Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informazione e Bioingegneria
MSc in Computer Science and Engineering

Use of predicted layouts of indoor environments to improve exploration strategies for autonomous mobile robots

Supervisor:
prof. Francesco AMIGONI

Co-supervisor:
dr. Matteo LUPERTO

MSc thesis by:
Luca FOCHETTA
Id number: 864463

Accademic Year 2017-2018

Abstract

An important field of study that has been addressed over the last few years is autonomous mobile robotics.

One of the main tasks that an autonomous mobile robot is required to perform is *exploration*, that amounts to incrementally discover an initially unknown environment in order to build its map. During an exploration, an autonomous robot perceives a portion of the environment with its sensors, integrates its sensor readings within its current map, identifies the boundaries between known and unknown portions of the environment, evaluates such boundaries according to an utility function in order to select one of them according to an *exploration strategy* and, finally, reaches it.

In this thesis, we propose a multi-criteria exploration strategy that uses the current map that has been built during the exploration (which represents the explored part of the environment) to predict the layout of the unknown part of the environment and that exploits this knowledge to enhance the exploration process. We evaluate the performance of our system in different simulated indoor environments, obtaining a consistent speed up at the end of the exploration process.

Sommario

La robotica mobile autonoma è un ambito di ricerca che ha ricevuto molta attenzione negli ultimi anni. Uno dei compiti principali che un robot mobile autonomo deve essere capace di compiere è l'*esplorazione*, definito come la scoperta incrementale di un ambiente inizialmente sconosciuto. Un robot mobile autonomo durante l'esplorazione usa i sensori per percepire l'ambiente, integra i dati raccolti in una mappa, identifica le frontiere tra le porzioni di mappa esplorata e le porzioni di mappa inesplorata, valuta le frontiere utilizzando una funzione di utilità e ne seleziona una utilizzando una *strategia di esplorazione*, e infine la raggiunge.

In questa tesi proponiamo una strategia di esplorazione multicriterio che usa la mappa costruita durante l'esplorazione (che rappresenta la parte esplorata dell'ambiente) per predirre il layout dalla parte sconosciuta dell'ambiente e che sfrutta questa conoscenza per migliorare il processo esplorativo. Valutiamo le prestazioni del nostro sistema in molteplici ambienti interni simulati, mostrando che migliora il processo esplorativo.

Ringraziamenti

Sembra ieri che abbia iniziato il mio primo giorno al Politecnico di Milano, ed ora eccomi qua, a coronare un percorso che mi ha dato tanti piaceri e qualche delusione, ma che oggi, sono molto contento di avere iniziato, e che rifarei. Questo percorso non l'ho mai fatto da solo, ma ho sempre avuto persone al mio fianco che mi hanno aiutato a percorrerlo.

Innanzitutto, devo ringraziare molto il Prof. Francesco Amigoni e il Dott. Matteo Luperto, che mi hanno accompagnato lungo tutto il percorso della tesi, aiutandomi e assistendomi. Li ringrazio anche per avermi fatto maturare sotto molti aspetti e per la dedizione e la pazienza che mi hanno dedicato durante tutta la tesi, soprattutto durante la parte finale dove mi hanno accompagnato in quella che è stata una vera e propria corsa contro il tempo.

Ringrazio i miei genitori, che mi hanno accompagnato e sostenuto e che sono sempre stati al mio fianco nei momenti in cui sono arrivate soddisfazioni, ma soprattutto per il sostegno che mi hanno mostrato quando sono arrivate le delusioni. Li ringrazio per avere sempre sostenuto le mie scelte. Ringrazio mio fratello, che mi ha sempre dato una mano quando mi ha visto in difficoltà, e che mi ha sempre aiutato al massimo delle sue capacità.

Ringrazio i miei amici Alessandro, Manuel e Stefano per avermi sempre incoraggiato a non mollare e per i tanti momenti felici vissuti insieme. Nonostante il passare del tempo siamo ancora qua a discapito di tutte le volte che avrei potuto darvi dei motivi per non farlo, e per questo vi sarò eternamente riconoscente. Ringrazio Camilla, Edoardo, Laura, Marcello, Dino, Matteo e Riccardo per essermi sempre stati vicini e per le tantissime serate passate a divertirci, a mangiare fino a scoppiare e a tirarci su l'un l'altro. Per questo e tante altre cose vi ringrazio.

Ringrazio Erco, Zane e tutti gli altri del Calcetto Sabatale per le tante serate passate a giocare a calcetto e a cercare di ammazzarci a vicenda, che mi ha sempre aiutato a distrarmi e a tirarmi su.

Ringrazio gli amici della Calabria, che pur non sentendoci e vedendoci spesso, so che potrò sempre contare su di loro.

Ringrazio i miei parenti, per avermi sempre sostenuto e aiutato a tenere duro.

Infine, dedico il mio ringraziamento più affettuoso alla mia ragazza Martina, che mi è stata sempre accanto per aiutarmi, incitarmi e a darmi una mano tutte le volte che pensavo di non farcela, e che è troppo modesta e troppo buona per rendersene conto; probabilmente non ce l'avrei mai fatta senza il tuo aiuto.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An important field of study that has been addressed in the last few years is autonomous mobile robotics. An autonomous mobile robot must have the capability to make choices to accomplish a task, perform actions trough *actuators* (like motors), and perceive the environment with *sensors* (like cameras) without any human intervention.

One of the main tasks that an autonomous mobile robot is required to be capable of performing is *exploration*, that involves collecting data from sensors to incrementally build maps of initially unknown, or partially known, environments. The maps represent the positions of obstacles, like walls, and of the free space. During the exploration, an autonomous mobile robot perceives a portion of the environment with its sensors, integrates sensor readings within its current map, identifies the boundaries between known and unknown portions of the environment, evaluates such boundaries according to a utility function, selects one of them according to an *exploration strategy* and, finally, reaches it. The selection of the most promising next location according to some criteria, like the distance from the current position of the robot, isù the main component of an exploration strategy.

Most of the methods that have been proposed for implementing exploration strategies are based on the concept of *frontiers*, defined as boundaries between free space and unexplored space [35]. The most simple exploration strategy is presented in [34], where the robot moves from its current posi-

tion to the closest frontier. A similar approach is the one proposed by the authors of [13]: they propose to generate a set of candidate locations and to evaluate them using both the distance of a candidate location from the current position of the robot, and the expected amount of information the robot can perceives from the candidate location using its sensor. This exploration strategy introduces the concept of using more than one criteria to evaluate the candidate locations. This type of approach is called *multi-criteria decision making*. The authors of [4] give a formal definition of this approach and propose a framework that exploits the dependencies between different criteria.

In the literature, some authors proposed exploration strategies that try to predict what lies in the unknown part of the environment. For example, the authors of [30] use a database of previously seen maps to improve the exploration of Roman catacombs. Instead, the authors of [5] use a convolutional neural network to predict the location of an emergency exit on the map.

The purpose of this thesis is to develop and implement a multi-criteria exploration strategy that is capable of using the map built during the exploration (which represents the explored part of the environment) to make predictions about the possible shape and structure of the unexplored part of the environment, identifying its *layout*, defined as its abstract geometrical representation. We implement our exploration strategy considering as criteria the distance of each candidate location from the robot position and the expected area that can be seen according to the predicted layout.

We test our exploration strategy in 10 simulated indoor environments, and we evaluate it in terms of explored area, time spent, distance travelled by the robot, and the capability to select a candidate location that is actually reachable from the robot. In order to evaluate its performance, we make a comparison with the algorithm presented in [13]. Results show that our proposed exploration strategy is able to improve exploration performance, especially at the end of the exploration missions.

The thesis is structured as follows.

In Chapter 2 we describe the state of the art regarding the exploration strategies proposed in the literature, focusing on exploration strategies that make predictions over the unexplored part of the environment. Moreover, we illustrate some algorithms proposed in the literature that use the map to extract more abstract knowledge.

In Chapter 3 we describe formally the problem addressed in our work, starting from a description of what the exploration problem is and then by discussing the possibility of using prediction over the unseen environment to solve it.

In Chapter 4 we describe our solution to the problem described in Chapter 3 illustrating the basis of our work.

In Chapter 5 we describe the implementation of our system, providing details about the software frameworks used and about its architecture.

In Chapter 6 we present the results obtained by applying our method in simulation on 10 indoor environments. We evaluate our solution, then we compare it to a method from the literature [13].

In Chapter 7 we summarize the purpose and the results obtained in our work. Then, we propose some future developments for our work.

Chapter 2

State of the art

For autonomous mobile robots, *exploration* is an important task where the robot has to visit an initially unknown, or partially known, environment in an autonomous way and incrementally build the map using the data collected by its sensors during the process [32].

The task of incrementally build the map of the environment, localize the robot in the map, and choose the next best location that has to be visited during the exploration is called the *exploration problem*. Many studies about the exploration problem are based on the assumption that no knowledge of the environment is available at the begin of the exploration [13, 34]. Different *exploration strategies* have been proposed to solve the exploration problem in an efficient way. The different exploration strategies are usually tested in a given environment and their performance are usually compared on the basis of time or distance travelled by the robots.

In order to increase the performance during the exploration, some studies are focused on the exploitation of some more abstract and conceptual knowledge, like identifying the walls in the environment or categorize the environment seen so far into specific categories like rooms [21].

In this chapter we present an overview of the state of the art related to both the exploration problem and the collection of knowledge from the environment in order to predict specific elements in the unknown part of the environment. In Section 2.1 we describe the sensors and perception

models used to acquire data from the environment; in Section 2.2 we describe the issue of integrating the collected data from the environment to create the map and localize the robot. In Section 2.3 we introduce some of the most important patterns that deal with the exploration problem. Finally, in Section 2.4 we discuss about some proposed methods (that are not used to solve the exploration problem) to make predictions about the unknown part of the environment.

2.1 Sensor

In order to be completely autonomous, the robot must be able to sense the environment. The robot's sensors perceive the environment conditions and transduce the perception in signals that can be processed by the robot. The selection of sensors is strictly dependant of the environmental information the robot needs to acquire [8]. The most important sensors related to our application field are:

- Laser range finders: also called *LIDARs*, they use pulsed laser light to measure the distance between target object and robot.

- Vision sensors: monocular or stereo cameras used to estimate the 3D structure of the target.

- RGB-D Sensors: a combination of depth sensors and RGB cameras, associate to the RGB image a depth channel relating each pixel to a distance between the image plane and the corresponding object.

- Inertial Measurement Sensors (*IMUs*): they measure accelerations and inclination of the robot.

- GPS: measures the position of the robot.

2.2 SLAM problem

One of the main and most difficult tasks for mobile robots is to build the map of the environment and to localize the robot in it while building it. This task is called *Simultaneous Localization and Mapping (**SLAM**)*. The ***SLAM*** problem [14, 32], is considered to be a complex problem since robot's localization requires a consistent map, which in turn requires a good estimation of the robot position. A more formal definition about the SLAM problem is the following:

$$x_{1:t} = x_1, ..., x_t\text{: trejectory of the robot} \tag{2.1}$$

$$z_{1:t} = z_1, ..., z_t\text{: observations from sensor} \tag{2.2}$$

$$u_{1:t} = u_1, ..., u_t\text{: odometry measurements} \tag{2.3}$$

$$m_t\text{: map} \tag{2.4}$$

The *SLAM problem* involves the calculation of the joint posterior probability on the map m and the trajectory $x_{1:t}$ given from the sensor's observation $z_{1:t}$ and the odometry measurements $u_{1:t}$:

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

The basic principle applied to solve the equation (2.5) is the *Bayes rules* and the scheme used to integrate such temporal data is known as the *Bayes filter*. Bayes filter extends the concept of Bayes rules to temporal estimation problems. It is a recursive estimator for computing sequence of a posterior probability distribution. The Bayes filter for the equation (2.5) is:

$$\begin{aligned} p(x_{1:t}, m_t|z_{1:t}, u_{1:t}) = \alpha p(z_t|x_t, m_t)\cdot \\ \cdot \iint p(x_t, m_t|u_t, x_{t-1}, m_{t-1}) p(x_{t-1}, m_{t-1}|z_{t-1}, u_{t-1})\, dx_{t-1}\, dm_{t-1} \end{aligned} \tag{2.6}$$

In the equation (2.6) α is used as a normalization factor in order to ensure that the left hand side of the formula is a probability distribution. If the environment is static, the time index t can be omitted when referring to the

map m. Then the formula 2.6 becomes:

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = \alpha p(z_t|x_t, m) \int p(x_t|u_t, x_{t-1}) p(x_{t-1}, m|z_{t-1}, u_{t-1})\, dx_{t-1} \tag{2.7}$$

As can we see from formula 2.7 the estimator does not require the integration over map m any more, greatly simplifying the integral. The generative model are $p(x_t|u_t, x_{t-1})$ and $p(z_t|x_t, m)$, respectively for pose x_t and observation from sensor z_t.

2.2.1 Rao-Blackwellized Particle Filter

Many algorithms are being proposed to solve the formula 2.7 in an efficient way, one of the most used is the **Rao-Blackwellized Particle Filter** (**RBPF**) [14]. The algorithm is based on the concept of *particle* which represents a potential pose of the robot. To each particle an individual map is associated; the maps are built from the observations and the trajectory from the corresponding particle. The Rao-Blackwellized can be divided into four steps:

- *Sampling*: The next generation of particles x'_t is obtained from x'_{t-1} by sampling from a proposed distribution π. The most used distribution is a probabilistic motion model.

- *Importance weighting*: To each particle an importance weight w'_t is assigned as follow:

$$w'_t = \frac{p(z_t|m'_{t-1}, x'_t) p(x'_t|x'_{t-1}, u_{t-1})}{\pi(x_t|x'_{1:t-1}, z_{1:t}, u_{1:t-1})} \cdot w'_{t-1} \tag{2.8}$$

  The weights take into account that the distribution π of the previous step $(t-1)$ could be different from the one of the following step(t).

- *Resampling*: Particles are chosen with replacements based on weights. This is done since only a finite numbers of particles is used to approximate a continuous distribution. The resampling allow the application

of a particle filter in situations where the target distribution is different from the proposal distribution π.

- *Map estimation*: For each particle, the corresponding map is estimated based on the trajectory x'_t and on the history of observations $z_{1:t}$.

Finally, the particle with the highest probability p and the corresponding map m are chosen.

2.2.2 Map representation

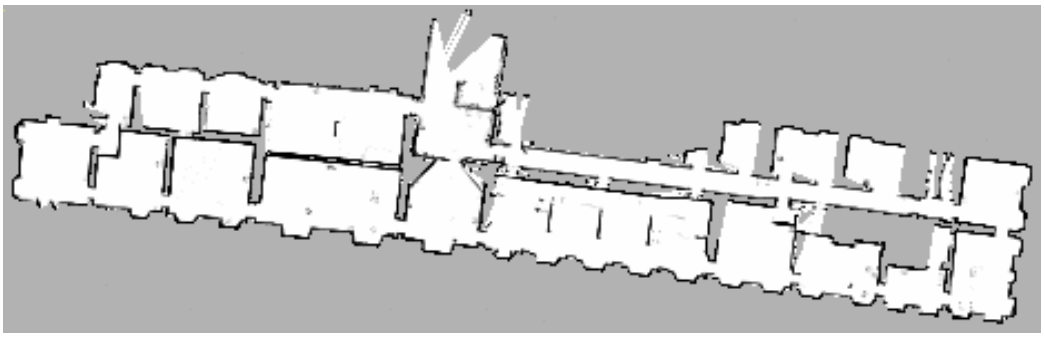
Autonomous robots require to be able to acquire and maintain a main representation of the environment in order to perform different operations on it. The map can be both known a priori or built with *SLAM*. Many representations have been proposed, the most commonly used are:

- Grid-based maps: The map is represented as a collection of cells, as a matrix, where each cell has a value that measures the probability to find an obstacle in that cell [33]. High values correspond to higher probabilities to find an obstacle, low values correspond to lower probabilities to find an obstacle. An example can be seen in Figure 2.1a.

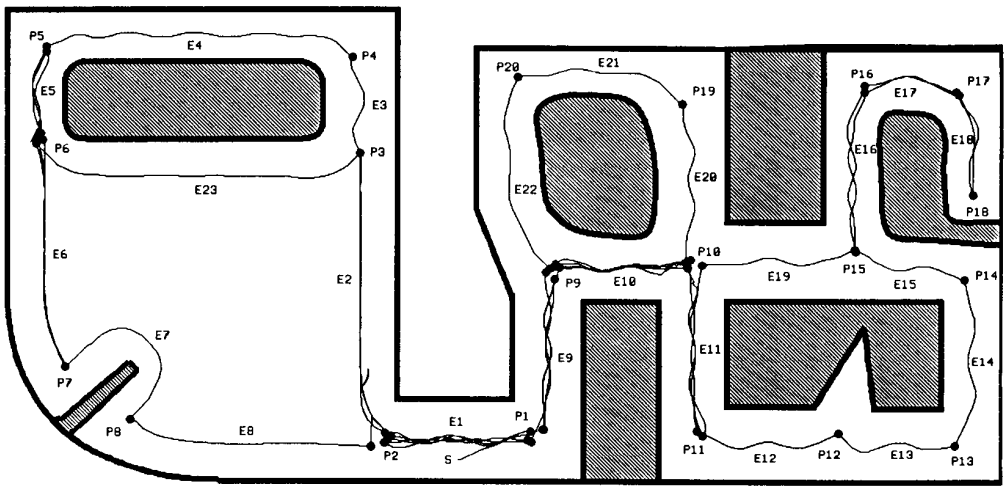
- Topological maps: The map is described as a graph. Each node corresponds to a different place(or different landmarks) and edges represent the existence of a path with two different places [33]. An example can be seen in Figure 2.1b.

- Coverage maps: Similar to the grid-based maps, the value attached to each cell represents the percentage of that cell being covered by an obstacle [29]. An example can be seen in Figure 2.1c.

- Feature-based: The environment is represented as set of line segments and corner points. An example can be seen in Figure 2.1d.



(a)



(b)

(c)

(d)

Figure 2.1: Figure 2.1a Occupancy grid map. Figure 2.1b Topological map. Figure 2.1c Coverage map. Figure 2.1d Feature based map. From [17, 29].

2.3 Exploration strategies

The objective of this section is to introduce different *exploration strategies* and some common patterns. For brevity, only the exploration strategies that

involve a single robot will be illustrated. This is done by illustrating the main basic exploration strategies in Section 2.3.1, by describing the Multi-criteria exploration strategies in Section 2.3.2 and finally in Section 2.3.3 by illustrating the predictive algorithms.

2.3.1 Basic exploration strategies

Frontier-based exploration strategies

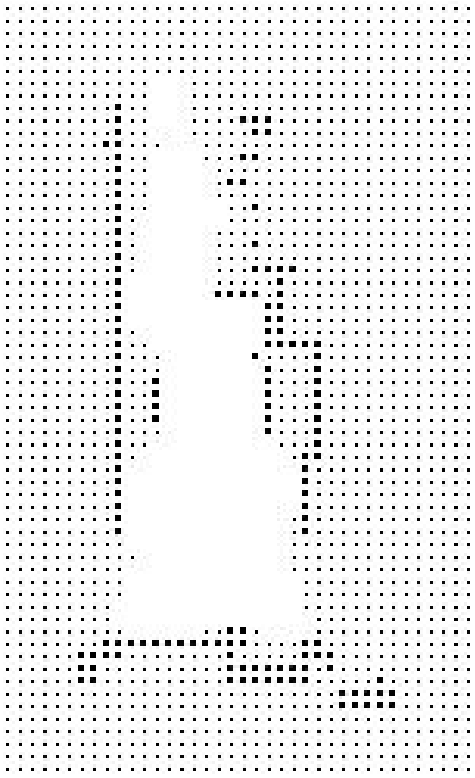
A frontier-based exploration approach uses the concept of *frontiers*, defined as the geometrical boundaries between known and unknown parts of the environment. Firstly introduced by [35], a frontier-based approach uses grids map cells where each cell stores the probability that the corresponding area is occupied by an obstacle or it is free. At the beginning of the exploration all the cells are set to the prior probability of occupancy. The cells are continuously updated using the sensors readings. Once the cells are updated they are classified comparing their actual value to their initial probability assigned at the beginning of the exploration. The classification is performed as following:

- open: if occupancy probability $<$ prior probability
- unknown: if occupancy probability $=$ prior probability
- occupied: if occupancy probability $>$ prior probability

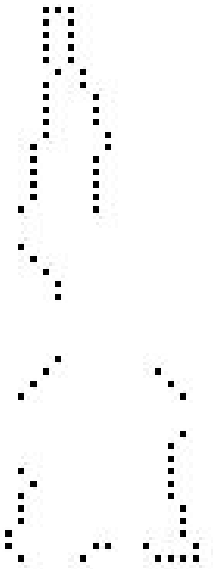
Each open cell that is next to an unknown cell is classified as an edge cell; all the edge cells adjacent to each other are grouped together into a frontier (Figure 2.2). Some threshold can be introduced to filter frontiers (e.g., a minimum number of cells).

Greedy Algorithm

The authors of [34] propose an algorithm that chooses as next exploration target the frontier that is closest to the robot position. They demonstrate that the robot is able to correctly explore all the environment using such a policy.



(a) *Grid Map*



(b) *Frontiers*

Figure 2.2: Figure 2.2a represents a map divided in cells, Figure 2.2b represents the frontiers extracted from 2.2a. From [35].

Next-Best-View Algorithm

The authors of [13] propose to choose the next target position by evaluating each candidate position based on the area of the unknown part of the environment that can be seen from the corresponding frontier. Each candidate position q_k is evaluated as follows:

1. Given map m_k (Figure 2.3a), starting position s_k, position q_k to evaluate and sensor's range r, generate randomly a number n of samples position $q'_{i,k}$ close to q_k (Figure 2.3b).

