jun. 2014, pp. 1–7.

[66] R. Dwiputra, J. Berghofer, A. Ahmad, I. Awaad, F. Amigoni, R. Bischoff, A. Bonarini, G. Fontana, F. Hegger, N. Hochgeschwender, L. Iocchi, G. Kraetzschmar, P. Lima, M. Matteucci, D. Nardi, V. Schiaffonati, and S. Schneider, "The rockin@work challenge," in *Proceedings of the 41st International Symposium on Robotics (ISR) and of the 6th German Conference on Robotics (ROBOTIK)*, Jun. 2014, pp. 1–6.

[67] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proceedings of the 2012 International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2012, pp. 573–580.

[68] D. Caruso, J. Engel, and D. Cremers, "Large-scale direct SLAM for omnidirectional cameras," in *Proceedings of the 2015 International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 141–148.

[69] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proceedings of the 2004 International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, Sept. 2004, pp. 2149–2154 vol.3.

[70] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "USARSim: a robot simulator for research and education," in *Proceedings of the 2007 International Conference on Robotics and Automation (ICRA)*, Apr. 2007, pp. 1400–1405.

[71] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardös, "A comparison of SLAM algorithms based on a graph of relations," in *Proceedings of the 2009 International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2009, pp. 2089–2095.

[72] S. Schwertfeger and A. Birk, "Evaluation of map quality by matching and scoring high-level, topological map structures," in *Proceedings of the 2013 International Conference on Robotics and Automation (ICRA)*, May 2013, pp. 2221–2226.

[73] A. Filatov, A. Filatov, K. Krinkin, B. Chen, and D. Molodan, "2D SLAM quality evaluation methods," in *Proceedings of the 21st Confer-*

ence of Open Innovations Association (FRUCT), Oct. 2017, pp. 120–126.

[74] T. Pinville, S. Koos, J.-B. Mouret, and S. Doncieux, "How to promote generalisation in evolutionary robotics: The progab approach," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '11, 2011, pp. 259–266.

[75] M. Chandrasekaran, M. Muralidhar, C. M. Krishna, and U. S. Dixit, "Application of soft computing techniques in machining performance prediction and optimization: a literature review," *The International Journal of Advanced Manufacturing Technology*, vol. 46, no. 5, pp. 445–464, Jan. 2010.

[76] S. H. Young, T. A. Mazzuchi, and S. Sarkani, "A framework for predicting future system performance in autonomous unmanned ground vehicles," *IEEE Transactions on Systems, Man and Cybernetics: Systems*, vol. 47, no. 7, pp. 1192–1206, Jul. 2017.

[77] S. H. Young, T. A. Mazzuchi, and S. Sarkani, "A model-based framework for predicting performance in self-adaptive systems," *Procedia Computer Science*, vol. 28, no. Supplement C, pp. 513–521, 2014, 2014 Conference on Systems Engineering Research.

[78] P. Regier, M. Missura, and M. Bennewitz, "Predicting travel time from path characteristics for wheeled robot navigation," in *Proceedings of the 2017 European Conference on Mobile Robots (ECMR)*, Sept. 2017, pp. 1–6.

[79] S. Dawson, B. Lowe, and M. A. Herzog, "Identification of issues in predicting multi-robot performance through model-based simulations," *Intelligent Control and Automation*, vol. 2, pp. 133–143, Jan. 2011.

[80] F. Amigoni, M. Luperto, and V. Schiaffonati, "Toward generalization of experimental results for autonomous robots," *Robotics and Autonomous Systems*, vol. 90, no. C, pp. 4–14, Apr. 2017.

[81] S. Karlin and H. Taylor, *A First Course in Stochastic Processes*. Academic Press, 1975.

[82] M. Calabrese, "Costruzioni di mappe multilivello per robot mobili autonomi," Master's thesis, Politecnico di Milano, 2016.

[83] V. Arcerito, "Un metodo per la predizione della struttura di edifici parzialmente esplorati per robot mobili autonomi," Master's thesis, Politecnico di Milano, 2017.

[84] P. Billingsley, *Probability and Measure*. Wiley, 1995.

[85] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96, 1996, pp. 226–231.