2. For each $q'_{i,k}$ calculate the area the robot can perceive with its sensor from $q'_{i,k}$ and draw the estimated area in map $m'_{i,k}$.

3. Merge all the $m'_{i,k}$ in a single map m'_k and then make the intersection

with the original map m_k and obtain the estimated area A(q_k) (Figure 2.3c).

4. Calculate the path $L(q_k)$ from position s_k to position q_k (Figure 2.3d).

5. Calculate the utility value g(q_k) with the following function:

$$g(q_k) = A(q_k)\exp(-\lambda L(q_k)) \text{ with } \lambda \text{ constant value} \qquad (2.9)$$

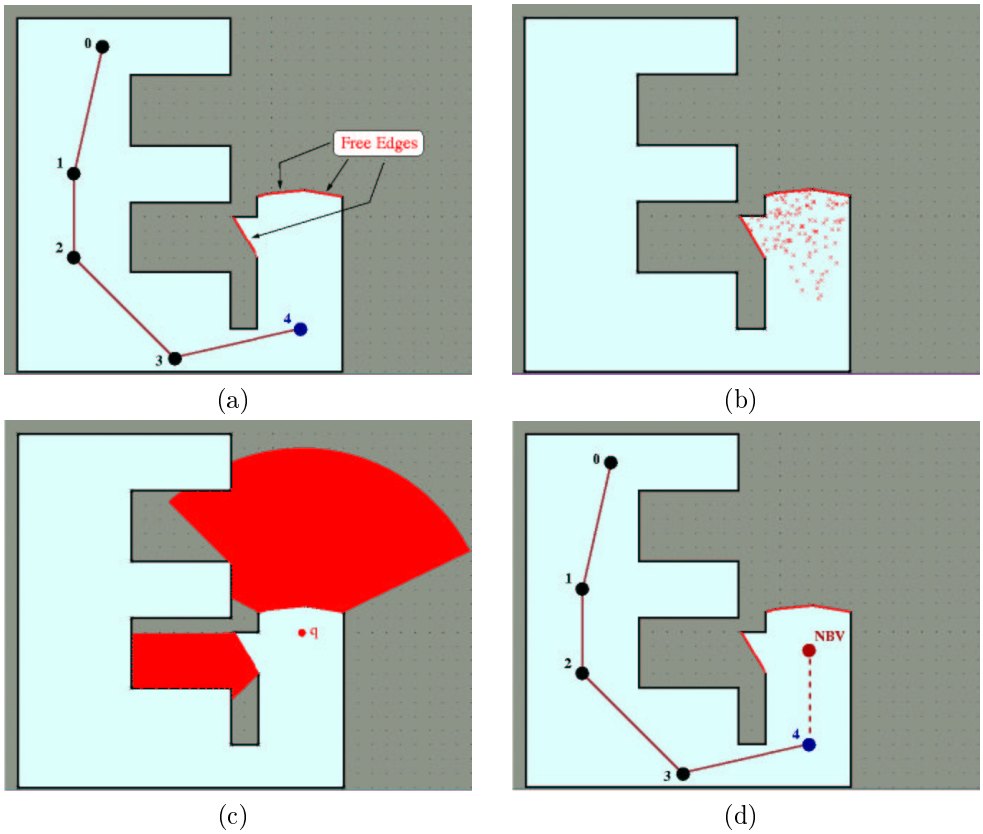


Figure 2.3: Figure 2.3a shows map m. Figure 2.3b shows the selection of $q'_{i,k}$ around q_k (Step 1). Figure 2.3c shows the result of steps 2, 3. Figure 2.3d shows the result of step 4.

The parameter λ is used to weight the cost of a motion against the expected gain in information. Finally, the destination d is selected as the q_k that maximizes the utility value.

2.3.2 Multi-criteria Strategies

This section illustrates some new exploration strategies that, similarly to the Next-Best-View algorithm presented in Section 2.3.1, use more than one criteria to evaluate each candidate position. The multi-criteria strategies are:

- *Multi-Criteria Decision Making (MCDM)*: this strategy exploits the correlation between different criteria in order to evaluate the candidate positions.
- *multi-objective exploration strategies*: this strategy evaluates the candidate positions without combining the different criteria in a single utility function.

Multi-Criteria Decision Making

Many studies on exploration strategies use ad hoc combinations between the different criteria selected. The *Multi-Criteria Decision Making*, **MCDM** [4] proposes a more formal definition in order to better exploit the possible correlations between the different criteria used.

Problem Formalization Given a set $N = \{1, 2, ..., n\}$ of criteria, a set of candidate positions C, given a candidate $p \in C$, MCDM denotes with $u_i(p) \in I$ its utility with the criteria $i \in N$, where $I \subseteq \mathbb{R}$ represents the set of possible utility values. Each candidate position p can be associated to a vector of n elements $u_p = (u_1(p), u_2(p), ..., u_n(p))$. The selection of a candidate position is performed using a *global utility function* $u(p) = f(u_p)$. The solution should be in the *Pareto frontier* that is defined as the subset $P \subseteq C$ such that $\forall p \in P \nexists q \in C : \forall i \in N, u_i(q) > u_i(p)$.

Choquet integral A total function μ is introduced and defined as $\mu : P(N) \rightarrow [0, 1]$ with the following properties:

- $\mu(\emptyset) = 0$
- $\mu(N) = 1$

- $if A \subseteq B \subseteq N, then\ \mu(A) \leq \mu(B)$

That is, μ is a normalized *fuzzy measure* on the set N. μ defines the dependency relations for each group of criteria. Given a group $G \subseteq N$ the criteria are:

- redundant if $\mu(G) < \sum_{i \in G} \mu(i)$
- synergic if $\mu(G) > \sum_{i \in G} \mu(i)$
- independent otherwise

Then the global utility $f(u_p)$ for a candidate p is calculated using the Choquet integral with respect to the fuzzy measure μ using p's utilities:

$$f(u_p) = C(u_p) = \sum_{j=1}^{n} (u_j(p) - u_{j-1}(p)) \cdot \mu(A_j) \tag{2.10}$$

Where u_j is the utility of the j-th criterion, after the utilities have been sorted in ascending order such that, for candidate p:

$$u_1(p) \leq ... \leq u_n(p) \leq 1 \tag{2.11}$$

Considering $u_0(p) = 0$. The set A_j is defined as:

$$A_j = \{i \in N | u_j(p) \leq u_i(p) \leq u_n(p)\}; \tag{2.12}$$

Multi-objective exploration strategies

The multi-objective exploration strategies proposed in [2] consider three different features to characterize a candidate position p:

- The travelling cost
- The information gain
- The precision of the localization of the robot

Differently from other approaches, the three features are not combined in a single utility function but are kept separated.

Map representation The map is stored into two list of line segments, that represent the obstacles and the boundary of the unexplored area. The first list, also called *obstacle list*, stores the line segments representing the edges of the observed obstacles. The second list, also called *free edge list* stores the line segments that separate the explored area from the unexplored area.

Candidate evaluation Given a candidate point p, the features are evaluated as follow:

- Travelling cost $c(p)$ is calculated as the total length of the segments forming the path that connects the robot's current position to p.
- Information gain $i(p)$ is evaluated as the total length of the segments in the free edges list that are visible from p.
- Overlap $o(p)$ between the new sensorial data and the information already stored is calculated as the total length of the line segments visible from the position p.

Proposed exploration strategies The algorithm operates in three steps:

1. Candidate positions are randomly generates along the free edges of the map.
2. The Pareto-optimal solutions are selected.
3. Since more than one Pareto-optimal solution exists, the position is selected as the one that is nearest to the ideal position, according to the following distance function:

$$D(p) = \sqrt{(c(p) - c_m)^2 + (i(p) - i_m)^2 + (o(p) - o_m)^2} \tag{2.13}$$

where p is the candidate position and c_m, i_m and o_m are, respectively, the minimum travelling cost, the maximum information gain and the maximum overlap obtained from the individual optimization.

2.3.3 Predictive Methods

A challenging problem in robotics is to integrate other forms of knowledge with the information gathered from maps. The works of [30, 31] propose to use datasets of previously seen environments and previously seen areas in order to exploit similarities with the area around the current frontier. The works of [5] uses the building floor plan and a convolutional neural network to generate a prediction of the position of an emergency exit, then the algorithm selects the frontier closest to the predicted position of an emergency exit.

P-SLAM

This algorithm proposes a new strategy to approach the problem of *SLAM*: it exploits common patterns found in the environment to speed-up the reconstruction of the map and at the same time makes it more accurate. The *Predictive - Simultaneous Localization and Mapping*(**P-SLAM**) [7] focuses on the unexplored regions closest to the explored ones because they are the next exploration targets. If a similar environment/structure is matched in the map of explored regions, an hypothesis is generated and the robot can decide to not explore that region and use the prediction as a virtual map and save exploration time. The algorithm is implemented using a Rao-Blackwellized particle filter, as described in Section 2.2.1. The prediction process can be divided into four major steps:

1. Locates a target frontier cell.

2. Collect structure informations near the target region.

3. Search for similar structures in the previously built map.

4. Generate an hypothesis if a similarity match exist.

An example of the P-SLAM structure is shown in Figure 2.4. We will now discuss in depth the four steps.

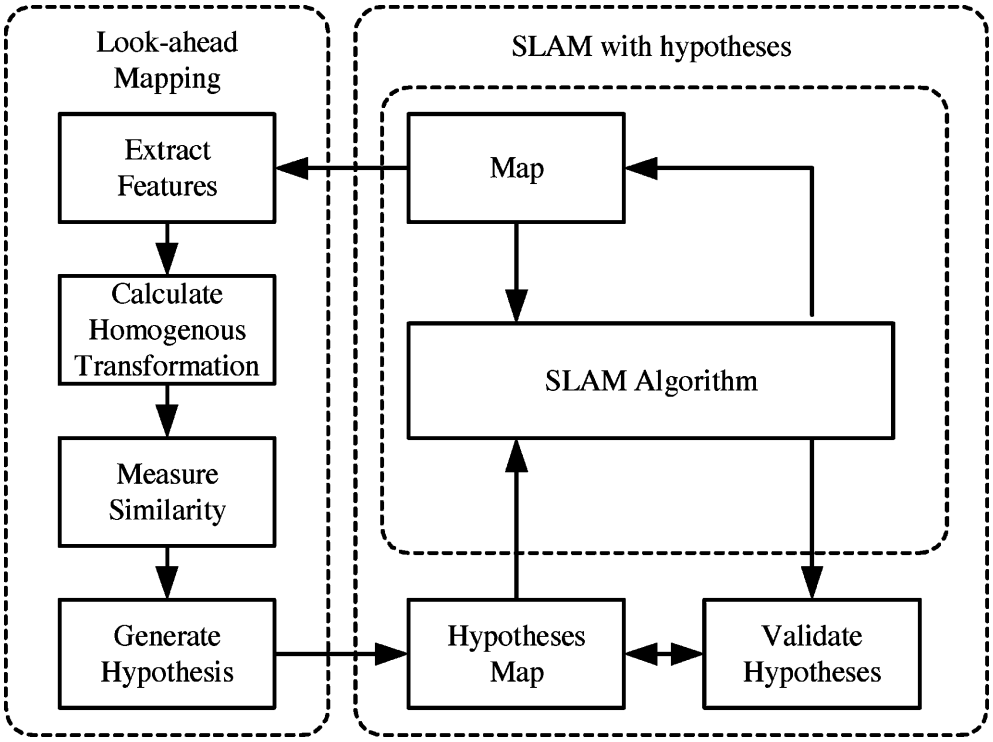
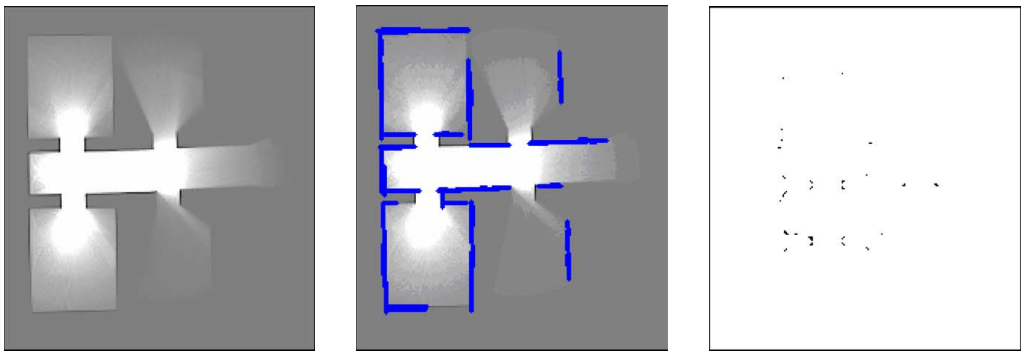


Figure 2.4: The figure shows the P-SLAM structure. From [7].

Target frontier choice In order to avoid useless predictions that will slow down the process, the prediction is applied only to the next exploration target which must be a frontier.

Collect structure information The features extracted from regions are lines and corners (Figure 2.5); they are extracted using respectively *Hough transform* and image gradients. It is not necessary that both features are present, as long as the information provided are useful to align the target and the reference region. The next step is to use the extracted features to obtain a pair of control points, where one point is in the target region and the other is in the reference region. With the pair of control points, an homogeneous-coordinate-transformation matrix is formed to align the control points. Since a corner feature is formed by two line segments, a control point usually has two angles (Figure 2.6). Consequently there are four different θ angles for alignment. For each θ an homogeneous-coordinate-matrix O is

built. The transformation matrix O and the pair of control points are used to obtain a reference cell f' with an aligned reference region $C_{f'}$. If the selected target cell is not the controlled point, the nearest corner feature in a preset range is selected. If no corner is found the prediction stops.



(a) (b) (c)

Figure 2.5: The figure shows the features extraction. Figure 2.5a shows the original occupancy grid. Figure 2.5b shows the extracted line features. Figure 2.5c shows the extracted corner features. From [7].

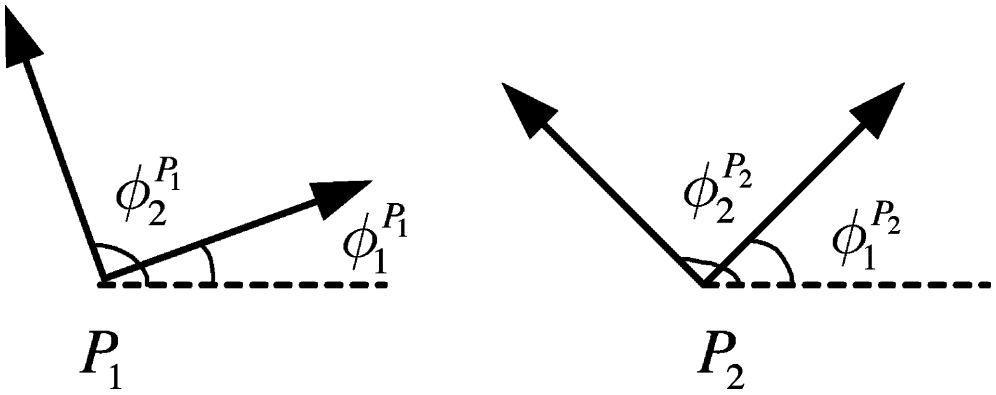


Figure 2.6: The figure shows control points P_1 and P_2 extracted from corner features. The angles are extracted from the line features. From [7].

Similar structure research Once a reference region $C_{f'}$ and a target region S_f are found, the similarity between them is calculated. As similarity

measure, the occupied cells are considered using the following filters:

$$I(i,j) = \begin{cases} 1, & \text{if } S_f(i,j) \text{ and } C_{f'}(i,j) \text{ are occupied} \\ 0, & \text{otherwise} \end{cases}$$

$$J(R,i,j) = \begin{cases} 1, & \text{if } R(i,j) \text{ is occupied} \\ 0, & \text{otherwise} \end{cases}$$

where $R(i,j)$ is an input region and i,j the position index of the cell. Then the similarity measure is calculated as follow:

$$\Psi(S_f, C_{f'}) = \frac{2 \cdot \sum_{i=1}^{d_s} \sum_{j=1}^{d_s} I(i,j)}{\sum_{i=1}^{d_s} \sum_{j=1}^{d_s} J(S_f,i,j) + \sum_{i=1}^{d_s} \sum_{j=1}^{d_s} J(C_{f'},i,j)} \tag{2.14}$$

where d_s is the size of the input region.

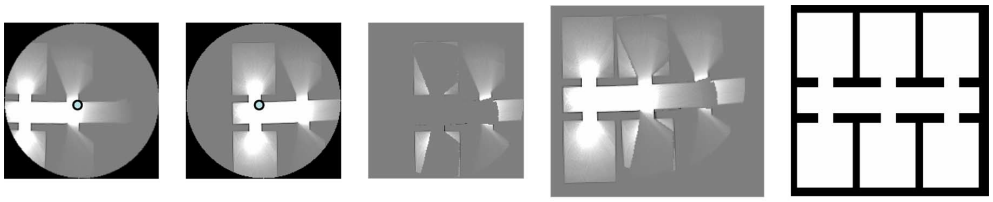


Figure 2.7: The figure shows all the steps. (a) The selected target region. (b) The found reference region. (c) The generated hypothesis for the selected region. (d) The result of the merge operation. (e) The simulated environment. From [7].

Hypothesis generation If equation (2.14) for at least one reference region is over a threshold, a hypothesis is generated and the transformation matrix is used to align the reference region and the target region. In case there are more than one reference region with similarity measure higher than the threshold, the one with highest similarity measure is selected. The generated hypothesis then is merged into the built map.

Algorithm of Ström, Nenci and Stachniss

Differently from [13], where the frontier is selected based on the distance and the potential area that can be seen, [30, 31] propose a different approach: since most robot maintain a probabilistic belief about the pose and the map of the environment, an alternative approach is to select the target location as the one that minimizes the uncertainty in the belief of pose and map of the robot. The selection is performed exploiting the concept of loop-closure: it is defined as the act of asserting that a robot has returned to a previously visited location [15]. A robot that is able to correctly recognize a loop-closure is able to reduce the uncertainty about its pose. The loop-closure is predicted using a database of previously seen maps (both from the same environment at earlier stages of the exploration and from different environments) used to search for similarities with the current map.

Information-driven exploration Defining a as the action performed at time t, z the sequence of observations obtained, the information gain (also called mutual information) of taking action a is defined as the change in entropy of the belief about the robot's pose X and the map M:

$$I(X, M : Z_a) = H(M, X) - H(M, X|Z_a) \tag{2.15}$$

With $H(M, X|Z_a)$ conditional entropy defined as:

$$(H, X|Z_a) = \int p(z|a)H(M, X|Z_a = z)\, dz \tag{2.16}$$

Unfortunately, equation (2.16) is intractable due to exponential growth of potential measurements.

Utility function The utility function is the following:

$$U(a) = I(M, Z; Z_a) - cost(a) \tag{2.17}$$

Where $cost(a)$ is the path length from current position of the robot to the designated target location of a. The best action a^* is the one that maximizes $U(a)$. As described in Section 2.3.3, the computation of $U(a)$ can become very expensive, so if we assume that the action a can reduce the uncertainty about the map, $U(a)$ becomes:

$$U(a) = I_{map}(a) + I_{traj}(a) - cost(a) \tag{2.18}$$

Another assumption could be that all the frontiers yield the same expected information gain about the uncertainty on the map. So the utility can be approximated to:

$$U(a) = I_{traj}(a) - cost(a) \tag{2.19}$$

The expected information gain about I_{traj} is mainly influenced by loop-closure.

Querying for similar structures The prediction of how the environment may look like is done by making a similarity query between the area near the frontier f and the structures explored before (from the same run or from previous run). This is done by creating a database storing all local grid maps, and then the query is done by checking image similarities with **FabMAP2**, that is an appearance based approach to efficiently query the database (Figure 2.8). A helpful tool of FabMAP2 is that it also provides a likelihood $l(m)$ for each match m.

Figure 2.8: The Figure shows as left image the query map, the other images are the four matches in the database. From [30].

Loop-closure prediction Once the query with FabMAP2 provides all the matches m, the matches are converted into the corresponding Voronoi graphs,

and finally aligned with the actual map using a RANSAC-based algorithm. The next step is to search for loop-closures using the generalized Voronoi graph (Figure 2.9). If the graph leads to a position close to any other frontier in the map, then it is a possible loop-closure.

Figure 2.9: The figure illustrates the loop-closure prediction. Left: map explored so far with the frontier under consideration(blue dot). Middle: One of the predictive belief(red) superimposed on the map. Right: Voronoi diagram. From [30].

Estimating the probability to close a loop Each map reported by FabMAP2 has an associated likellihood $l(m)$. Then the probability of closing a loop is:

$$S_f = \sum_{m \in M(f)} l(m) \cdot \sum_{c \in C(f,m)} l(c|m) \tag{2.20}$$

where:

- $M(f)$ is the set of matches returned from FabMAP2 with f the frontier selected for the query.
- $C(f, m)$ refers to the possible loop-closures.
- $l(c|m)$ is the likelihood that the loop-closure can be reached. It is assumed to be inversely proportional to the length of the path of the loop-closure.