[86] V. Latora, V. Nicosia, and G. Russo, *Complex Networks: Principles, Methods and Applications*. Cambridge University Press, 2017.

[87] A. Bowyer, "Computing Dirichlet tessellations," *The Computer Journal*, vol. 24, no. 2, pp. 162–166, 1981.

[88] D. F. Watson, "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes," *The Computer Journal*, vol. 24, no. 2, pp. 167–172, 1981.

[89] S. Suzuki and A. Keiichi, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics and Image Processing*, vol. 30, no. 1, pp. 32–46, 1985.

[90] T. Y. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, Mar. 1984.

[91] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[92] M. H. Kutner, *Applied Linear Statistical Models*. McGraw-Hill, 2005.

[93] H. Zou and T. Hastie, "Regularization and variable selection via the Elastic Net," *Journal of the Royal Statistical Society, Series B*, vol. 67, pp. 301–320, 2005.

[94] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *Proceedings of the 2009 International Conference on Robotics and Automation (ICRA), Workshop on Open Source Software*, May 2009.

[95] R. Bormann, F. Jordan, W. Li, J. Hampp, and M. Hägele, "Room segmentation: Survey, implementation, and analysis," in *Proceedings of the 2016 International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1019–1026.

[96] E. Whiting, J. Battat, and S. Teller, "Generating a topological model of multi-building environments from floorplans," in *Proceedings of the 2007 Computer-Aided Architectural Design Futures (CAADFutures) Conference*, 2007, pp. 115–128.

[97] M. Luperto, A. Q. Li, and F. Amigoni, "A system for building semantic maps of indoor environments exploiting the concept of building typology," in *RoboCup 2013: Robot World Cup XVII*, 2014, pp. 504–515.

Appendix A

Parameters manual

In this appendix, we provide an in depth review of the parameters that control the behavior of our system and we detail the configurations we adopted for our experiments. The appendix is organized in three sections: first, we describe the parameters that control the data collection process; then, we review the main settings of the feature extraction module; finally, we discuss the parameters that control model learning.

A.1 Data collection

A.1.1 Stage

The Stage robotic simulator provides a significant variety of parameters to control the many details of the simulations it performs. We hereby limit the discussion to the two types of parameters that we use in the context of our experiments: those related to the capabilities of the virtual robot and those controlling the general properties of the simulation.

A.1.1.1 Virtual robot configuration

The parameters related to the capabilities of the virtual robot are:

- **Localization mode**: defines whether the odometry readings of the robot should reflect its ground truth position in the environment (`gps` mode) or simulate the presence of noise and imprecisions (`odom` mode). To obtain a realistic model of the robot's behavior, we set this parameter to `odom` mode.
- **Odometry error**: defines the amount of noise that should be added to the odometry readings along the `x`, `y`, `z`, and `theta` axes when using

the `odom` localization mode. Each component represents the maximum proportion of error that is introduced on top of the true location and orientation of the robot during the integration of the robot's velocities to compute the odometric position estimate, expressed with respect to a unitary change of that component (one meter and one radian respectively for the translational and rotational measures). The translational components are expressed in meters, while the rotational component is expressed in degrees. For any given value E on an axis, the actual amount of error on that axis is chosen randomly at startup according to a uniform distribution in the range $-E/2$ to $+E/2$.

- **Laser range**: defines the minimum and maximum distance within which the simulated laser range scanner is able to perceive objects, expressed in meters.
- **Laser field of view**: defines the overall symmetrical angle within which the laser range scanner is able to perceive objects, expressed in degrees.
- **Laser number of samples**: defines the number of measurements the laser range scanner is able to perform in a single scan, distributed over the entire field of view. The ratio between the laser's field of view and number of samples defines the laser angular resolution.
- **Type of drive**: defines whether the robot is of differential-drive type, i.e., it turns by moving its wheels at different speeds, or car-drive type, i.e., it turns by steering the wheels in the desired direction. Our experiments are conducted by simulating a differential drive robot.
- **Velocity limits**: defines the minimum and maximum velocity the robot is able to sustain. In our case, we assume the robot is able to sustain speeds up to 1.0 m/s.
- **Acceleration limits**: defines the minimum and maximum acceleration the robot is able to sustain. In our case, we assume the robot is able to sustain accelerations up to 1.5 $\mathrm{m/s^2}$.
- **Robot size**: defines the shape of the robot. In order to be able to safely traverse environments with narrow doors and corridors, we simulate a small square-shaped robot with sides measuring 0.3 m in length.
- **Laser position**: defines the exact mounting position of the laser range scanner on the robot's body. We assume the laser to be mounted in a centered position along the robot's perimeter at a height of 0.3 m.