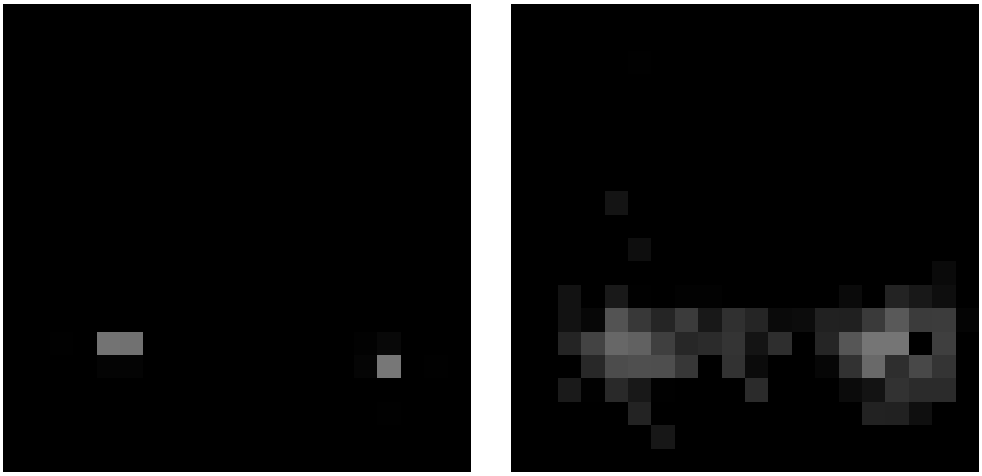
Finally, assuming that all the loop-closures trough unknown area yields the same uncertainty reduction, I_{traj} can be approximated with S_f.

Application of Deep Learning

The author of [5] proposes the use of deep learning to predict the position of the exit locations on a previously unseen building by giving the algorithm access to the building floor plan.

Problem formulation The algorithm uses a frontier-based exploration. The frontier closest to an estimated exit is selected as next target; the exploration goes on until an exit location is found. The objective is to minimize the area explored during the exploration to reach the exit.

Implementation The *Convolutional Neural Network* (*CNN*) is trained using downsampled 256×256 blueprint image as input. The output is a 20×20 image with pixel values equal to 1 for exit locations, 0 otherwise. The size 20×20 is choosen since bigger patches would increase the complexity the CNN would need to learn. The CNN is built as following:



(a) (b)

Figure 2.10: Figure 2.10a are the labeled output locations. Figure 2.10b shows the output of CNN. From [5].

1. Input layer ($1 \times 256 \times 256$, the pixels of the blueprint image)
2. Convolutional layer (16 filters, filter size 9)

3. Max pooling (size 2)
4. RELU Non-Linear Layer
5. Convolutional layer (16 filters, filter size 9)
6. Max pooling (size 2)
7. RELU Non-Linear Layer
8. Convolutional layer (16 filters, filter size 5)
9. Max pooling (size 2)
10. RELU Non-Linear Layer
11. Convolutional layer (16 filters, filter size 5)
12. RELU Non-Linear Layer
13. Fully Connected layer (550 nodes)
14. Output layer (400 nodes - 20 20)

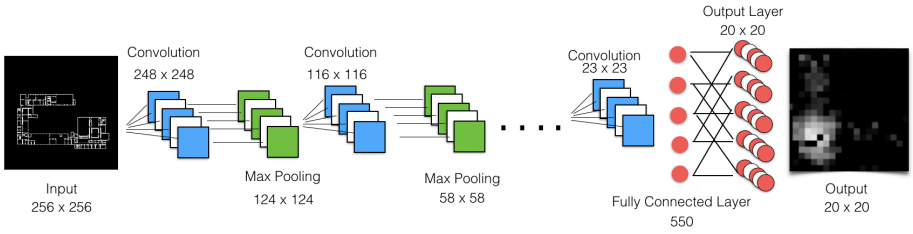


Figure 2.11: The figure shows the structure of CNN. From [5].

The error function is computed as a pixel-by-pixel comparison between the result of the CNN and the ground truth image of the true exit locations. The false negative error is multiplied by 400/3 in order to balance positive and negative exit locations. The objective of this normalization factor is to encourage the CNN to predict a 'heat map' of possible location (as shown in Figure 2.11) instead of learning to output an all black image. From the output images a clustering method is used to determine the exact location of the exit prediction. A k-mean clustering algorithm is applied.

A semantically informed exploration

The authors of [18] propose an algorithm that exploits a semantic map of the environment to enhance the exploration of the environment in a search and rescue settings. A semantic map is a map that attaches to elements of the environments a set of related concepts that give a more human-like knowledge (e.g. label a part of the map as a 'small room'). The search and rescue setting is a particular type of exploration where the objective is to find the highest number of persons in an environment in a short amount of time. The algorithm uses a semantic map in order to explore those areas that are more likely to have an higher number of persons. A priori knowledge of the labels of the semantic map that have an higher priority to be explored is given as parameter(called target label). This assumption is interesting in realistic scenario; for example, if a disaster inside an office happens during office hours, it is most likely that a large portions of humans are inside small-sized rooms.

System overview The robots use laser sensors with a coverage of 360 degrees around them. Each robot builds a two dimensional occupancy grid map of the explored environment, meanwhile a global map is maintained by a base station which is in a fixed position of the environment, and to which the robots send their maps. The authors assume that the system has a semantic map that labels each free cell of the grid map with its room type. The semantic map is assumed to be given since the authors are only interested in its use (an example of semantic mapping is described in Section 2.4.1). The semantic labels used are:

- small room
- medium room
- large room
- corridor

Exploration strategies The authors propose a multi-criteria decision making approach using several criteria to evaluate the goodness of a candidate location for all the robots used. The authors assumes that to each cell in the free space of the map the label of the corresponding room is available. The candidate locations p belongs to one of the frontiers of the environment and the criteria to evaluate them are:

- $A(p)$: expected amount of free area beyond the frontier of p computed as the length (in cell) of the frontier.
- $d(p, r)$: is the euclidean distance between the position p and the robot r.
- $b(p, r)$: is an estimation of the energy spent to reach the position p for robot r.
- $S(p)$: is the relevance of p. This value depends from the target label given as parameter. If the label of p is equal to the target label, $S(p) = 1$, 0 otherwise. If the label of p is 'corridor', then $S(p) = 0.15$ independently from the a-priori parameter.
- $ND(p)$: is the number of doors in the room where p is located. This is helpful to exploit highly-connected rooms that can ease finding relevant rooms.

All the criteria $N = \langle A, d, b, S, ND \rangle$ are combined using a multi-criteria decision making in order exploit the correlation between the various criteria (for example, the Euclidean distance and the energy spent for a position p are dependant).

Coordination method A coordination method is used to assign candidate locations to robots. The mechanism is auction-based, where the items are the candidate locations p and the robots attempt to maximize their utility function $u(p, r)$. The base station is the auctioneer. More than one robot can be assigned to the same candidate locations. For example, two robots allocated

in a candidate position of a 'large room' can speed up the exploration, overcoming the potential negative effect due to the initially overlapping views. A fuzzy-based function $i(p)$ is used to compute the ideal number of robots that should be assigned to a room. The algorithm combines the different criteria listed before to calculate the ideal number of robots. An example of how the fuzzy based algorithm works is shown in Figure 2.12.

1 **if** *RoomSize* is *SMALL* **and** *#Doors* is *HIGH* **and** *Visibility* is *LOW* **and** *AlreadyPercArea* is *MEDIUM* **then** *#Robots* is *MEDIUM*;
2 **if** *RoomSize* is *BIG* **and** *#Doors* is *LOW* **and** *Visibility* is *LOW* **and** *AlreadyPercArea* is *HIGH* **then** *#Robots* is *LOW*;
3 **if** *RoomSize* is *BIG* **and** *#Doors* is *MEDIUM* **and** *Visibility* is *MEDIUM* **and** *AlreadyPercArea* is *LOW* **then** *#Robots* is *HIGH*;

(a)

1 **if** *CorridorSize* is *SMALL* **and** *#Doors* is *HIGH* **and** *#IntersectingCorridors* is *MEDIUM* **and** *AlreadyPercArea* is *MEDIUM* **then** *#Robots* is *LOW*;
2 **if** *CorridorSize* is *SMALL* **and** *#Doors* is *MEDIUM* **and** *#IntersectingCorridors* is *LOW* **and** *AlreadyPercArea* is *LOW* **then** *#Robots* is *MEDIUM*;
3 **if** *CorridorSize* is *MEDIUM* **and** *#Doors* is *MEDIUM* **and** *#IntersectingCorridors* is *MEDIUM* **and** *AlreadyPercArea* is *LOW* **then** *#Robots* is *HIGH*;
4 **if** *CorridorSize* is *BIG* **and** *#Doors* is *HIGH* **and** *#IntersectingCorridors* is *MEDIUM* **and** *AlreadyPercArea* is *LOW* **then** *#Robots* is *VERY_HIGH*;

(b)

Figure 2.12: Figure 2.12a and Figure 2.12b show some example of how the fuzzy function works. From [18].

Each robot evaluates the candidate positions as soon as the base station makes an auction or when requested by a robot that has reached its assigned location, and submits a bid $u(p, r)$ with $u(p, r)$ being the utility function of position p for robot r. Two coordinates method are proposed:

- MRv1: it greedily allocates the best pair (p', r') avoiding to allocate to p' more than $i(p')$ robots($i(p)$ ideal number of robots for position p).
- MRv2: similar to MRv1 but after each allocation of a robot to p' it discount the utility of p' for the other robots according to the number of robots already assigned to p'. The discount is linear until the number of robot assigned to p' is less or equal than $i(p')$, and the decays exponentially. An example is shown in Figure 2.13.

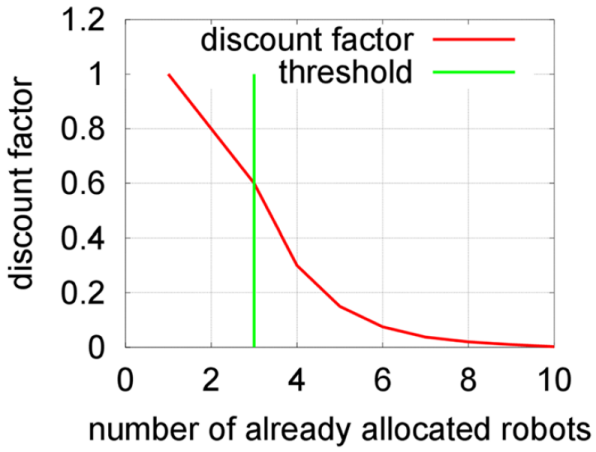


Figure 2.13: The figure shows an example about the function of the discount factor. From [18].

2.4 Off-line map inference

In this section we introduce some studies that focus on the exploitation of general knowledge from the map. The objective is to obtain more human-level concepts from the environment, like rooms. The authors of [21] propose an algorithm that, knowing the building topology (e.g. school, office) performs a semantic mapping on the environment. Similarly, [26] proposes to use multi-modal sensory information including information gathered from humans to make inference about unexplored concepts (e.g. objects, rooms) and allows for goal oriented exploration making prediction on the possible extension of the known world.

2.4.1 Semantic mapping

In order to implement human-like behaviour with robots it becomes very important to provide them the fundamental capability of understanding complex and structured environments [25]. *Semantic mapping* has the objective of classify different parts of the environment based on their general appearance and geometry. We illustrate some algorithms that propose different approaches to perform semantic mapping.

Semantic mapping based on building topology

The authors of [21] propose a classification algorithm that is based on the concept of building topology. The *building topology* denotes a set of buildings that have the same function and that share common structural features. The classifiers give to each room a label that can be:

- small room
- medium room
- large room
- corridor
- hall

During the classification of an unknown environment, the only a priori knowledge is the building topology in order to perform an "informed" semantic mapping.

Building topology Every building is created for a specific function that imposes its structure, floor plan and the structure of the rooms. Many algorithms that apply semantic mapping ignore this important feature, considering all the environments as natural, fixed and immutable entities. The authors of [21] consider three different building topologies:

- House
- Office
- School

Costruction of the classifiers For each different building topology a specific classifier is trained with data regarding that specific building topology. The indoor environments are represented by line segment-based metric maps using data produced from laser range scanners. The system first derives the topological map, represented as a graph where each node is a room and the

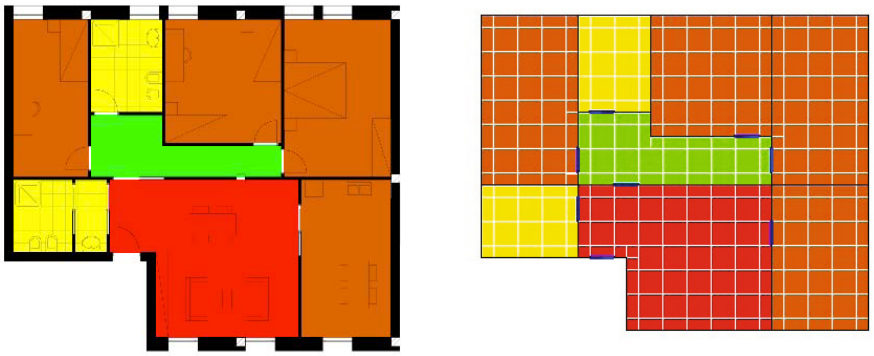
doorways are the edges that connect two different nodes (rooms). Each node is associated with a label ('small room', 'medium room', 'large room', 'corridor' and 'hall'). Each room is described with a vector of features chosen to capture some of the characteristics of the room. The features are divided into three groups:

- Area a of the room and the axes ratio $rt = M/m$ of the major axis M and minor axis m.
- Number d of doors for room r.
- Labels of room directly connected to r.

For each room we count for each label s (that could be S, M, L, C, H for 'small room', 'medium room', 'large room', 'corridor' and 'hall') the number of door l_s that connect room r to a room of label s. r can be described as $F_r = \langle a, rt, d, l_S, l_M, l_L, l_C, l_H \rangle$. If a room is connected to an unclassified room, a temporary label is assigned using a simple classifier that uses only some features. All the classifiers are trained using a supervised learning approach using data sets. For each entry, a vector of features F_r is given together with the corresponding label. The floor plans used were selected from books for the design and analysis of buildings in architecture, where each room has a label assigned based on its purpose. Then the labelled floor plans have been digitally represented using a custom CAD-like software.

For each typology-specific classifier data sets are composed of approximately 2000 rooms each.

Use of the classifiers Data are assumed to be collected from a mobile robot with a view of 360 degrees around it. After the metric map is built, the free space already seen is represented by a polygon (even with holes) where the line segments are either lines representing obstacle or frontiers. At the beginning, short line segments are removed to filter parts of furniture and small obstacles. An example is shown in Figure 2.15. From the filtered map the features are collected in the following way:



(a) (b)

Figure 2.14: Figure 2.14a shows floor plan with AutoCAD(left), Figure 2.14b shows floor plan drawn with CAD-like software. Colours indicate semantic labels. From [21].

- Walls are identified by finding out collinear line segments (line segments are considered collinear if their angular coefficient is equal or similar up to a certain threshold).

- Doors are identified as gaps between collinear line segments and have a certain size (higher than 30cm and lower than 150cm).

Portions of area in free space are identified with a Monte Carlo method. A random number of points are thrown within the space, then for each point p a set of line segments L_p visible from p is calculated. The points are checked for straight visibility between them. If point p and point p' are mutually visible, they are considered to belong to the same room. A graph representing the topological map is created considering the rooms as nodes and the doorways as edges. Finally the set of features F_r for each room r is calculated:

- a is calculated as the average area of the inner and outer boxes that best approximate the room.

- M is selected as the longest wall of the room.

- m is selected as the longest wall of the room perpendicular to M.

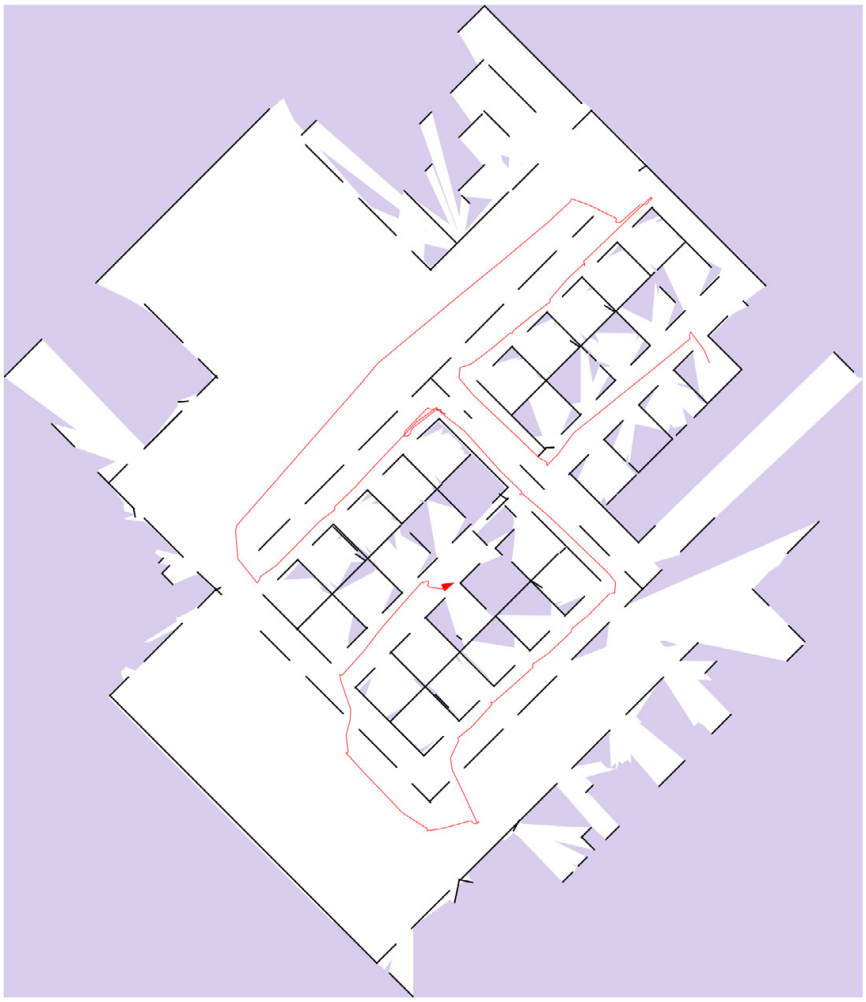
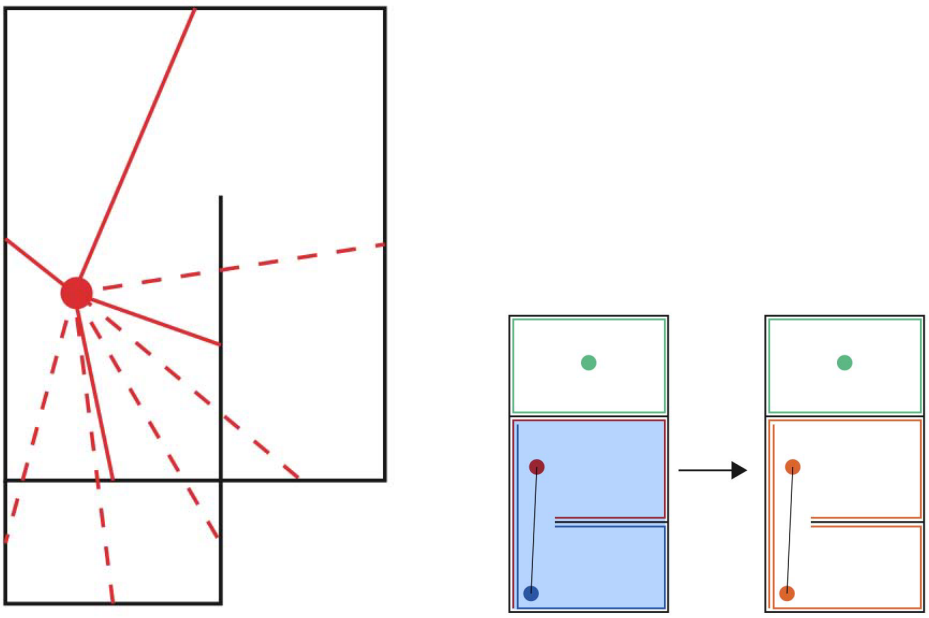


Figure 2.15: The figure shows the filtered map of an environment. From [21].

The rest of the features representing the connection graph are directly extrapolated from the graph.

Pronobis and Jensfelt method

The authors of [26] propose the use of multi-modal sensors integrated with conceptual common-sense knowledge in a fully probabilistic framework. It relies on the concept of spatial properties. A probabilistic graphical model is used to represent the conceptual information and to perform spatial rea-



(a) (b)

Figure 2.16: Figure 2.16a shows the line segments directly visible from a point (red line) and not visible (dotted line). Figure 2.16b shows mutually visible points in the same room. From [21].

soning. The system uses the information about the existence of objects, landmarks, geometry and topology of space. Spaces are distinguished between places and placeholders; the difference is that the second ones are locations not fully explored that can become possible targets of the exploration. A conceptual map of the relations is built (an example is shown in Figure 2.17), that may be:

- concepts relations (kitchen has cornflakes)
- instance and concept relations (object1 is-a cornflakes)
- instance relations (object1 is-in place1)

The paradigm used for semantic mapping is a property-based approach. To each space different properties are assigned that may be like the number of specific objects found in the space, the shape and size of the room. This allows a fine-grained and a more descriptive representation of the space.

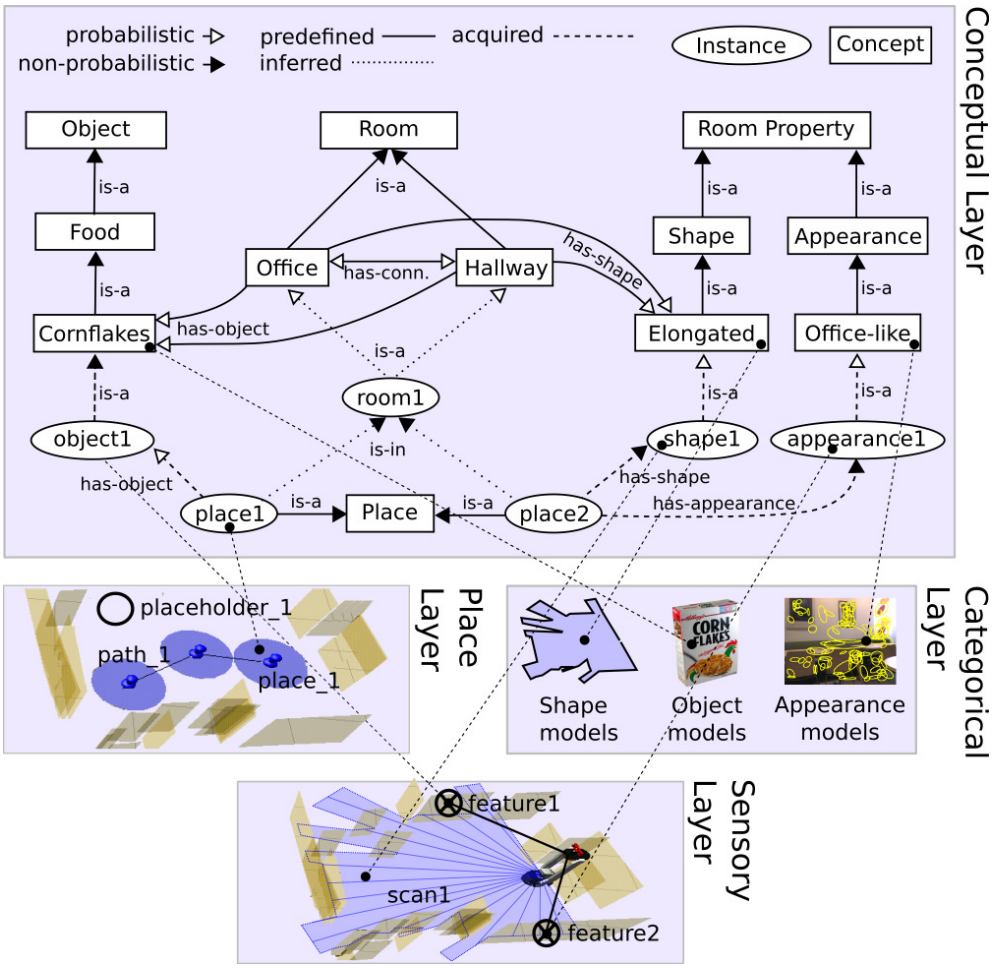


Figure 2.17: The figure shows the spatial representation and a visualization of an excerpt of the ontology of the conceptual level. From [26] .

Sensory model of properties In order to capture the semantic properties of spatial objects, different models of sensory information are trained to specifically capture different features of the environment. The models trained are:

- Geometrical property models to capture shape and size of places.
- Appearance property models to capture the visual appearance of places.
- Object models to capture specific models for objects belonging to a typical office environment.

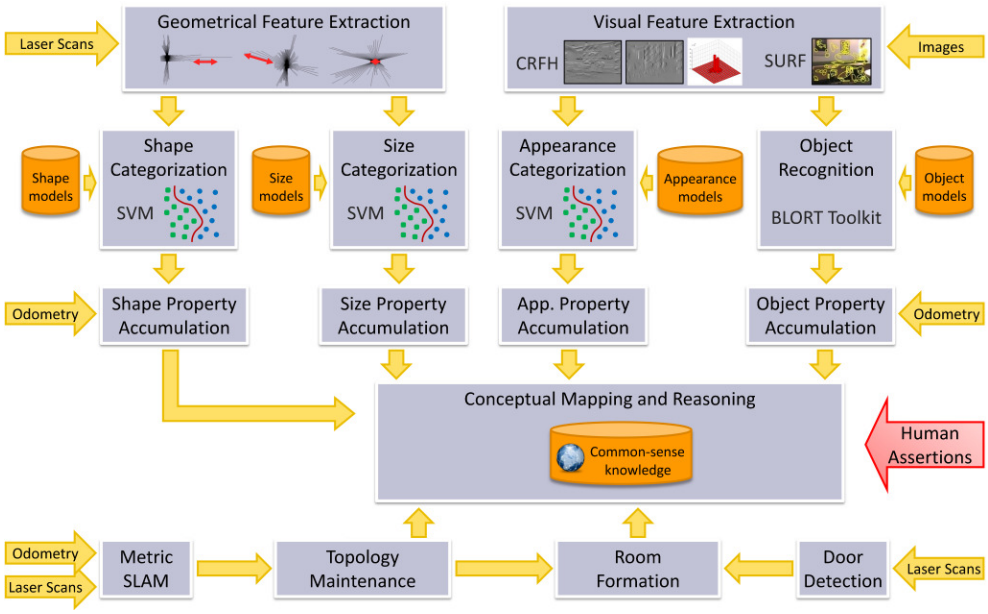


Figure 2.18: The figure shows the structure of the system and the data flows. From [26] .

Finally, a *chain graph model*is used to create the conceptual map. This allows a probabilistic relational conceptual representation, allowing to make inference about some concepts based solely on their relations to other concepts. For example, it is possible to predict what kind of label a placeholder (unexplored area) could have, or what kind of objects can be found inside. Another possibility is to predict whether:

- Placeholders do not lead to a new room.
- Placeholders lead to a new room.
- Placeholders lead to a new room connected to a new room.

2.4.2 Graph-based prediction

In this section we present an algorithm that uses a dataset of different maps to make inference on the graph representation of the examined environment and predict the presence of possible missing rooms (node of the graph). The

authors of [3] propose a data-driven graph-based approach for building models of indoor environments, assigning labels to rooms of the environment and predicting what lies ahead in the topology of the environment. The algorithm uses of two big datasets of floor plans, each one represented as a graph; the datasets consist of 940 floors and over 38,000 rooms. An important property is that in floor plans graphs the local complexity remains nearly constant for an increasing global complexity.

Problem formulation The graph used is defined as a three-tuple $G = (V, E, \alpha)$, where V is a finite set of nodes, $E \subseteq V \times V$ is a finite edge set and $\alpha : V \to \mathcal{L}$ is a node label mapping. Given a *graph database* $D = \{G_1, ..., G_n\}$, and a graph G, D_G is defined as the subset of D whose elements are the graph $G_i \in D$ such that G is a sub-graph of G_i.$freq(G)$ is defined as $freq(G) = |D_G|$ and the support of graph G as:

$$supp(G) = \frac{freq(G)}{|D|} \tag{2.21}$$

A graph is called *a frequent subgraph* of D if $supp(G) \geq \pi$ with π as minimum support threshold. Finally, two different solutions to extend the graph are proposed.

Count Based Prediction Given a graph G_p the steps are:

- Compute the projected dataset D_{G_p}.
- For each possible edit operation a with resulting G'_p, calculate $supp(G'_p)$.
- The edit operation whose resulting graph has the highest support is chosen.

From the data collected, this algorithm is able to perform well with small graph sizes.

Exploiting subgraph Given the data set D, the most frequent sub-graphs are extracted using the gSpan algorithm and used to create a new data set S composed of sub-graphs. The following steps are:

- 1: Split the input graph G_p into smaller, overlapping subgraphs, forming a new set C.
- 2: For each element in C, determine the probability of every possible edit operation.
- 3: For each subgraph $c \in C$, we define the corresponding subgraph $c' \in C'$, with c' derived from c by applying the most likely one edit operation. The edit operation selected is the one that has the highest support from the graph database.

Chapter 3

Problem definition

In this chapter we formally describe the problem we addressed during our work. In Section 3.1 we discuss the exploration problem and in Section 3.2 we discuss about the purpose of this thesis.

3.1 Exploration problem

The *exploration* is one of many important tasks that an autonomous robot can perform: the robot placed in initially unknown environment needs to move around the environment and build a map that represents the free areas and the obstacles of the environment. An important feature the robot requires is the capability to chose how to move in the partially-known environment. To achieve this task many exploration strategies have been developed.

Several algorithms are based on the concept that no knowledge of the environment is given, then the algorithms generate a set of next candidate locations and evaluate them using only the map of the environment built so far. Some of this algorithms are described in Section 2.3; for example [34] described in Section 2.3.1 proposes to move the robot to the closest frontier. Another example is the one of [13], and described in Section 2.3.1, that evaluates the candidate positions based on the estimation of potential visible area and the distance of the candidate position from the robot.

The exploration process can be described by the following operations:

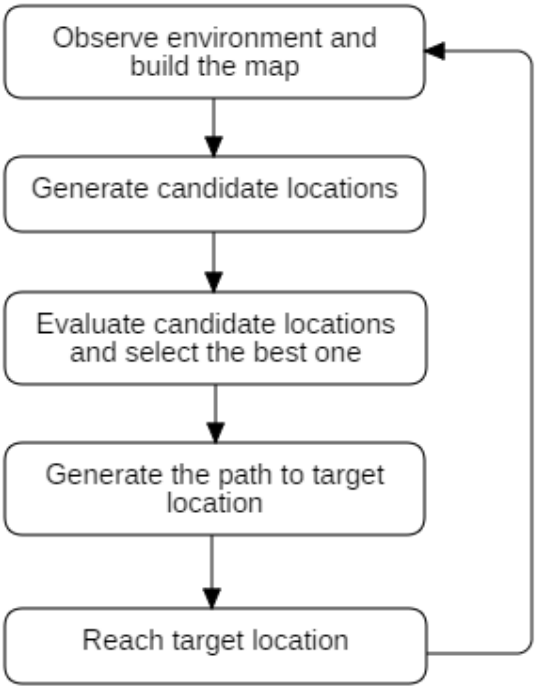


Figure 3.1: The figure shows the steps of the exploration process.

1. Observe the environment and integrate the data collected into the global map.
2. Generate candidate locations.
3. Evaluates candidate locations and select the best one.
4. Generate the path that goes from the actual location to the selected target location.
5. Reach the target location.

A diagram of the steps described can be found in Figure 3.1.

The main limit of this kind of approaches is that the evaluation of the candidate positions is performed using only the point of view of the robot: the distance of the candidate location from the robot, how much area the sensors of the robot can potentially perceive. The structure of the environment

known so far is not used. Another important limit is that the robot needs to visit all the environment in order to correctly build a complete map. Typically, the robot needs to visit all the remaining frontiers until no one is left in order to end the exploration.

A potential solution to increase the performance of the robot during the exploration is to extract some knowledge from the environment explored so far, and use this knowledge to make some prediction on what may be found in the unexplored area.

Although some algorithms that make predictions on the map of the environment exist, most of them are used for very specific uses [5, 30] or need some a priori knowledge to work [18]. Another issue is that no information about the possible geometrical structure, or layout, of the unexplored part of the environment is given except for [31], despite it is used only to find possible loop closures, without providing any information about the possible areas that can be seen.

3.2 Purpose of this thesis

The purpose of this thesis is to study and implement an algorithm able to reconstruct the layout of the environment seen during the exploration and to predict the layout of the unknown part of the environment. The next step is to implement an exploration strategy that uses the predicted information to perform exploration, and check whether it produces an enhancement on the performance of the exploration. Finally, we focus on the possibility to use the predicted informations about the unexplored part of the environment to complete the map without having to explore the environment completely and thus saving exploration time. The steps for the exploration would become:

1. Observe the environment and integrate the data collected into the global map.

2. Generate candidate locations.

3. Predict the layout of the environment.

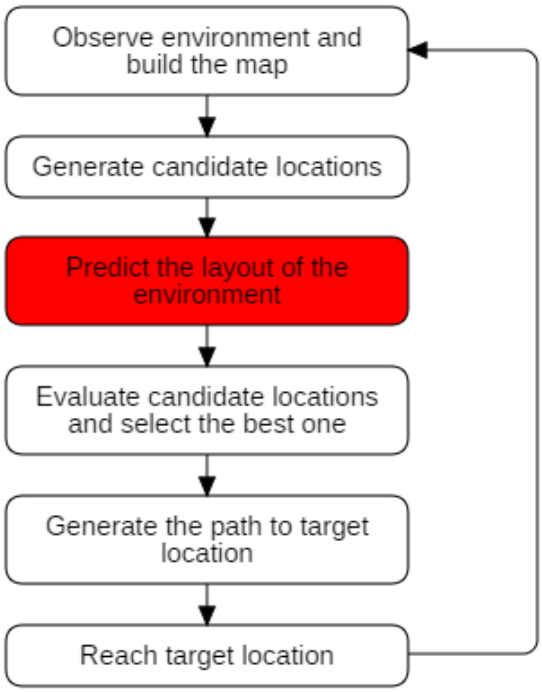


Figure 3.2: The figure shows the steps of the new exploration process. The new step in enlighten in red.

4. Evaluates candidate locations using the predicted layout and select the best one.

5. Generate the path that goes from the actual location to the selected target location.

6. Reach the target locations.

A graphical representation of how the new exploration process is organized is shown in Figure 3.2.

Chapter 4

Problem solution

In this chapter a proposed solution to the problem described in Chapter 3 is presented. The solution can be divided into two modules that solve specific parts of the problem:

- Exploration module: This module builds the map of the environment, localizes the robot, generates the candidate locations and moves the robot.

- Layout prediction module: This module uses the map of the environment and the frontiers sent from the exploration module to reconstruct the layout of the environment, predicts the layout of the unexplored part and finally returns the potential area the robot can perceive for each frontier.

In Figure 4.1 the interaction between the two modules is shown. In Section 4.3 we describe how the layout prediction works and in Section 4.2 how the exploration module works to solve the problem described in Chapter 3. In Section 4.4 we discuss about an important feature of our works: the capability to understand when the exploration can be stopped since the map has been fully reconstructed and predicted by the layout prediction module.

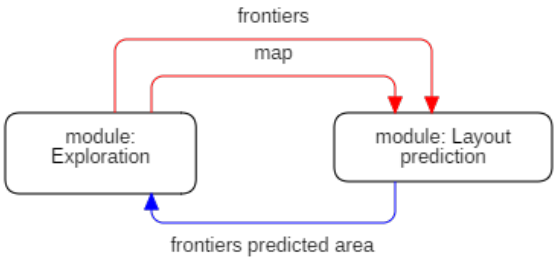


Figure 4.1: The figure shows the interaction between the exploration module and the layout prediction module.

4.1 Assumptions

Environments In our work the environments are assumed static, it means the environments do not change during the exploration. Only indoor environments are considered in our work formed of a single floor. All the free areas of the environments can be visited by the robot. The environments used are uncluttered, namely no clutter like pieces of furniture can be found.

Map The map of each environment is incrementally build and is represented as an occupancy grid map [33] indicated as M, with a fixed resolution. Each cell $c \in M$ is represented by a tuple $\langle x_c, y_c, state_c \rangle$ where (x_c, y_c) are the coordinates of the centre of cell c in some coordinate system and $state_c$ indicates whether the cell is:

- free
- occupied
- unknown

The map is assumed to be oriented such that walls correspond mostly to horizontal and vertical lines. It is assumed that no oblique walls are present.

Robot The robot is assumed to be a wheeled autonomous robot that can perceive the environment trough a laser range scanner, whose range is r.

The robot during the exploration is free to move in all the free spaces of the environment and it is able to localize itself in the map.

4.2 Exploration

In this section we describe how the *exploration module* solve the *exploration problem*. Our solution is a modification of the algorithm proposed by [1]. The scope of [1] was to solve the exploration problem using a multi-criteria exploration strategy that extends the frontier-based approach of [13]. The algorithm uses an hand-drawn map or a floor-plan of the environment given before the exploration in order to improve the performance. The main steps of the our algorithm are shown in Algorithm 1, where:

Algorithm 1 SELECT NEXT TARGET LOCATION

1: **function** (SELECTNEXTTARGETLOCATION(M, st, α))
2: $F \leftarrow$ frontiersDetection(M, st)
3: $C \leftarrow$ generateCandidateLocations(M, F)
4: $I, F', C' \leftarrow$ predictLayoutOfTheEnvironment(M, F)
5: $U \leftarrow$ evaluateFrontiers(C', st, M, F', I, α)
6: $p' \leftarrow$ selectBestLocation(U)
7: **return** p'

- M is the partial map built during the exploration represented as an occupancy grid map.
- st is the cell of the map M corresponding to the actual location of the robot.
- α is a parameter used to weight the different criteria used during the evaluation of each frontier.
- p' is the next target location for the exploration.

The common steps between our solution and [1] that are described in Algorithm 1 are:

- FRONTIERSDETECTION: Given the map M of the environment represented as an occupancy grid, this step produces the frontiers of the environment.

- GENERATECANDIDATELOCATIONS: given the map M and the set of frontiers F, for each frontier $f \in F$ it generates a candidate positions $c \in C$.

- SELECTBESTLOCATION: given the map and the evaluation of each frontiers, it selects the best locations and move the robot to it.

The step that has been modified is:

- EVALUATEFRONTIERS: the candidate locations are evaluated with a multi-criteria exploration strategy, using as criteria both the distance from the robot and the potential information the frontier can provide, but how this criteria is calculated between our method and the one of [1]. Their system uses the map given a priori to know how much area the robot is going to perceive with sensors, meanwhile our algorithm uses the predicted area produced with PREDICTLAYOUTOFTHEENVIRONMENT. The result is the set U that contains the evaluation of all the frontiers.

The step that is completely new is:

- PREDICTLAYOUTOFTHEENVIRONMENT: Performed by the layout prediction module described in Section 4.3, it reconstructs the layout of the environment and returns the areas of the partially-observed rooms I together with the associated frontiers F'. C' is the set of candidate locations c' associated to frontier $f' \in F'$.

In the following sections the steps previously listed will be discussed in depth.

4.2.1 Frontiers detection

The first step to find all the possible frontiers is to detect all of the edge cells E: those are 'free' cells adjacent to at least one 'unknown' cell. In order to be part of a frontier a constrain must be satisfied: the 'free' cell p must be

reachable from the actual position of the robot st. In order to be reachable a sequence of 4-connected 'free' cells from st to p must exist. An example of a possible 4-connected path of 'free' cells from cell st to cell p can be seen in Figure 4.2b highlighted in yellow. Two 'free' cells are considered connected they have one edge in common. Once the that the edge cells has been found and the constrain of the 4-connected path for each of them has been satisfied, the last step is to merge the edge cells in frontiers. An example of edge cells that are reachable from st is shown in Figure 4.2c; the cells are highlighted in orange. Two edge cells belongs to the same frontier $f \in F$ if the two cells are adjacent or exist a 4-connected path composed of the points of the frontier F' between the two cells. In Figure 4.2d three different frontiers can be seen. The Algorithm 2 shows the various step to individuate the edge

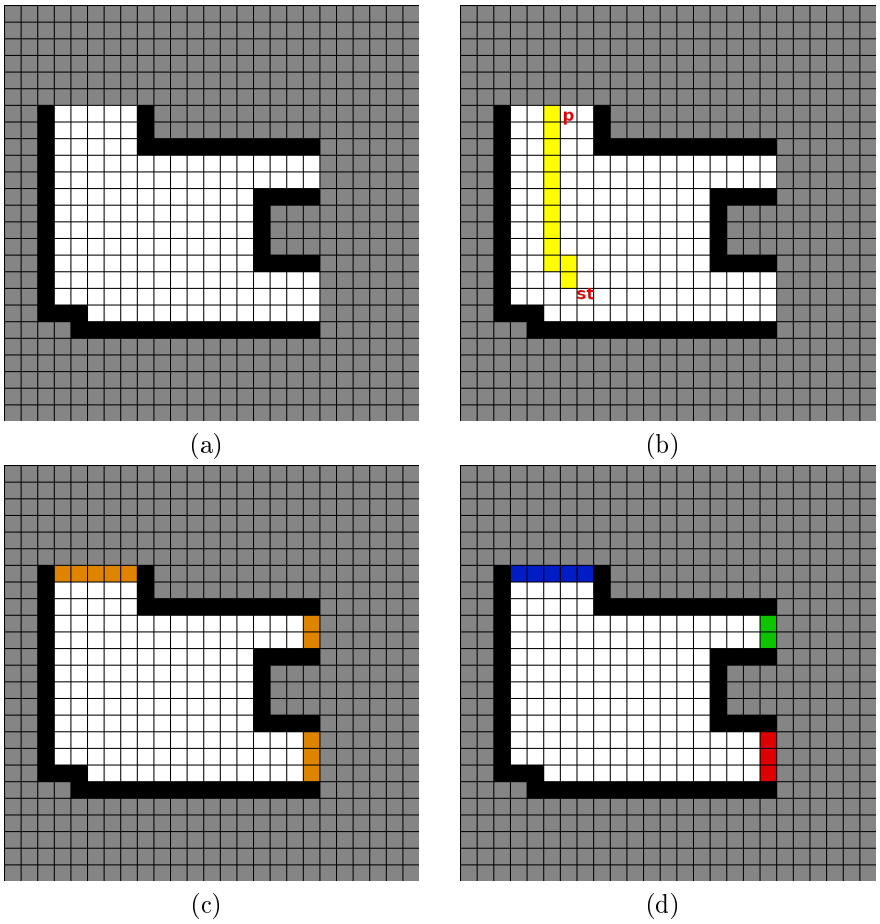
Algorithm 2 FRONTIERS DETECTION

```
1: function (frontiersDetection(M, st))
2:     E ← findEdgeCells(M)
3:     E' ← {}
4:     F ← {}
5:     for all e ∈ E do
6:         if isReachable(e, M, st) then
7:             E' ← E' ∪ e
8:     F ← createFrontiers(M, E')
9:     return F
```

cells E, remove the unreachable ones and finally create the frontiers F using the filtered edge cells E'.