A.1.1.2 Simulator configuration

The parameters related to the behavior of the simulator are:

- **Simulation interval**: this parameter, called `interval_sim` in the Stage configuration files, represents the amount of milliseconds between simulation steps. For our experiments, we use the default value of 100 ms.
- **Grid resolution**: represents the smallest unit of distance the simulation can handle, i.e., the virtual robot cannot distinguish any detail of the map smaller than this amount. In our context, we use a grid resolution of 0.025 m/pixel as a compromise between simulation accuracy and computational effort.
- **Environment path**: this parameter, called `bitmap` in the Stage configuration files, represents the path to the bitmap image of the floor plan of the simulated environment.
- **Environment size**: defines the size in meters of the simulated environment. It is defined as a tuple whose first and second component represent the width and height of the environment's 2D representation, respectively.
- **Initial robot pose**: defines the initial absolute position in meters of the virtual robot inside the simulated environment. A value of $0, 0$ represents the center of the simulated environment.

A.1.2 GMapping

- **Map size**: defines the boundaries of the rectangular area of the environment that the algorithm maps, with the center coinciding with the initial position of the robot, expressed in meters. It is controlled by four parameters: `xmin`, `xmax`, `ymin`, `ymax`. In our experiments, the maximum values are set to 100 m and the minimum values are set to -100 m.
- **Angular update**: defines the maximum amount of robot rotation, expressed in radians, after which the algorithm triggers the processing of a new laser scan. In our experiments, this parameter is set to 0.25 rad.
- **Linear update**: defines the maximum amount of robot translation, expressed in meters, after which the algorithm triggers the processing of a new laser scan. In our experiments, this parameter is set to 1.0 m.

- **Temporal update**: defines the maximum amount of time, expressed in seconds, after which the algorithm triggers the processing of a new laser scan. Negative values have the effect of disabling time based updates. In our experiments, this parameter is set to 5.0 s.

- **Map update interval**: defines the amount of time, in seconds, between two consecutive updates of the map. In our experiments, we leave this parameter set to its default value of 5.0 s; lower values allow a higher refresh rate of the produced SLAM map at the expense of a significantly increased computational load.

- **Number of particles**: defines the number of possible hypothesis on the state of the world that GMapping maintains at any given moment in time. In our experiments, we set this parameter to 40; higher values can potentially lead to increased SLAM performance and reduce the risk of particle depletion at the expense of a significantly increased computational load, while lower values can result in spurious mapping errors due to an insufficient amount of information being retained at each update step.

- **Map resolution**: defines the meters-to-pixels conversion ratio for the reconstructed SLAM map. In our experiments, we leave this parameter set to its default value of 0.05 m/pixel as a compromise between the quality of the produced map and the amount of computational resources, both in terms of processing power and of available memory, that are necessary to maintain it.

- **Maximum laser range**: defines the maximum range of the laser range scanner. According to the GMapping guidelines[1], this parameter should be set to match the maximum operational range of the sensor; in our experiments, we set its value to 60 m.

- **Maximum usable laser range**: defines the maximum usable range of the laser range scanner. Any beam is cropped to this value. As the accuracy of a laser range scanner decreases with distance, the GMapping guidelines suggest to follow a conservative approach and set its value to be lower than the maximum operational range of the actual sensor; in our experiments, we set this parameter to a conservative value of 30 m.

[1] `http://wiki.ros.org/GMapping`

- **Odometry error confidence**: defines how much the SLAM algorithm should be confident in the correctness of the odometry readings provided by the robot. It is controlled by four different components: `stt`, which defines the odometry error in translation as a function of translation; `str`, which defines the odometry error in translation as a function of rotation; `srt`, which defines the odometry error in rotation as a function of translation; and `srr`, which defines the odometry error in rotation as a function of rotation. Their default values are 0.2, for `srt` and `str`, and 0.1, for `stt` and `srr`; in our experiments, we decrease their values by an order of magnitude to reflect the relatively small margin of error of the odomety readings of our simulated robot.