4.2.2 Generate candidate locations

The next step is the generation of the target locations C. A target location $c \in f$ is chosen as the point that divides a frontier $f \in F$ into two equal segments. In Figure 4.3 an example of frontiers and relative candidate locations is shown. The Algorithm 3 shows the steps to create a candidate location $c \in f$ for each of the frontier $f \in F$. C is the set of all the possible candidate locations c.



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

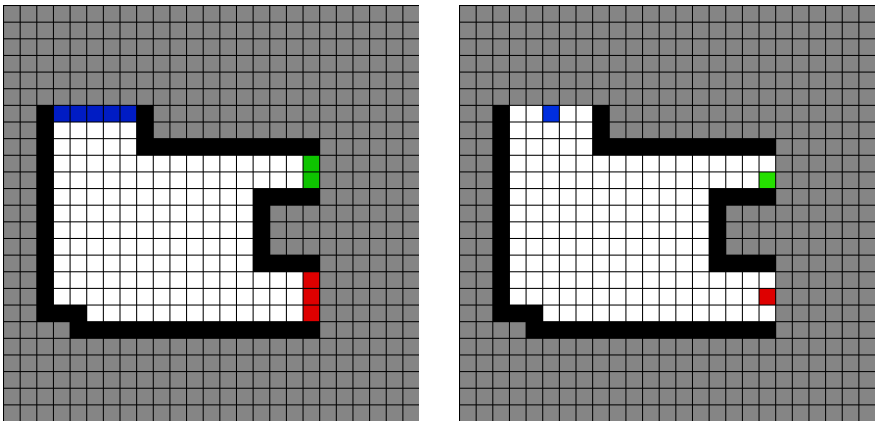
Figure 4.2: Figure 4.2a shows an example of occupancy grid map of an environment. Figure 4.2b shows an example of 4-connected path of 'free' cells from cell *st* to cell *p* highlighted in yellow. In Figure 4.2c an example of edges cells that are reachable from *st* in orange is shown. Figure 4.2d shows three different frontiers with 3 different colours.

Algorithm 3 GENERATE CANDIDATE LOCATIONS

1: **function** (GENERATECANDIDATELOCATIONS(M, F))
2: $C \leftarrow \{\}$
3: **for all** $f \in F$ **do**
4: $C \leftarrow C \cup$ getMiddlePoint(f, M)
5: **return** C

4.2.3 Predict the layout of the environment

This step is described in details in Section 4.3. From the point of view of the exploration module, the module send an occupancy grid map representing



(a) (b)

Figure 4.3: Figure 4.3a shows 3 different frontiers. Figure 4.3b shows the candidate location for each frontier.

the environment seen so far together with the set of frontiers F calculated in Section 4.2.1 to the *layout prediction module*. Then the layout prediction module return the expected area of rooms r_i that have not being fully explored during the exploration yet, and the associated frontiers f'. For each partially-observed room r_i, typically more than one frontier can be found inside the room. The frontier f' that has the highest number of points inside the room r_i is selected. This lead to a reduction of the number of frontiers $F' \subseteq F$ returned from the exploration module. Then the information gain $i(c')$ for the a candidate location $c' \in C'$ is defined as the expected area of the room associated to frontier $f' \in F'$ with candidate location c' that can be explored by the robot. The set I is the collection of all the information gain $i(c')$.

4.2.4 Frontiers evaluation

The next step is the evaluation of all the possible candidate locations $c' \in C'$. The set $C' \subseteq C$ corresponds to the set of candidate locations on the frontiers F' that have been returned by the layout prediction module. The evaluation is performed using a *multi-criteria decision making* that uses as criteria:

- $d_e(c', st)$: the *Euclidean distance* of the candidate location c' from the

robot's position st. The length of the path from c' to st is not used since is much slower to compute than the Euclidean distance that is however a good approximation.

- $i(c')$: the *information gain* that measures the visible unexplored area of the room r_i associated to the frontier $f'_i \in F'$.

The utility function $u(c')$ is calculated as following:

$$u(c') = \alpha \cdot \tilde{d}_e(c') \cdot (1 - \alpha) \cdot \tilde{i}(c') \tag{4.1}$$

Where:

- $\tilde{d}_e(c')$ is the normalized Euclidean distance between the candidate position c' and the robot position st calculated as:

$$\tilde{d}_(c') = \frac{d_{max} - d_e(c', st)}{d_{max}} \tag{4.2}$$

  With $d_{max} = \text{argmax}_{c' \in C'} d_e(c', st)$.

- $\tilde{i}(c')$ is the normalized measure of the information gain calculated as:

$$\tilde{i}(c') = \frac{i(c')}{i_{max}} \tag{4.3}$$

  With $i_{max} = \text{argmax}_{c' \in C'} i(c')$.

The result is the set U which is described by the tuple $\langle c', u(c') \rangle$ with c' being the candidate location and $u(c')$ its utility value.

4.2.5 Select best location

The last step is to select the best candidate location to explore. Given the set U which elements are described by a tuple $\langle c', u(c') \rangle$ for each candidate position $c' \in C'$, the best candidate location is selected as:

$$p' = \underset{c' \in C'}{\text{argmax}}\, u(c') \tag{4.4}$$

The selection is performed by selecting the candidate position c' that has the maximum utility value. The Algorithm 4 shows the various step performed to select the best location.

Algorithm 4 GENERATE CANDIDATE LOCATIONS

```
1: function (selectBestLocation(U))
2:     p' ← None
3:     u(p') gets − 1
4:     for all ⟨c', u(c')⟩ ∈ U do
5:         if u(c') > u(p') then
6:             p' ← c'
7:             u(p') ← u(c')
8:     return p'
```

4.3 Layout prediction

In this section we describe how the *layout prediction module* reconstructs the layout of an indoor environment starting from a metric map. The algorithm is an extension of the one introduced by [19, 20]. The algorithm can be summarized in four main steps, which are:

1. Layout reconstruction: given a metric map, the algorithm find walls, create faces that represent the various portions of the environment, and finally merges together faces into different clusters that represent the rooms of the environment.

2. Identify partially-observed rooms: given the layout of the room $\mathcal{L}$ and the frontiers the algorithm calculates whether a room has been fully-observed or not and labels them as *partially-observed* or *fully-observed*. For each partially-observed room assigns the corresponding frontier.

3. Extract partially-observed rooms from frontiers: given the layout of the rooms $\mathcal{L}$, look for external faces (not belonging to any rooms) adjacent to the already built rooms, and use the frontiers to check whether they

can be added to the previously built rooms, or used to add them to the layout $\mathcal{L}$ as new rooms.

4. Geometric prediction of the entire layout of partially explored rooms: the algorithm for each partial room extract the external faces close to the room, create all the possible combinations, and evaluate whether one of these combination can be added to extend the partial room.

In this section the four steps previously mentioned will be presented in details.

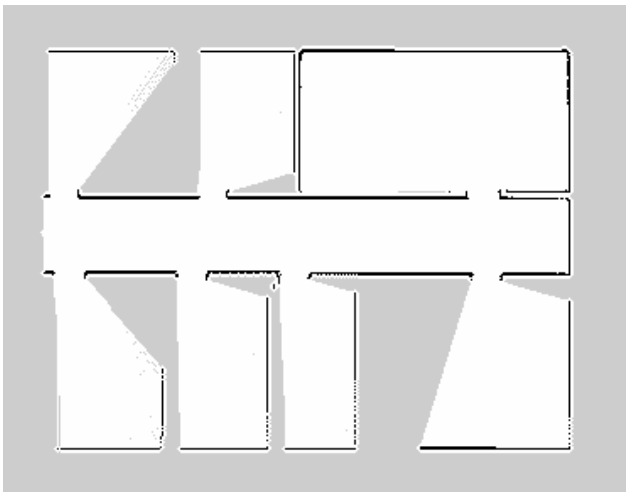
4.3.1 Layout reconstruction

The algorithm that reconstructs the layout of the environment is based on the method introduced by [19] and extended in [20]. In this section a description of how the algorithm works is provided. The algorithm originally used as input an image m of the environment to perform the various steps. We modified the algorithm to take as input the occupancy grid map M received by the *exploration module* and converting it to an image in grey scale m where the colours represent:

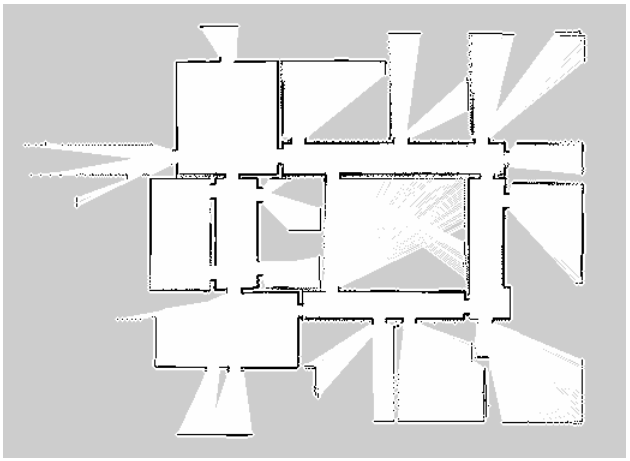
- White: the free cells of the occupancy grid.
- Grey: the cells not yet seen of the occupancy grid.
- Black: the occupied cells of the occupancy grid.

Two example of the images m of the maps are shown in Figure 4.4.

Line segments and contour detection The first step of the algorithm is the detection of the line segments S from the map m. The line segments S are important for the layout reconstruction since they represent the various walls of the map. In order to extract the line segments S the Canny edge detection [6] algorithm and Hough line transform [16] algorithm are used in sequence. The algorithms of Canny edge detection is used to extract the borders of the map m to simplify the analysis of the image by reducing the data to process, and at the same time without losing the information about



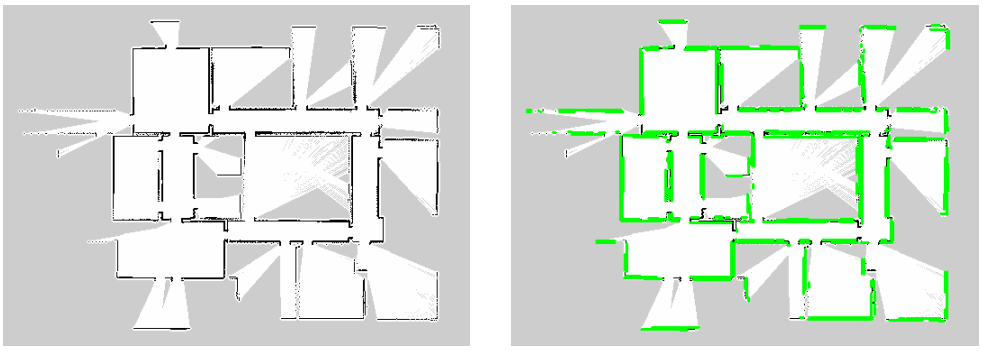
(a)



(b)

Figure 4.4: An example of two different images of two different maps.

the structural borders. The result of the Canny edge detection is passed to the Hough line transform algorithm that uses the borders to create the line segments S that correspond to the walls of the environment. In Figure 4.5 an example is shown: Figure 4.5a shows the original image of the environment, meanwhile the Figure 4.5b shows the result the Canny and Hough algorithm with the line segments S extracted highlighted in green. The next step is the detection of the contour represented as polygon that cover all the observed parts of the map. The detection of the contour is very important since it allows to identify the area of the map that corresponds to the internal part



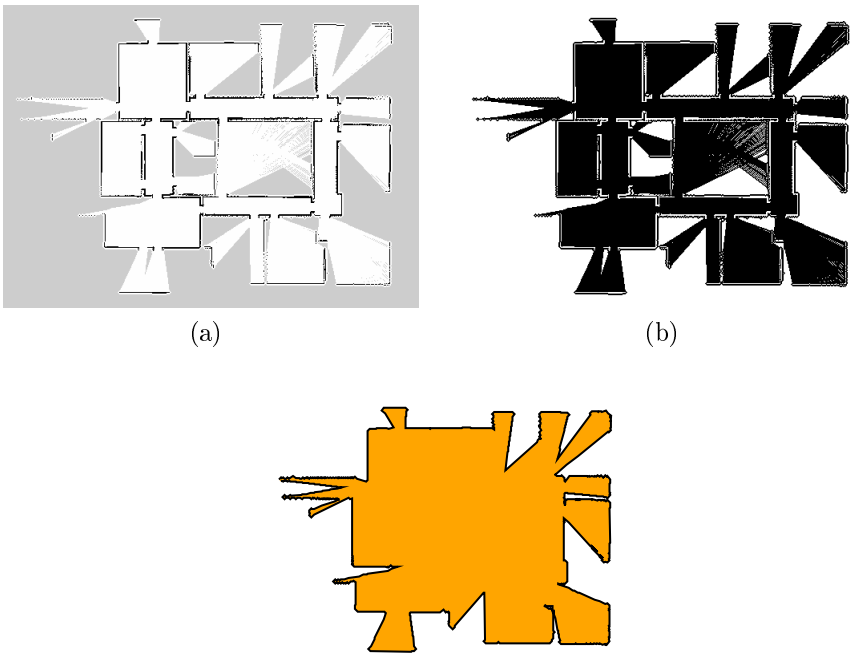
(a) (b)

Figure 4.5: Figure 4.5a shows the original image. Figure 4.5b shows the previous image with the line segments extracted with Canny and Hough algorithms highlighted in green.

of the environment. The image is modified by changing the colour of the unknown part of the map into white, and the colour of the known part into black. An example is shown in Figure 4.6b. This mechanism allow to create a clear distinction between the explored and the unexplored part of the environment. Then the algorithm [24] extracts the polygon representing the contour of the map. In Figure 4.6 an example of extracted contour is shown.

Angular clustering The objective of angular clustering and the spatial clustering (next step) is to merge line segments to create a representation of the walls of the map. As first step the detected line segments S are clustered according to the angular coefficients of their supporting lines using the *mean shift* algorithm [9]. This allows to divide line segments that surely do not belong to the same wall according to their angular coefficient. The resulting set of cluster $C = \{C_1, C_2, ...\}$ is such that $C_i \subseteq S$ and $C_1 \cup C_2 \cup ... = S$. C_i are set of line segments with similar angular coefficient α_i. In Figure 4.7 an example of angular clustering is shown where line segments belonging to the same angular cluster are highlighted with the same colour.

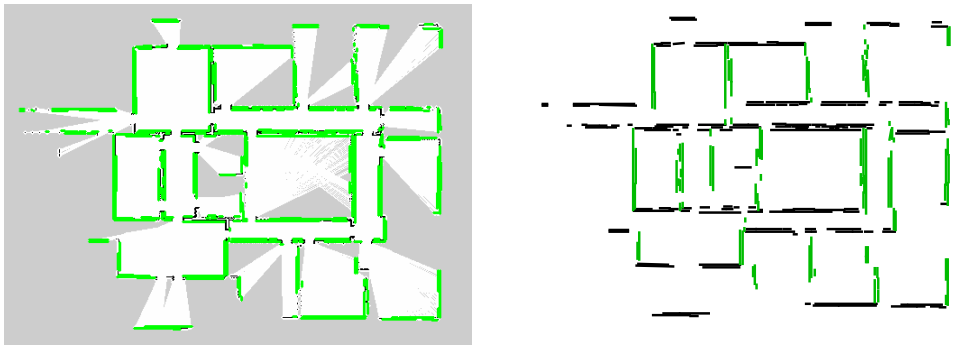
Spatial clustering The spatial clustering has the objective to merge line segments belonging to the same angular cluster in order to identify the walls



(a) (b)

(c)

Figure 4.6: Figure 4.6a shows the original image. Figure 4.6b shows the modified image to extract the contour. Figure 4.6c shows the highlighted contour.

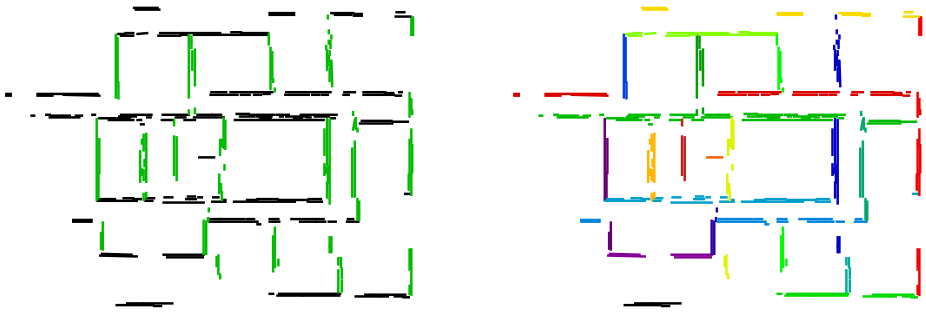


(a) (b)

Figure 4.7: Figure 4.7a shows the image with the line segments in green. Figure 4.7b shows the angular clusters highlighted with different colours.

of the map. The line segments in the same cluster C_i are clustered according to their spatial separation. Considering segments $s, s' \in C_i$ and l, l' the

lines passing trough the middle points of s and s' respectively with angular coefficient α_i, if the distance between the two parallel lines l, l' is lower than a certain threshold (set to 90 cm, approximately a width of a doorway) then the segments s, s' are put in the same spatial cluster $C_{i,k}$. An example of walls detected is shown in Figure 4.8.

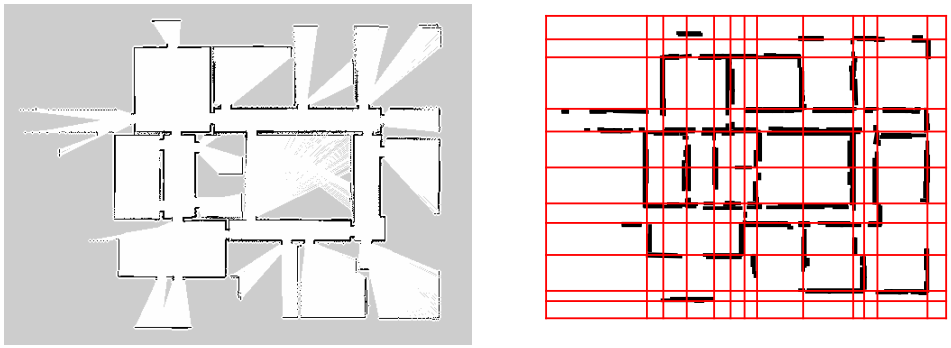


(a) (b)

Figure 4.8: Figure 4.8a shows the angular cluster. Figure 4.8b shows the walls detected.

Representative lines The representative lines are created in order to be used in the next step for the creation of faces. For each cluster $C_{i,k}$ a representative line $l_{i,k}$ that represents the cluster is determined. It is computed as the line that passes trough the median of the set of middle points of the line segments in $C_{i,k}$ and with angular coefficient α_i. The line $l_{i,k}$ indicates the the direction of a wall of the buildings. An example is shown in Figure 4.10a with the representative lines drawn in red.

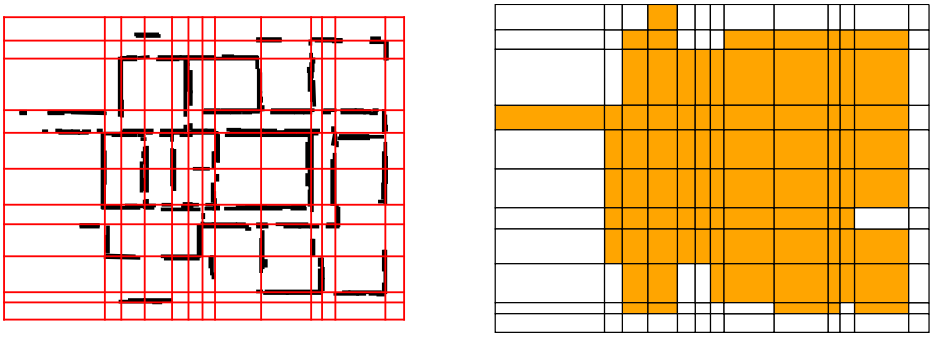
Faces creation and classification The intersections between all representative lines divides the map into different areas, called *faces*. The faces represent small portions of the environment. The set of faces is called O and each face is represented as polygon. The next step is to separate the faces that are internal to the contour of the map from the ones that are external (robot has not completely explored them yet). The division is done by performing the intersection between the faces and the contour; for each face



(a) (b)

Figure 4.9: Figure 4.9a shows the original image. Figure 4.9b shows the representative lines over the original map m.

$o \in O$ if the area of the intersection between the polygon that represents the contour and the polygon that represents o is greater than a certain threshold, the face o is considered internal, external otherwise. In Figure 4.10 the classification of the faces are shown: in yellow the ones classified as internal meanwhile the ones classified as external are white.

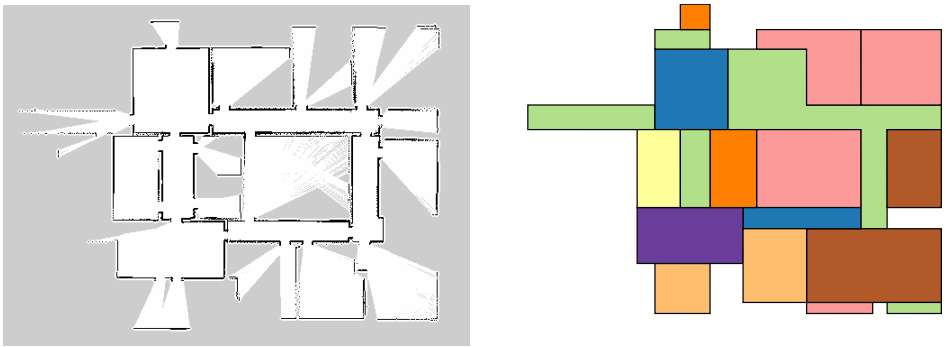


(a) (b)

Figure 4.10: Figure 4.10a shows the representative lines. Figure 4.10b shows the created faces and their classification.