A.1.3 Controller settings

- **Controller frequency**: defines the rate at which to run the control loop and send velocity commands to the robot. In our experiments, this parameter is set to 10 Hz.
- **Planner frequency**: defines the rate at which to run the global planning loop. If its value is set to zero, a new global plan is computed only when the robot reaches the current goal. To prevent the robot from getting potentially stuck following erroneous plans, we set this value to 0.2 Hz.
- **Controller patience**: defines the amount of time, in seconds, the controller waits a valid control for before attempting to perform space-clearing operations. In our experiments, we leave this parameter set to its default value of 15 s.
- **Planner patience**: defines the maximum amount of time, in seconds, the global planner spends to find a valid plan towards the current goal before attempting to perform space-clearing operations. In our experiments, we leave this parameter set to its default value of 5 s.
- **Maximum planning retries**: defines the number of times the global planner will attempt to find a valid plan towards the current goal before executing recovery behaviors. In order to limit the amount of time the global planner spends on potentially unreachable goals, we set this value to 3.
- **Velocity limits**: define the translational and rotational velocity ranges of the robot. In order to simulate the safety limitations that are imposed to wheeled robots in human-populated indoor environments,

we impose a maximum forward velocity of 0.85 m/s and a maximum angular velocity of 0.7 rad/s.

- **Acceleration limits**: define the translational and rotational acceleration ranges of the robot. In order to simulate the safety limitations that are imposed to wheeled robots in human-populated indoor environments, we impose a maximum linear acceleration of $1.5\,\mathrm{m/s^2}$ and a maximum angular acceleration of $2.5\,\mathrm{rad/s^2}$.
- **Minimum in-place rotational velocity**: defines the minimum rotational velocity, expressed in radians per second, allowed for the robot while performing in-place rotations. In our experiments, this value is set to 0.4 rad/s.
- **Escape velocity**: defines the linear velocity at which the robot should move backwards while trying to move away from an obstacle. As our robot is not equipped with a backward-facing laser range scanner, we limit the value of this parameter to -0.15 m/s for safety reasons, where the minus sign denotes the backward motion.
- **Escape reset distance**: defines the distance, expressed in meters, after which the robot considers itself sufficiently far from the obstacle that induced the escape behavior to resume its normal navigation operations. In our experiments, it is set to a conservative value of 0.15 m.
- **Holonomic mode**: this parameter is true if the robot is holonomic, i.e., if it can move laterally by using omnidirectional wheels. Since the robot we are simulating is non-holonomic, this parameter is set to false.
- **Oscillation reset distance**: defines the minimum distance the robot has to travel continuously in a given direction before it may consider moving in the opposite direction. In our setting, this parameter is set to 0.25 m.

A.1.4 Path planner settings

- **Goal tolerance**: defines the maximum distance and rotation difference from the desired goal position and orientation within which the controller considers the goal reached. In our experiments, we tolerate a distance margin of 0.3 m and a yaw margin of 0.2 rad.
- **Path distance scale**: controls the relative importance the local planner should give to staying close to the trajectory proposed by the

global planner in the evaluation of the cost of the actual trajectory to be followed. It is set to 0.6.

- **Goal distance scale**: controls the relative importance the local planner should give to the distance between the endpoint of the local trajectory and the true local goal for that intermediate planning step in the evaluation of the cost of the actual trajectory to be followed. It is set to 0.8.
- **Obstacle distance scale**: controls the relative importance the local planner should give to avoiding obstacles in the evaluation of the cost of the actual trajectory to be followed. It is set to 0.05.
- **Meter scoring**: if true, the path distance, goal distance, and obstacle distance values used in the evaluation of the cost of the trajectory to be followed are expressed in meters; if false, they are expressed in cells. In our setting, this parameter is true.
- **Simulation time**: defines the amount of time, in seconds, to forward-simulate trajectories in order to evaluate their cost. Higher values produce more accurate results at the expense of an increased computational effort. In our setting, this parameter is set to 1.5 m.
- **Simulation granularity**: defines the simulation step size between points on a given trajectory in meters. In our setting, it is set to its default value of 0.025 m as a compromise between simulation accuracy and computational effort.
- **Vx samples**: defines the number of samples to use when exploring the `x` velocity space to determine the intensity of the forward motion control signal the robot should send to the motors. Higher values produce more accurate estimates. In our setting, this parameter is set to 10.
- **Vtheta samples**: defines the number of samples to use when exploring the angular velocity space to determine the intensity of the rotation control signal the robot should send to the motors. Higher values produce more accurate estimates. In our setting, this parameter is set to 20.