Rooms creation The nex step is to reconstruct the different rooms of the environment by merging the faces that belongs to the same room. The *DBSCAN* algorithm [10] is used to cluster internal faces together using as metric weight $w(o, o')$, where o, o' are adjacent faces and where the metric

weight $w(o, o')$ represents how much of the common edge between the two faces is covered by a wall (the greater the edge is covered by line segments, the stronger the hypothesis that there is a wall between the two faces). If two adjacent faces have a seen wall that separates them (high weight w), then it is unlikely that the two faces belong to the same room. The result is a partition of faces where each set is a room and the elements are the faces of the room. Finally, each set of faces F_i is merged to create the layout corresponding to room r_i. The layout is then defined as $\mathcal{L} = \{r_1, ..., r_n\}$. In Figure 4.11 an example of reconstructed rooms is shown.



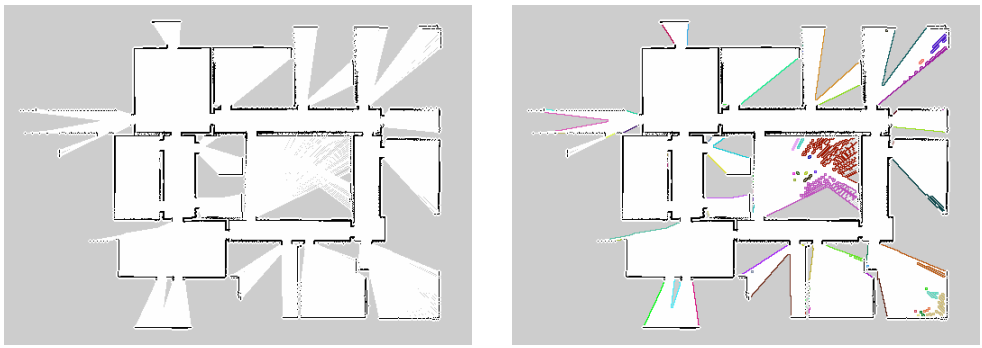
(a) (b)

Figure 4.11: Figure 4.11a shows the original map. Figure 4.11b shows its reconstructed layout.

4.3.2 Identify partially-observed rooms

Given the layout $\mathcal{L} = \{r_1, ..., r_n\}$, the next step is to identify which rooms have been partially observed. Using the frontiers sent from the exploration module, for each face $o_{i,j}$ of each room r_j, the algorithm counts the number of occurrences of points belonging to the different frontiers. The frontier that have the highest number of occurrences is selected as candidate frontier for the cell. If the number of occurrences of the candidate frontier is higher than a certain threshold (in the experiment this value has been setted to 5), the face $o_{i,j}$ is setted as partial and the candidate frontier is assigned as frontier of the face. Finally, if a room has at least one partial face, the room

is considered partially-observed and the frontier associated is the one with the highest number of occurrences. In Figure 4.12 an example of identified frontiers is shown. Indicating as f_i the frontier, $f_i \in F$ with F the set of



(a) (b)

Figure 4.12: Figure 4.12a shows the original map. Figure 4.12b shows the frontiers of the map.

all the frontiers, $list_{f_i}$ the list of points of the frontier f_i and *threshold* the threshold for a face to be considered partial, the Algorithm 5 shows the operations performed.

In Figure 4.13 the classification of rooms of partially observed is shown.

4.3.3 Extract partially-observed rooms from frontiers

The next step is to use the layout $\mathcal{L}$ and the frontiers to find partially-observed rooms. At first, the subset of partially-observed rooms $\mathcal{L}_P$ is created. For each room $r_i \in \mathcal{L}_P$ with observed faces O_i, the partial faces adjacent to a least one face in O_i that contain some points of the frontiers are added to O_i, obtaining set O_i'. The check whether a faces close to O_i contains a frontier is done using the Algorithm 5. In Figure 4.14 the result of this first step is shown. Then the set EF of the external faces next to any of the room $r_i \in \mathcal{L}$ is created (r_i can be both a fully-observed or a partially-observed room). The next step is to apply the Algorithm 5 to all the faces $f \in EF$ to find which faces contain a frontier. The subset of faces $EF_P \subseteq EF$ containing the partial faces of EF is created. The assumption is that if an external

Algorithm 5 Assign frontier to a face

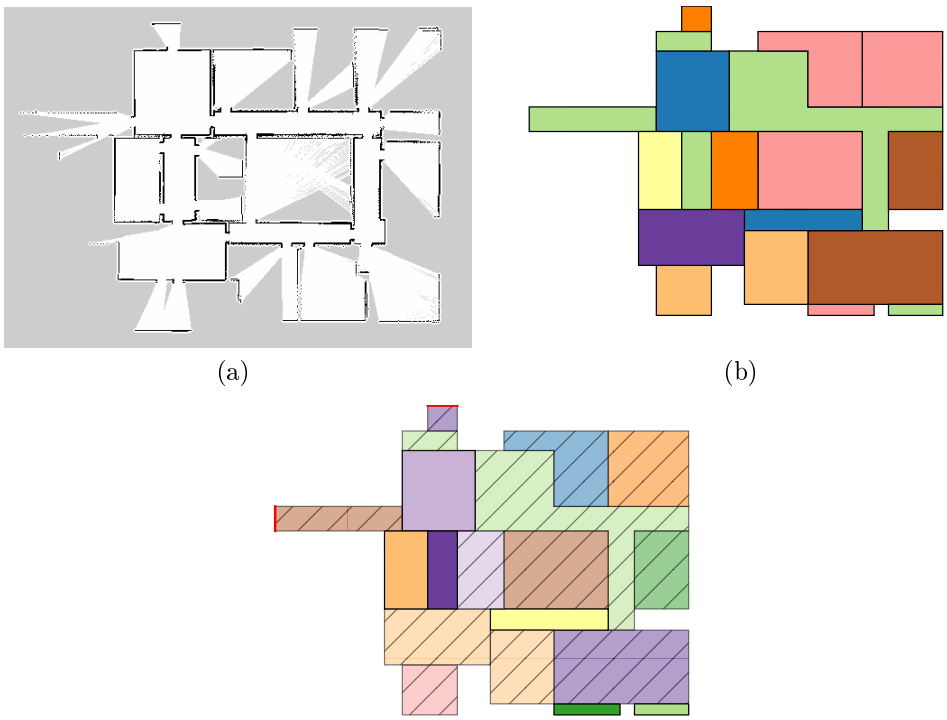
```
1:  function assignFrontierToFace(o)
2:      for all f_i ∈ F do
3:          frontierList[f_i] ← 0
4:      for all Point ∈ o do
5:          for all F_i ∈ F do
6:              if Point ∈ list_{f_i} then
7:                  frontierList[f_i] ← frontierList[f_i] + 1
8:      max ← 0
9:      frontier ← 0
10:     for all f_i ∈ F do
11:         if frontierList[f_i] > max then
12:             max ← frontierList[f_i]
13:             frontier ← f_i
14:     if max ≥ threshold then
15:         return frontier, max
16:     return None, 0 no frontier has to be assigned
```

face contains a frontier it means that there could be a potential place that can be explored. The next steps are applied for all the rooms $r_i \in \mathcal{L}$:

1. Create the subset $EF_{P_i} \subseteq EF_P$ set of faces adjacent to the room r_i.

2. If the room r_i is not classified as partially-observed this step is skipped. Check for each face $f \in EF_{P_i}$ if the frontier assigned is equal to the frontier assigned to the room r_i. If positive, the face f is added to the room r_i and the face f is removed from the set EF_P.

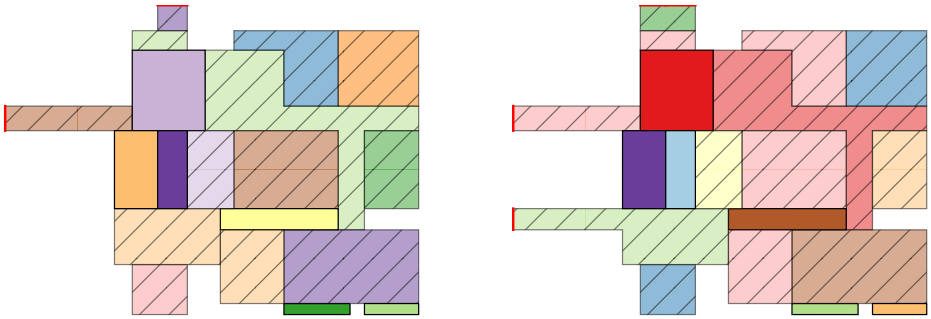
If at the end of all the steps the set EF_P is not empty, it means that there are some faces that could not be assigned to rooms but that can still be explored. The next step is finally the creation of new rooms. For each face $f \in EF_P$ a new room is created with f being the only face of the new room. The new rooms are added to the layout $\mathcal{L}$. In Figure 4.15 the result of this second step are shown.



(a) (b)

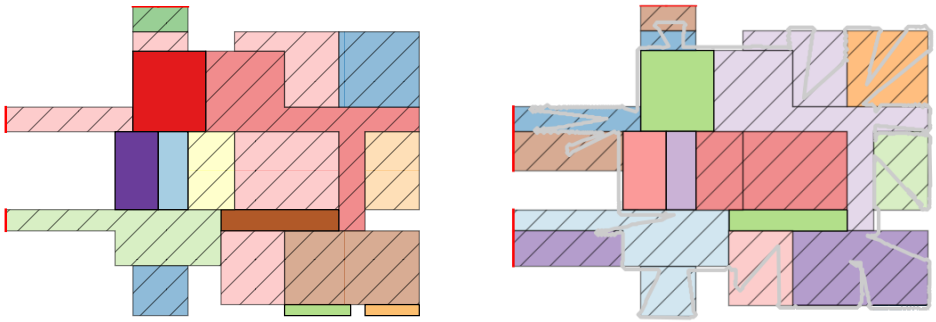
(c)

Figure 4.13: Figure 4.13a shows the map of the environment. Figure 4.13b shows the rooms. Figure 4.13c shows the rooms labelled as partially-observed rooms. The fully-observed rooms are the ones in plain colour.



(a) (b)

Figure 4.14: Figure 4.14a shows the layout of the environment. Figure 4.14b shows the result of the application of the first step of the algorithm.



(a) (b)

Figure 4.15: Figure 4.15a shows the layout before the second step. Figure 4.15b shows the result of the second step of the algorithm.

4.3.4 Geometric prediction of the entire layout of partially explored rooms

This part of the algorithm is a little variation of [20]. The objective of this part of the algorithm is to give an hypothesis of how the partial room can completed using the external faces of the environment. The first step is to re-create the set of partial rooms $\mathcal{L}_P \subseteq \mathcal{L}$ in order to include the new rooms created in the previous step. For each room $r_i \in \mathcal{L}_P$ we indicates as O_i the set of faces that composes the room r_i. The next step is to try to enrich O_i with a set of external faces $\hat{O}'_i$. The faces in $\hat{O}'_i$ are calculated as following:

1. $\hat{O}_i$ is created with faces adjacent to O_i, or adjacent to faces which are in turn adjacent to faces in O_i.

2. The faces of $\hat{O}_i$ that are behind an already observed wall of r_i are removed from $\hat{O}_i$.

3. The sets $\hat{O}'_{c,i}$ that corresponds to all the possible combinations of the faces contained in $\hat{O}_i$ is generated.

In the final step the each combination $\hat{O}'_{c,i}$ is evaluated as following:

$$score = \Phi(O_i \cup \hat{O}'_{c,i}) \tag{4.5}$$

With $O_i \cup \hat{O}'_{c,i}$ the union between the original faces of room r_i and the faces in $\hat{O}'_{c,i}$. The function $\Phi()$ is a weighted sum of:

$$\Phi(O_i \cup \hat{O}'_{c,i}) = k_\Upsilon \cdot \Upsilon(O_i \cup \hat{O}'_{c,i}) - k_\Psi \cdot \Psi(O_i \cup \hat{O}'_{c,i}) - k_\Omega \cdot \Omega(O_i \cup \hat{O}'_{c,i}) \quad (4.6)$$

Where:

- $\Upsilon()$ evaluates the consistency of the new room with respect to currently known rooms. Defining $w(l_{j,k}, s) = cov(s, l_{j,k})/len(l_{j,k})$ with s the line segment of the cluster $W_{j,k}$ and $l_{j,k}$ as a general segment belonging to $W_{j,k}$. $cov(s, l_{j,k})$ is the length of the projection of s in $l_{j,k}$ and $len(l_{j,k})$ is the length of $l_{j,k}$. Denoting as $\hat{L}$ the set of representative lines that are the boundaries of $\hat{r}_i$ room created from the union of the original faces O_i of the room r_i with the faces of the combination $\hat{O}'_{c,i}$, $\Upsilon(O_i \cup \hat{O}'_{c,i})$ is:

$$\Upsilon(O_i \cup \hat{O}'_{c,i}) = \sum_{l_{j,k} \in \hat{L}} \sum_{s \in W_{j,k}} w(l_{j,k}, s) \quad (4.7)$$

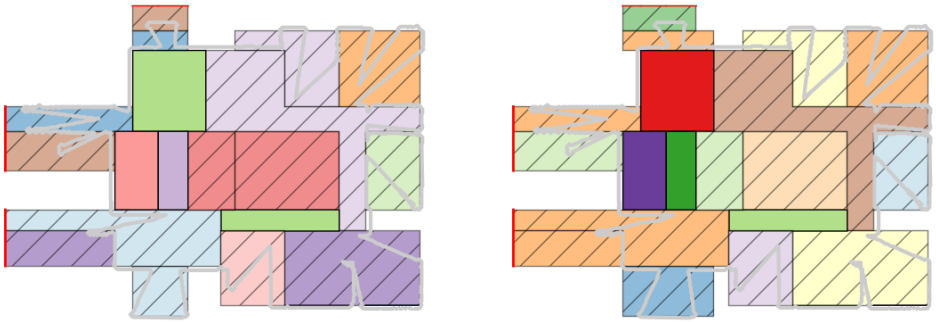
- $\Psi()$ is designed in order to prefer simple (predicted) layouts over complex ones. Defining $Area()$ as the operator that calculates the area of a polygon and $Hull()$ as the operator that computes the convex hull of a polygon and as $\hat{r}_i$ the result of the union of faces O_i with the faces of $\hat{O}'_{c,i}$:

$$\Psi(O_i \cup \hat{O}'_{c,i}) = \frac{Area(Hull(\hat{r}_i)) - Area(\hat{r}_i)}{Area(\hat{r}_i)} \quad (4.8)$$

- $\Omega()$ prefers layouts with a small number of walls. If room $\hat{r}_i$ has n different wall, $\Omega(O_i \cup \hat{O}'_{c,i})$ is calculated as:

$$\Omega(O_i \cup \hat{O}'_{c,i}) = n \quad (4.9)$$

Finally, the set $\hat{O}_i$ is selected from the set $\hat{O}'_{c,i}$ that maximize the Equation 4.5. Once the algorithm has ended the predicted layout is completed. The results of those steps are shown in Figure 4.16.



(a) (b)

Figure 4.16: Figure 4.16a shows the layout before the geometric prediction. Figure 4.16b shows the result of the geometric prediction.

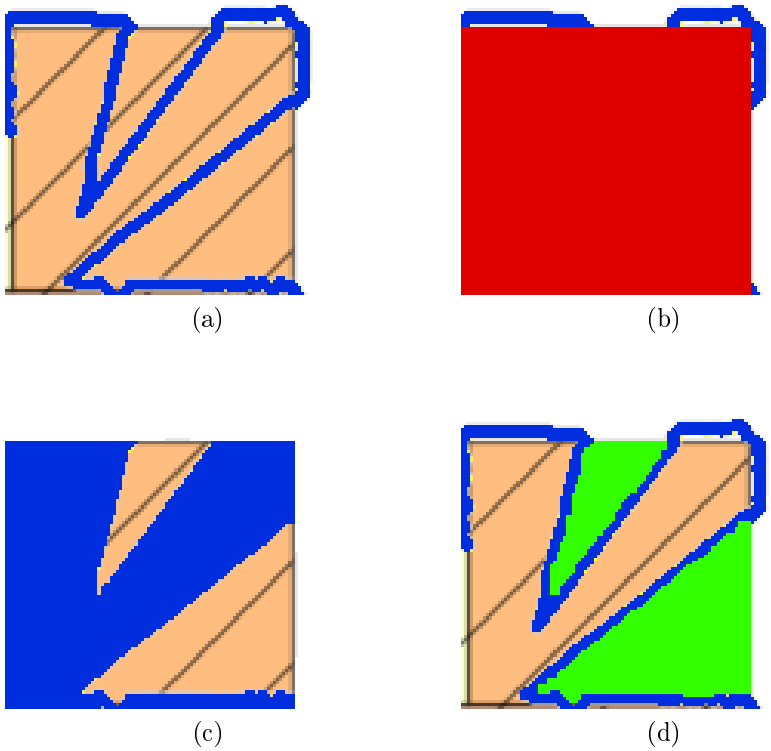
4.3.5 Area calculation and result sent to exploration module

After, the predicted layout has been built, the last step is to evaluate the potential area that can be explored in the partially-observed rooms. Two different methods are used:

- Unknown pixel method: This method calculates the unexplored area for each partial room r_i by locating the room in the original map r_i and by counting the grey pixel (representing unknown cells) in that area. The area calculated with this method is called *real area*.

- Contour method: For each partial room r_i the area is calculated as the difference between the polygon that represents the room calculated as the union of the polygons that represent the faces of the room, and the intersection between the polygon that represents the contour and the polygon that represents the room. An example is shown in Figure 4.17: Figure 4.17a shows the overlap between the polygon of the room and the polygon of the contour, in Figure 4.17b the area of the polygon representing the room is highlighted in red, in Figure 4.17c the area of the intersection is highlighted in blue and finally in Figure 4.17d the resulting area is shown in green. The area calculated with this method is called *external area*. Those areas are used in Section 4.4 to implement

the *early stop*.



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

Figure 4.17: Figure 4.17a shows the overlap between the contour polygon (in blue) and the room (orange). Figure 4.17b shows the area of the polygon representing the room (in red). Figure 4.17c shows the area of the intersection between the two polygon (in blue). Figure 4.17d shows the resulting area calculated with the contour area method (in green).

The areas calculated with the *unknown cells method* are passed to the exploration module that will use them to evaluate the candidate locations and move the robot to the best location.

4.4 Early stop

Commonly, explorations are performed until the entire area of the environment has been completely mapped by the robot. For this reason, many

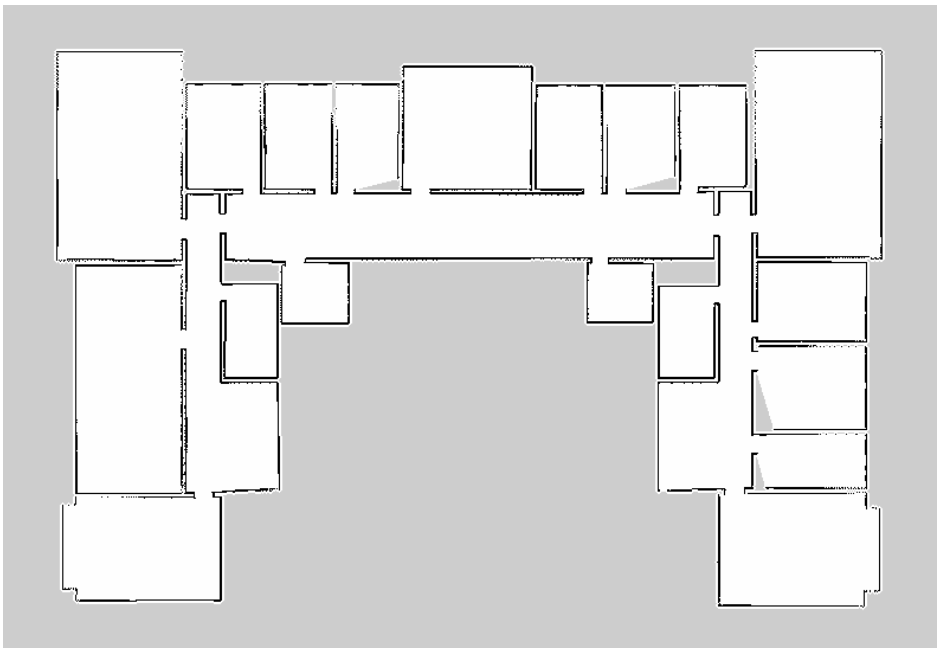
algorithms result in a common behaviour: at the beginning of the exploration the robot is able to quickly increment the map M of the environment by visiting the various frontiers with high information gain. usually, the result is that small scattered frontiers across different rooms are left behind. These small frontiers are then explored at the end of exploration, increasing enormously the time required for the exploration since typically these small portions of environment are often far away one from each other. Usually they represent small gaps like corners. The reconstructed layout $\mathcal{L}$ is able to estimate the missing parts of partially observed rooms and automatically fill the small gaps without actually exploring them. Fixing a threshold on this unexplored area, the algorithm is able to perform an *Early Stopping* (ES) of the exploration when all the predicted areas are lower than the threshold. An example of triggered early stop is shown in Figure 4.18.

Triggering the early stop As described in Section 4.3.5, two different types of areas for each rooms are calculated:

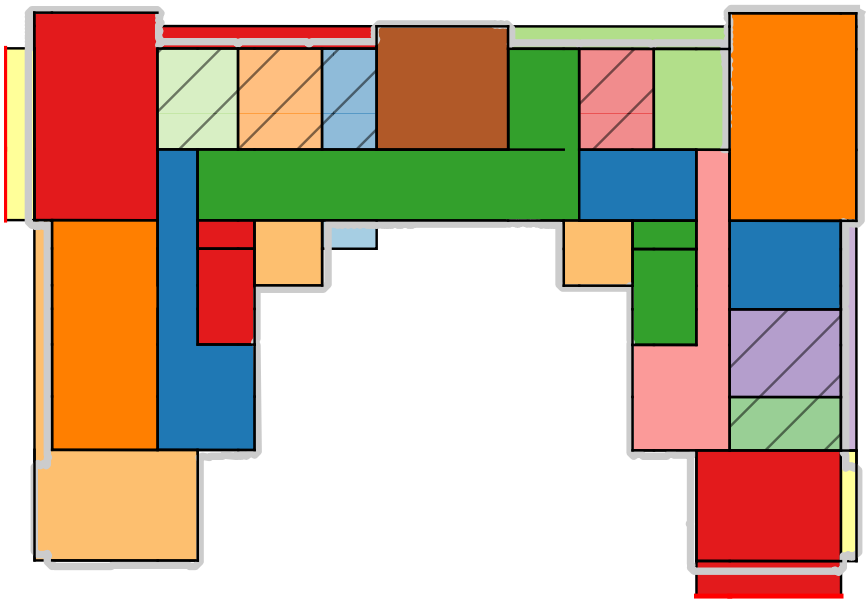
- Real area
- External area

The *External area* is calculated as the difference between the area of the polygon representing the room and the area of the intersection between the polygon representing the room and the polygon that represents the contour of the map. An important features of this method is that all the areas that are internal to the map or, more precisely, that are completely enclosed by a sequence of 'free' and 'occupied' cells are automatically filtered. An example of this feature is shown in Figure 4.19. The result is that only the portions of area that are not enclosed like in Figure 4.19b will be counted. It means that only areas that are 'external' to the map are considered. This feature is helpful since the filtered areas do not lead to new unexplored parts of the environment. These parts of the environment are typically distributed all over the map and the complete exploration of these parts leads to an huge increase of exploration time required. On the other hand, this reasoning can

not be applied to 'external' areas, since they can lead to a new part of the environment not yet explored.

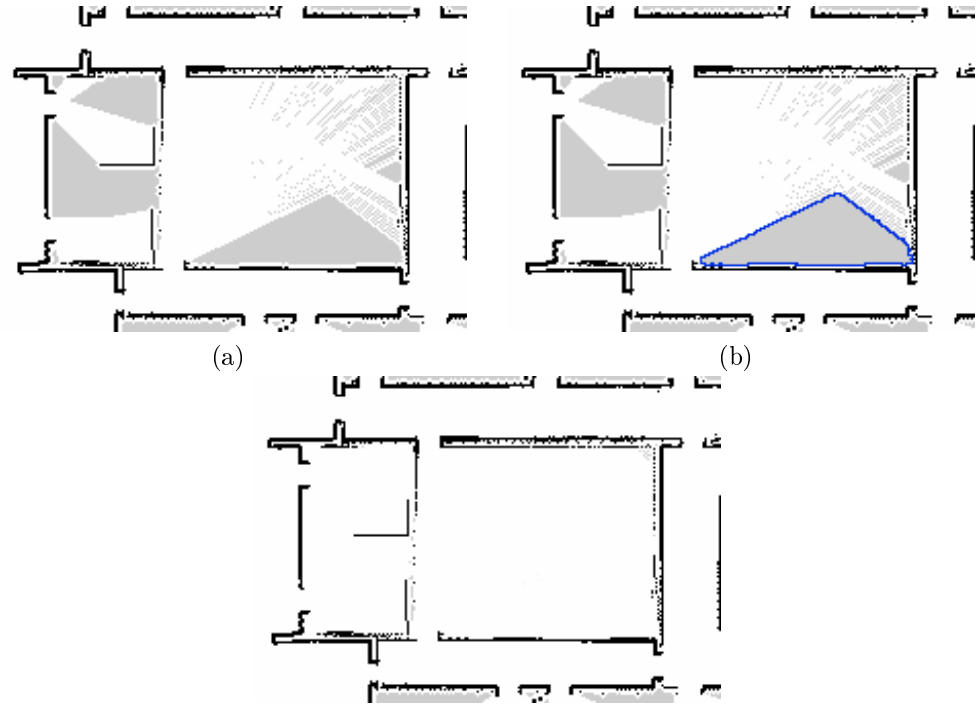


(a)



(b)

Figure 4.18: Figure 4.18a shows the image of the map when the early stop is triggered. Figure 4.18b shows the reconstructed layout when the early stop is triggered.



(a) (b)

(c)

Figure 4.19: Figure 4.19a shows an unexplored part of the environment. Figure 4.19b shows the enclosure in blue. Figure 4.19c show the result of the filter applied.

Chapter 5

Implementation

In this chapter we are going to describe the details about the software implementation of this work of thesis. In Section 5.1 we describe the main software components used for our system, in Section 5.2 we describe what is the navigation stack and how it has been used and built. Finally in Section 5.3 we describe the architecture of our system.

5.1 Software framework

In this section we describe the main software components used in our work. They are:

- **Robot Operating System (ROS)**: it is a modular, tool-based, open-source, language independent framework used to program robotic system [27, 28].
- **Stage**: it is a 2D simulator that simulates virtual worlds and robots and allows to simulate the behaviour of robots in the environments without deploying real robots [11].

5.1.1 Robotic Operating System (ROS)

ROS is a meta-operating system designed for robots [28]. It provides different services like hardware abstraction, low-level device control, implementation

of commonly used functionalities, message passing between processes and package management [27]. In order to support message passing, a communication layer above the host operating system is provided. The main objectives of ROS are:

- Distributed computing: a system built in ROS is composed of a number of processes that can be executed trough a centralized or distributed architecture using a *peer-to-peer* topology.
- Multilingual: ROS has been designed in order to be language neutral. Different programming languages like C++, Python, Lisp and Octave are supported, allowing cross-language development.
- Tools-based: ROS is built with a microkernel design, where a large number of tools are used to build and run the various ROS components.
- Thin: the development of drivers and algorithms is encouraged to be built in stand-alone libraries. This is done in order to allow ROS code to be easily re-used with other robot software frameworks.
- Free and open-source: ROS source code is publicly available.

Nomenclature

The fundamental concepts of the ROS implementation are *nodes, messages, topics* and *services*. Nodes are processes that perform computation. Each one of them is a separated entity that is independent from the others but that can interact with them through the ROS network. Nodes communicate with each other by passing messages, that are strictly typed data structures. Standard primitive types like float and integer types are supported, as are arrays of primitive types. The data exchange between nodes follows the pattern publish/subscribe. A node that wants to send a message has to publish to a specific *topic*, which is a simple string. If a node is interested to the data sent by another node, it will subscribe to the appropriate *topic*. Another pattern implemented to exchange messages is the *services*. A service is a synchronous transaction similar to a client/server mechanism. A service

is defined by a string name and a pair of strictly typed messages: one for the request and one for the response. In order to perform all the exchanges of messages, a main node needs to provide name registration and lookup for the rest of the nodes, otherwise the communication won't be possible. This node is called Master or ROS Master.

ROS tools

Some helpful tools that ROS provides are:

- Silent nodes restart: many times it happens that a specific task requires different nodes that perform different operations to be run. Typically some nodes required for the task are stable meanwhile other nodes require some debugging. ROS allows to shut down a node in order to update its source code and restart it without influencing the behaviour of the other nodes.

- Rosbag: The use of logged sensor data is useful many times to permit controlled comparison of various algorithms. ROS supports this function by providing generic logging and playback of specific topics. One of the advantages of this system is that it does not require any modification of the source code. This is performed using a specific type of log file, called *rosbag*.

- Roslaunch: ROS allows to start group of nodes simultaneously without having to start them singularly.

- ROS packages: in order to support collaborative development, ROS allows to organize the software in packages described by an XML file which also states the package dependencies.

- RVIZ plugin: this plugin is used to visualize specific types like images or maps. Others plugins can be integrated with RVIZ to display more types of data. In our work we used RVIZ to have a 2D representation of the map built by the robot, the current position of the robot, the

path that the robot is following to reach its target location and the current frontiers of the map.

- Namespaces: a stack of software in ROS is a cluster of nodes that work together to accomplish a specific task. Sometimes multiple instantiations of the cluster are required in order to do it without having name collisions. ROS supports this by allowing nodes and entire roslaunch cluster-description files to be pushed into a child namespace.

- Transformations: robots need to track spatial relationships, for example between a mobile robot and some fixed frames of reference for localization. ROS provides a tool that builds a dynamic transformation tree which relates all the frames in the system. This allows the computations of a transformation between different frames by navigating the transformation tree, constructing a path and performing the necessary calculations.

5.1.2 Stage

Stage is a lightweight, highly configurable robot simulator that supports large populations of robots [11] that operate in a 2D bitmapped environment. Stage allows the simulation of tens or hundreds of robots to enable rapid development of systems that will eventually guide real robots, without having to use real hardware and environments. The main aspects of Stage are:

- *Good-enough* fidelity: Stage provide fairly simple and computational cheap models for simulation. Low fidelity simulation can actually be an advantage when designing robot controllers that must run on real robots, as it encourages the use of robust control technique.

- Linear scaling population: All sensor models and actuators are independent from robot population size.

- Configurable, composable device models: Various sensors like lasers and visual colour segments are provided together with a versatile mobile

robot base with odometry. The models are very general, so each model is configured to approximate the target device.

Robot In our work we tested our exploration strategy using a simulated wheeled robot equipped with a laser ranger scanner able to perceive obstacles. The robot uses odometry to record its movements. Both the data from the laser ranger scanner and those from odometry are simulated with Stage and are used to build the map of the environment.

5.2 Navigation stack

In order to autonomously explore the environment a robot must acquire data from the environment, elaborate them and take actions that will move the robot in the environment. The collection of modules that allows all these operations is called *navigation stack*. In Figure 5.1 the navigation stack of

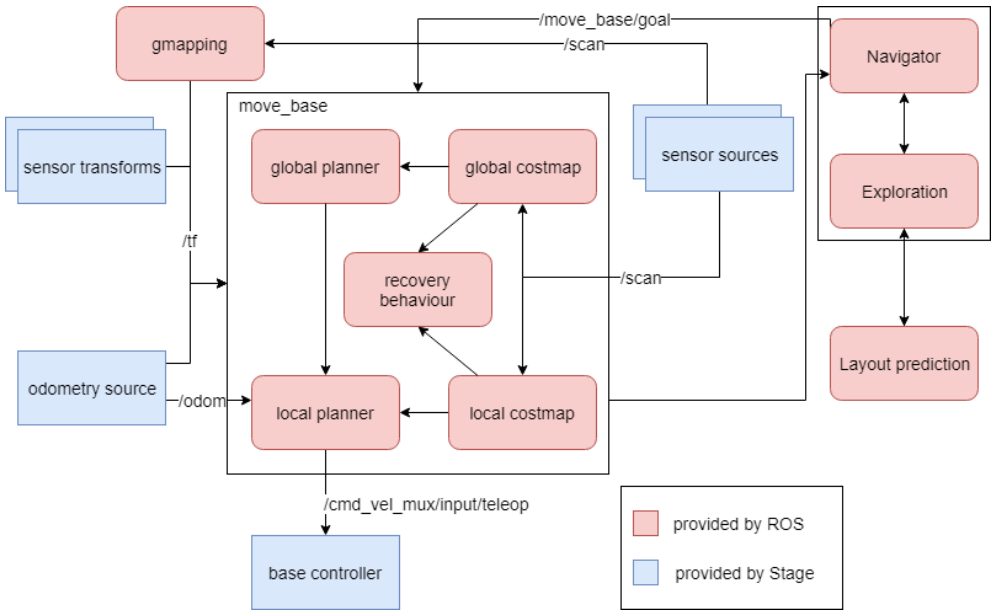


Figure 5.1: The figure shows the navigation stack of our system.

our system is shown. The main packages we used, provided by ROS, are coloured in red meanwhile the components related to Stage are coloured in

blue. The components of Stage are the ones that simulate the perception of the robot in the environment and that move the robot in the simulated environment. Now a description of the main packages used is given. In this section we describe the main packages we used in our work. The packages are divided into two sections:

- ROS packages provided by ROS: these packages are natively from ROS; we used them to integrate some specific functions like navigation and map building.
- ROS packages implemented in our work: these packages have been created or modified by us in order to implement our work.

5.2.1 ROS packages provided by ROS

Gmapping The gmapping package provides a laser-based SLAM [12, 14]. Its functions are:

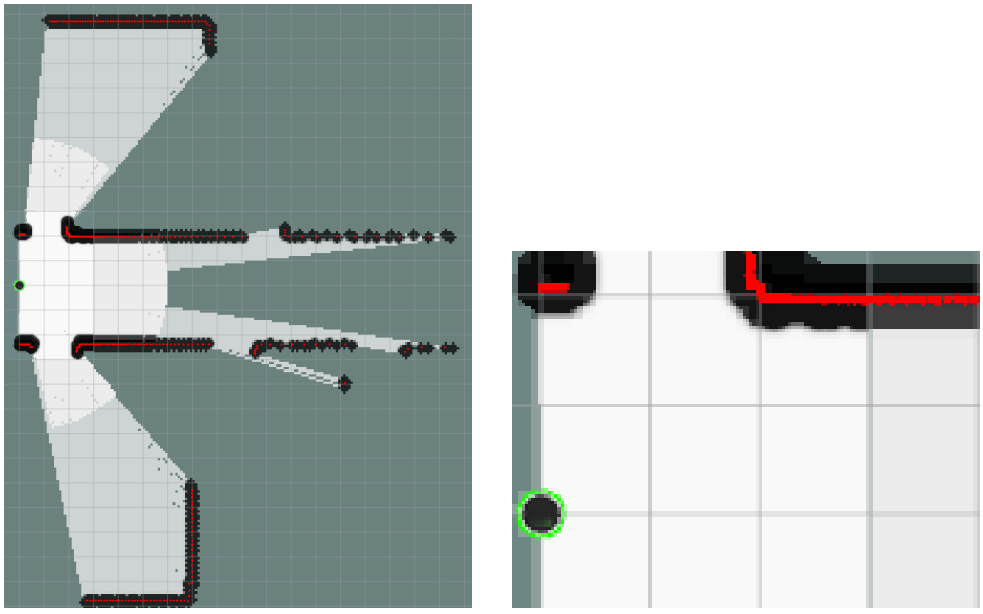
- Map generation: generate the map using the laser sensor's data.
- Localization: localize the robot in the built map.

It generates a map represented as an occupancy grid map. The cells' values are in the range [0,100]. A threshold value is set in order to determine whether a cell can be considered 'free' (below the threshold) or 'occupied' (equal or over the threshold). If the cell's value is 255 (or -1), the cell is classified as 'unknown'. The map is published under the topic "/map".

Move-base Move-base is the package that receives a target position in the environment and tries to create a path from the actual position of the robot and moves the robot to the target position [22]. To perform these operations the Move_base package uses two different planners:

- Global planner: it maintains a global representation (called global map) of the environment seen so far and tries to create a path from the actual position of the robot to the goal and passes it to the local planner.

- Local planner: it maintains a *local costmap* of the environment on the proximity of the robot. The costmap is a particular implementation of an occupancy grid map where the obstacles around the robot are "inflated" with an user specified radius. Figure 5.2 shows an example of a costmap: the obstacle cells are coloured in red, the inflated obstacle are coloured in black. In order to avoid obstacles, the robot's centre point should not pass on the "inflated" part. The local planner uses the global plan produced by the global planner and the local costmap to produce an obstacle-free path and sends the commands for the movement to the robot.



(a) (b)

Figure 5.2: Figure 5.2a shows the an example of costmap. Figure 5.2b shows an example of inflation: the cells representing seen obstacles are in red, in black the inflated obstacles.

Move-base package has been chosen over other packages that perform navigation (e.g., Nav2d [23]) thanks to its flexibility and because it is supported by many real robots systems. This allows to move from simulated to real environments without modifying the system tested during the simulations.

ROS packages implemented in our work

In this section we describe briefly the packages implemented or modified by us to perform our work.

PartialMap This package has been implemented both to produce the goal for the exploration target (see Section 4.2) and to keep track of the actual map of the environment and the position of the robot. The organization of this package is described more in depth in Section 5.3.

MapAnalyzer This package provides the node that implements the method described in Section 4.3.

5.3 Architecture

In this section we describe how we implemented the solution proposed in Chapter 4, illustrating the main components of our system and their interaction with components of the navigation stack described in Section 5.2. In Figure 5.3 we show the components we implemented.

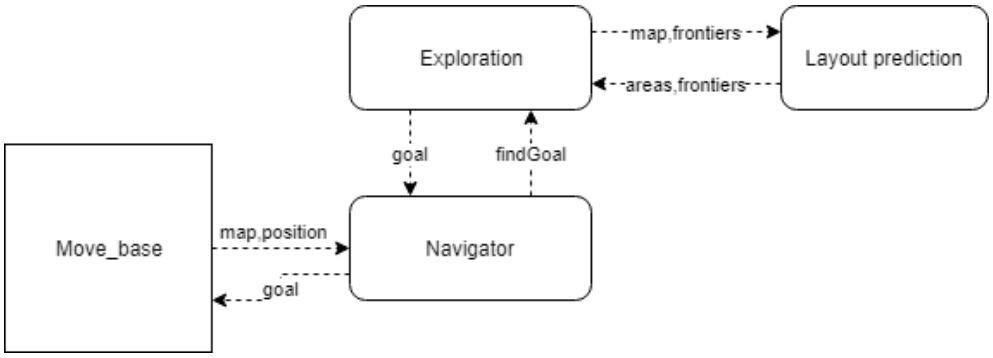


Figure 5.3: The figure shows the architecture of the component of our system.

Navigator This node is organized in the package *PartialMap* and it has been built to provide us a node used for navigation. The main steps of this node are:

- Ask for next goal: it communicates with the exploration node to produce the next exploration goal. The navigator passes both the map and the actual position of the robot to the exploration node. The call for the goal is asynchronous: the exploration node computes the various steps to calculate the goal meanwhile the navigator continues to keep updated the robot position and the map.

- Goal received: once the exploration node has produced the goal, it checks whether the robot has reached its previous goal: if positive, the new goal is passed to Move-base; if negative, the goal is stored and will be sent to Move_base once the previous goal has been reached.

Figure 5.4 shows the interactions with exploration node. The node is implemented in C++.

Exploration This node was re-developed from the node contained in the *Nav2d-exploration* [23] package edited by the authors of [1]. It has been modified to be independent from Nav2d-exploration package since it had compatibility problem with the Move-base package. Then the node has been moved in the *PartialMap* package. This node implements the concept of the exploration module described in Section 4.2. It receives a call from the navigator package, computes all the candidate locations and sends back the target location. An example of its behaviour is shown in Figure 5.5. The main class used by the exploration node is the *Frontier class*. Each instance of this class maintains the representation of a frontier detected, contains the edge cells that compose it, generates the candidate location of the frontier, computes the distance of the candidate location and finally computes the global utility value with respect to the information gain and the distance and performs also the normalization of the criteria. The Frontier class also implements the computation of the information gain using as exploration strategy the method of Next-Best-View algorithm [13], described in Section 2.3.1. This implementation allow to make a comparison between our work and the Next-Best-View algorithm in the following chapter. The algorithm is implemented in C++.

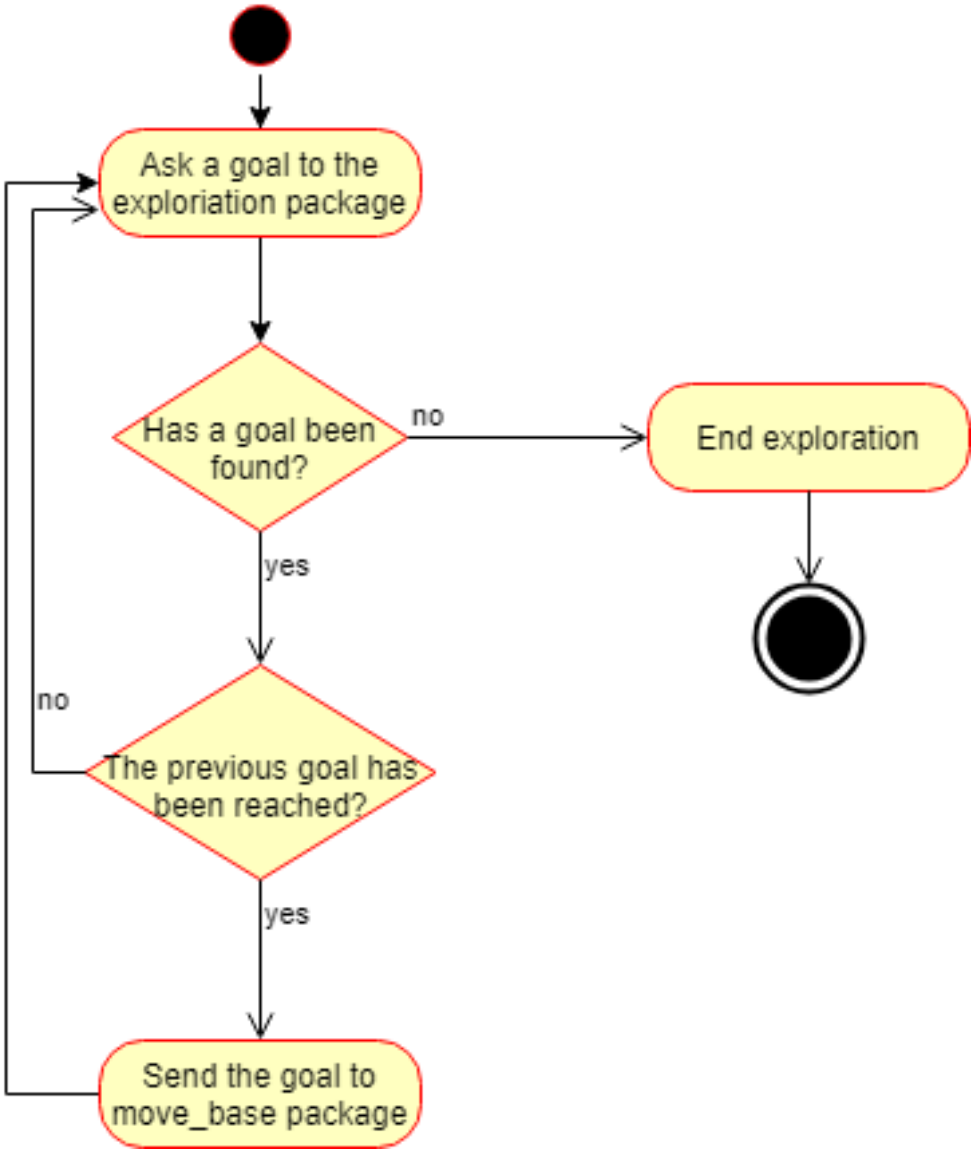


Figure 5.4: The figure shows the interaction of the navigator node with the exploration node.