A.1.5 Common costmap settings

- **Robot footprint**: defines the area that should be considered as occupied by the robot while performing path planning. To reduce the prob-

ability of collisions with obstacles, we set the footprint to be slightly larger than the true size of the robot, encompassing a square area with sides of 0.4 m.

- **Maximum obstacle height**: defines the maximum height an obstacle may have to be inserted into the costmap, in meters. As this parameter must be set to be slightly higher than the true height of the robot, we set its value to 0.6 m.

- **Obstacle range**: defines the maximum distance from the robot at which an obstacle may be inserted in the costmap, in meters. Considering the range of our laser range scanner, we set this parameter to 30 m.

- **Raytrace range**: defines the maximum distance in meters at which to raytrace obstacles. This parameter is also set to 30 m.

- **Inflation radius**: defines the radius in meters to which an obstacle cost is inflated by the path planner to include its neighboring cells. All cells that lie at a distance from the obstacle greater than this value are considered to have zero cost and can certainly be used for path planning; cells that have a non-zero cost may also be used for path planning, but preferring lower cost cells to higher cost cells to reduce the possibility of collisions. In our setting, this parameter is set to 2.25 m.

- **Cost scaling factor**: controls the rate at which the cost of the costmap cells exponentially decays as a function of the robot's distance from the obstacle. In our setting, this parameter is set to 7.5 m.

A.1.6 Global costmap settings

- **Global update frequency**: defines the rate at which the global costmap must be refreshed to reflect new observations of the laser range scanner. In our setting, this parameter is set to 2 Hz in order to limit the computational effort required to keep the costmap updated.

- **Global static map**: defines whether the costmap should be initialized with data coming from a map server. In order to use a SLAM algorithm, this parameter must be set to true.

- **Transform tolerance**: defines the maximum delay the costmap tolerates while processing incoming transform data, in seconds. If the

transform between the robot's reference frame and the costmap's reference frame takes longer than this amount, the information contained in the transform is considered to be outdated and is ignored. This parameter is set to 0.2 s.

A.1.7 Local costmap settings

- **Local update frequency**: defines the rate at which the local costmap must be refreshed to reflect new observations of the laser range scanner. In our setting, this parameter is also set to 2 Hz in order to limit the computational effort required to keep the costmap updated.

- **Local costmap size**: defines the area covered by the local costmap. The local costmap's center coincides with the robot's position and the costmap follows the robot using a rolling window approach. In our setting, both height and width are set to 6 m.

- **Local costmap resolution**: the resolution of the local costmap in terms of length of a cell's sides, expressed in meters. It is set to 0.02 m.

- **Transform tolerance**: defines the maximum delay the costmap tolerates while processing incoming transform data, in seconds. If the transform between the robot's reference frame and the costmap's reference frame takes longer than this amount, the information contained in the transform is considered to be outdated and is ignored. This parameter is set to 0.2 s.

A.1.8 Automatic exploration script

- **Seconds between map saves**: defines the frequency at which the exploration termination condition is verified by defining the amount of time, in seconds, between two consecutive map snapshots. In our experiments, we set this value to 600 s, except for very small environments (under 250 m^2) for which its value is 120 s.

- **Maximum number of map saves**: defines the maximum duration of an exploration run by capping the number of consecutive map snapshots that can be taken before the simulation is forcefully terminated. It serves as a safety procedure to prevent the simulation from stalling in case of particularly bad exploration failures. In our experiments, we set this value to 18.

- **Image similarity threshold**: defines the maximum level of difference between two consecutive map snapshots, as defined by the mean square error (MSE) metric, for the images to be considered identical. Whenever the difference between two consecutive map snapshots falls below this threshold, the exploration is considered complete and the simulation is stopped. In our experiments, we set this value to 10, which we empirically verified to be a good compromise with respect to the size of the reconstructed SLAM maps between the necessity to avoid premature terminations and the need to tolerate a certain level of noise.

- **Number of runs**: defines how many consecutive simulation runs should be performed for each environment of the batch. In our experiments, we start with an initial value of 10 runs as described in Section 4.2.2, which can then be increased to match the required sample size.

A.2 Feature extraction

In this section, we discuss the main parameters that affect the way in which the feature extraction module computes the environmental features that are used for model learning and prediction. In particular, we focus on the parameters controlling the computation and the exploration of the Voronoi graph for the elaboration of the Voronoi traversal distance and Voronoi traversal rotation environmental features.

A.2.1 Voronoi graph computation

The parameters affecting the computation of the Voronoi graph are:

- **Dilation kernel**: defines the size in pixels of the kernel that is used to dilate the segments of the bitmap Voronoi graph as part of the skeletonization process. In our experiments, we use a 5×5 pixels kernel.

- **Pass-through node detection distance**: defines the length in pixels of the alignment line that is used to verify whether two adjacent nodes in the bitmap Voronoi graph are part of a longer straight line or sit on a curve. The alignment line is centered on the two nodes and effectively represents an extension of the line that joins them. If at least one pixel in both the 3×3 pixels areas at the extremes of the alignment line is black, the two nodes are assumed to belong to a longer straight

line and one of them is removed as it represents a pass-through node; otherwise, the nodes are interpreted to be part of a curve and are both kept in the Voronoi graph. In our experiments, this parameter is set to 5 pixels.

A.2.2 Voronoi graph exploration

The parameters affecting the exploration of the Voronoi graph are:

- **Laser field of view**: defines the field of view of the simulated laser. To match the configuration used to perform data collection, we set this value to 270°.

- **Laser range**: defines the usable range in meters of the simulated laser. To match the configuration used to perform data collection, we set this value to 30 m.

- **Starting position**: defines the node of the Voronoi graph from which the exploration starts. This parameter is automatically set for each environment to coincide with the pose from which the exploration would start.

- **Scale**: defines the pixels-to-meters conversion factor. This parameter is automatically set for each environment to coincide with the resolution of the associated floor plan.

- **Minimum rotation distance**: defines the minimum distance between two nodes that the virtual robot has to travel before estimating the Voronoi rotation between the second and the first node. In our experiments, this parameter is set to 0.5 m to avoid misinterpreting imperfections in the pixels alignment as actual rotations of the robot, only accounting for rotations in the correspondence of curves in the simulated trajectory.

A.3 Model learning

In this section, we describe the most relevant parameters that control the behavior of the model learning module. These parameters affect the way in which the model learning module attempts to find correlations between the values of the available environmental features and the components of the localization error for the environments of the training set.

- **Model class**: allows the user to specify which of the three machine learning techniques described in Section 5.3 should be used for training. Therefore, its possible values are `Linear regression`, `Feature selection` and `ElasticNet`.

- **Linear regression features**: allows the user to specify the behavior of the model learning module when using the `Linear regression` model class. It can be set to an individual feature, in which case only the regression model using that feature as independent variable is trained, or to `override mode`, in which case the module trains a regression model for each of the available features.

- **Feature selection repeats**: defines the number of times for which the F-regression test is performed for each candidate number of features K to identify the best performing features of each set when using the `Feature selection` technique described in Section 5.3.2. In our experiments, this value is set to 100.

- **ElasticNet L1_ratios**: allows the user to specify a list of candidate values for the ratio controlling the relative importance of the L1 regularization term compared to the L2 regularization term when using the `ElasticNet` model. A `L1_ratio` of 0.0 means that only L2 regularization is used, while a `L1_ratio` of 1.0 means that only L1 regularization is used.

- **ElasticNet alphas**: allows the user to specify a list of candidate values for the `alpha` hyperparameter when using the `ElasticNet` model.