Layout prediction The algorithm was built by the authors of [20] and was an offline algorithm. It has been modified in order to be used as a ROS node and re-implemented in different points in order to make it more robust. The main objective of the modification was to make the algorithm suitable to be used in an online exploration by speeding up the computation.

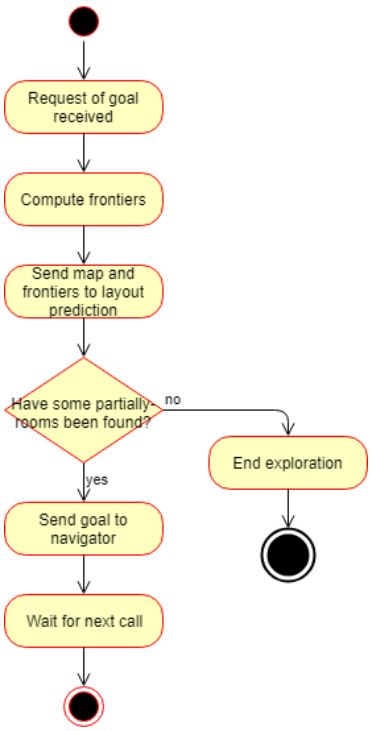


Figure 5.5: The figure shows the various steps performed by the exploration node.

The node created is inserted in package *MapAnalyzer*. It performs all the operations described in Section 4.3 to reconstruct and predict the layout of the environment. An example of its behaviour is shown in Figure 5.6.

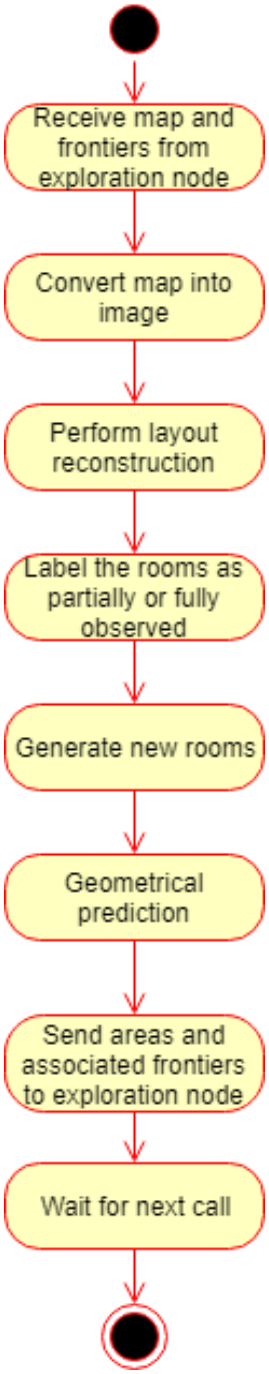


Figure 5.6: The figure shows the various steps performed by the exploration node (see Section 4.3 for the description of the various steps).

Chapter 6

Experimental results

In this chapter we describe the experiments performed to evaluate our work. In Section 6.1 we describe the tools used to evaluate our work of thesis. In Section 6.2 we describe the parameters that need to be set in our exploration algorithm and the method to select the proper values. In Section 6.3 we describe how we evaluated our work and in Section 6.4 the results we have achieved. In Section 6.5 we describe more in general the results achieved with the exploration strategy that we developed and we show how our method can be employed for stopping early the exploration run when only unrelevant frontiers are left to be explored.

6.1 Evaluation tools

In this section we describe the environments used to test our exploration method and the data collected to evaluate the performance.

6.1.1 Environments

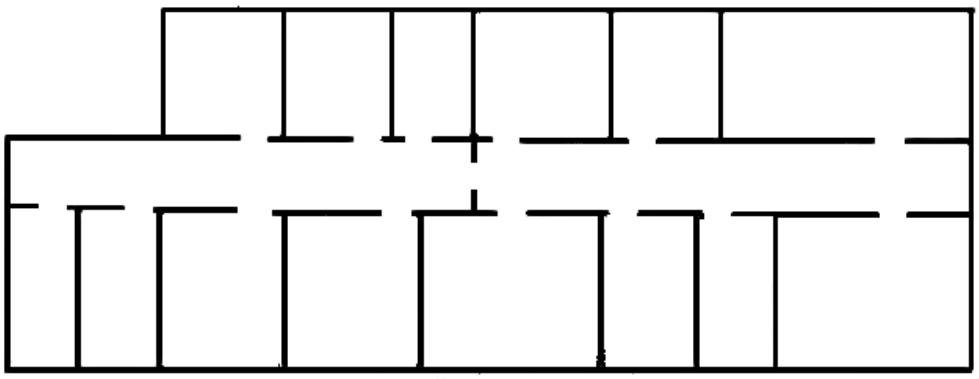
The evaluation of our system is performed in 10 simulated large-scale indoor environments that have been derived from floor plans of real world buildings. The environments are all simulated with Stage. Those environments can be grouped together according to their different levels of complexity:

- Low-complexity environments: these buildings are characterized by a single corridor with all the other rooms attached to it.

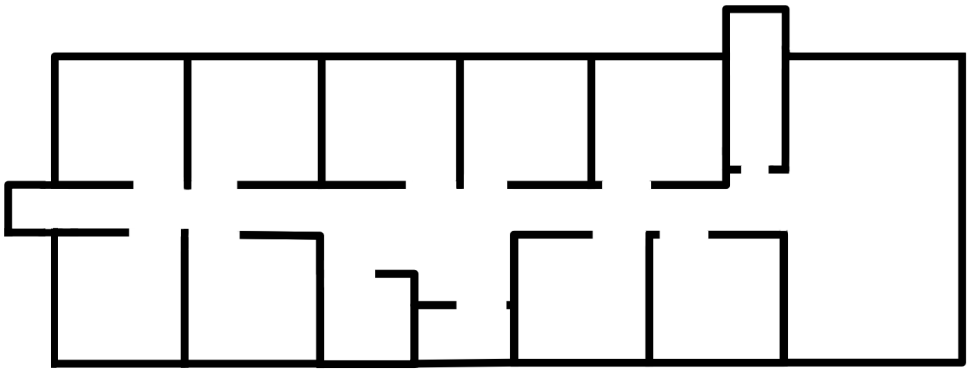
- Medium-complexity environments: these buildings are characterized by multiple rooms, that are arranged in a more complex way, making the prediction of partially seen rooms a little more difficult.

- High-complexity environments: these buildings have a very peculiar disposition and structure of rooms, making the prediction of missing parts of partially seen rooms very challenging.

In Figure 6.1 the maps of the simulated environments are shown.

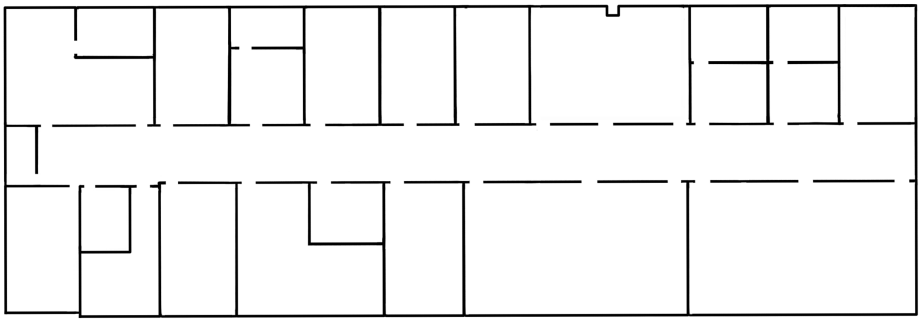


(a) Environment 1 (low-complexity building).

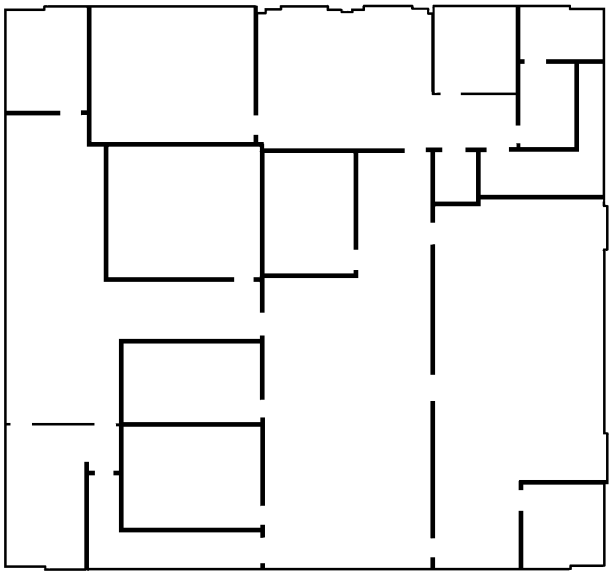


(b) Environment 2 (low-complexity building).

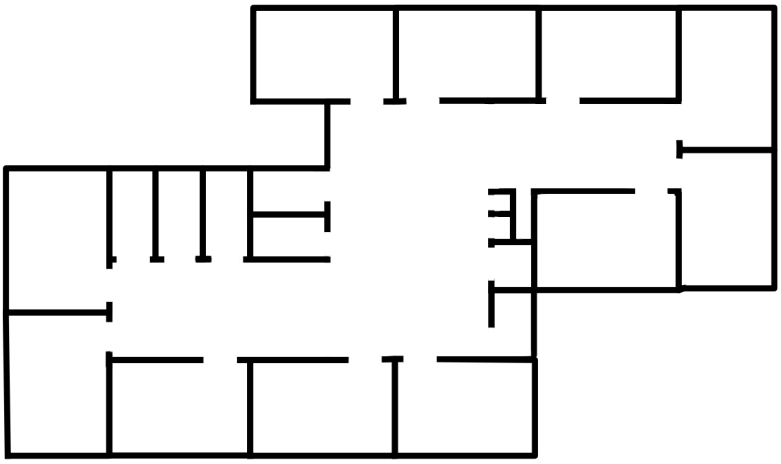
Table 6.1 shows the areas of the various environments.



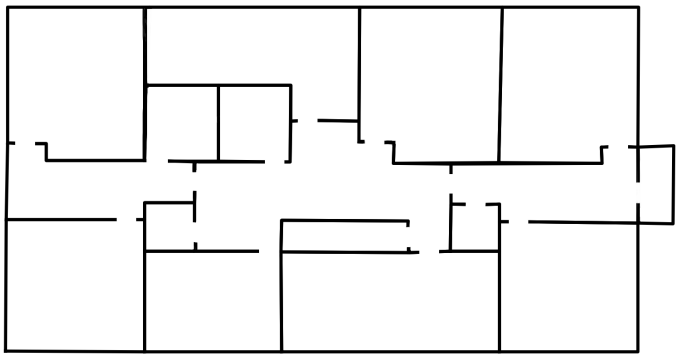
(c) Environment 3 (medium-complexity building).



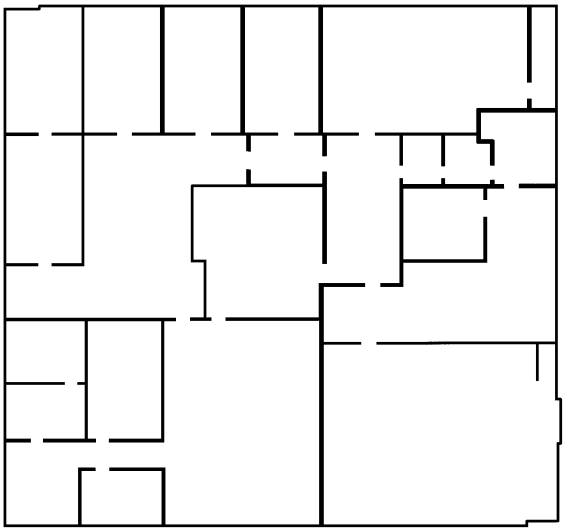
(d) Environment 4 (medium-complexity building).



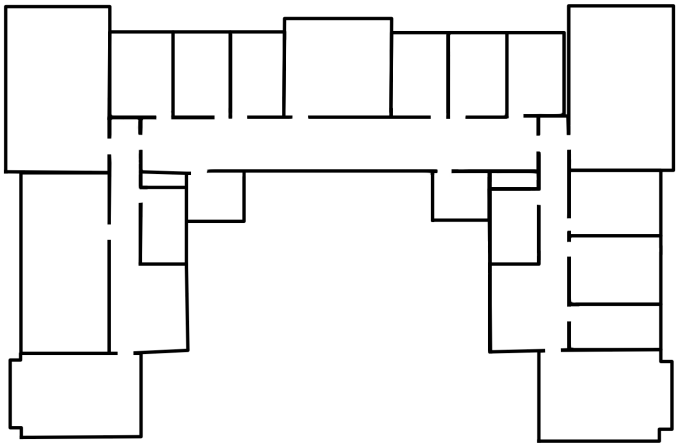
(e) Environment 5 (medium-complexity building).



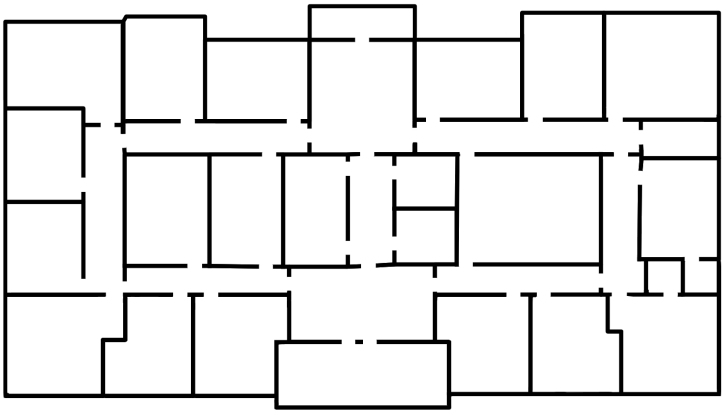
(f) Environment 6 (medium-complexity building).



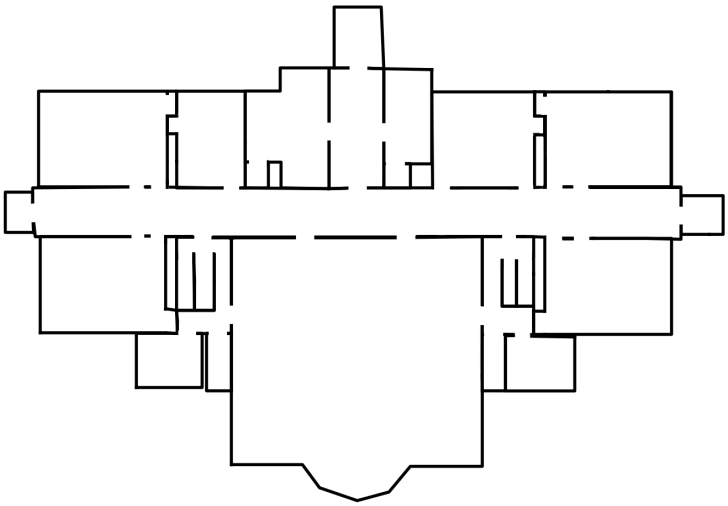
(g) Environment 7 (high-complexity building).



(h) Environment 8 (high-complexity building).



(i) Environment 9 (high-complexity building).



(j) Environment 10 (high-complexity building).

Figure 6.1: Figures show the maps of the simulated environments.

6.1.2 Data collected

In order to evaluate our system we collected different data during the various simulations in the environments. The data collected are:

- Time: the time spent from the beginning of the exploration till the end. The time is measured in seconds.
- Exploration area: the amount of free area of the environment expressed as the percentage of free cells that have been visited during the exploration.

Environment	Area
Environment 1	$1420\,\mathrm{m}^2$
Environment 2	$1000\,\mathrm{m}^2$
Environment 3	$3410\,\mathrm{m}^2$
Environment 4	$1270\,\mathrm{m}^2$
Environment 5	$1740\,\mathrm{m}^2$
Environment 6	$2820\,\mathrm{m}^2$
Environment 7	$3180\,\mathrm{m}^2$
Environment 8	$2150\,\mathrm{m}^2$
Environment 9	$1940\,\mathrm{m}^2$
Environment 10	$1800\,\mathrm{m}^2$

Table 6.1: Areas of the various environments.

- Distance: the total distance travelled by the robot during the exploration. The distance is measured in meters.

- Goal failure ratio: this ratio measures the number of times that the robots select a goal and it is not able to reach it. An example of frontiers that lead to a goal failure is shown in Figure 6.2. The most common reason is that the target location is a wall. This ratio is calculated as:

$$ratio = \frac{\text{number of failed goal}}{\text{total number of goal selected}} \tag{6.1}$$

6.2 Parameters

In this section we describe the main parameters we have set in order to perform the exploration, and the relative values chosen. These parameters are:

- Map resolution: the resolution of the occupancy grid that represents the size of each cells. The resolution in our work is set to 0.1 meters/cell. It means that each cell has a size of $0.1\ m \times 0.1\ m$. An higher resolution would improve the precision of the map's representation, but it would

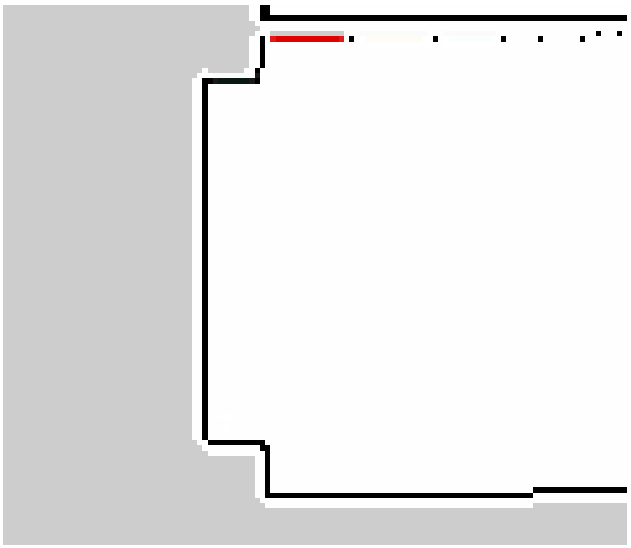


Figure 6.2: An example of a frontier that lead to a goal failure (enlighten in red).

increase the computational effort. On the other hand, a lower resolution would decrease the computation effort and the precision.

- Laser scanner setting: the parameters related to the laser range scanner are:
  - Horizontal field of view: it represents the angle trough which the laser is able to perceive the environment. It is set to 180°.
  - Maximum range: it is the maximum distance at which the laser is able to perceive an obstacle. It is set to 18 m, a range similar to those of indoor laser range scanners as the SICK LMS100 family.
  - Noise: a noise with 0 mean and 0.1 standard deviation is applied to laser scanning.
- α: as described in Section 4.2.4, α is the parameter that weights the two criteria used for the frontiers evaluation: distance and information gain. An higher value gives more importance to close frontiers, meanwhile a lower value would increase the importance of frontiers with high information gain. We decide to set the value to 0.5, in order to give the same importance to the distance and to the information gain.

- Inflation radius: as described in Section 5.2.1, the local planner uses a costmap that 'inflates' obstacles for a safe navigation. The radius at which the obstacle are 'inflated' is 0.4 m.

- Frontier threshold: as described in Section 4.3.2, this parameter is used to identify whether a room has been partially or fully observed based on the dimension of the biggest frontier. This parameter is dependant from the map resolution. We set this parameter to 5, which means that rooms that have frontiers smaller than 0.5 m are considered fully-observed.

- Early stop threshold: in order to perform the early stop (see Section 4.4), a threshold is chosen to check whether it can be triggered or not. We set this parameter to 1 m^2. This value is chosen since it is very unlikely that portion of unseen area smaller than this value can hide other rooms.

6.3 Evaluation procedure

In this section we describe how our work has been evaluated. We performed 10 runs for each environment using always the same values for each of the parameter previously listed, reporting for each environment the amount of explored area as function of time and distance travelled, and calculating mean and standard deviation. We then made a comparison with another algorithm in order to evaluate the benefit of performing a layout prediction on the environment and using it during the exploration. The comparison is made by:

- Layout prediction: it uses the layout prediction of the environment to calculate the information gain. We indicate it as "*with* $\mathcal{L}$".

- Frontier based: it calculates the information gain of each frontier as the potential area that can be seen. This algorithm is based on the algorithm of [13], we indicate it as "*without* $\mathcal{L}$". This strategy represents the state of the art, since it is an exploration strategy that uses only the

map of the environment built during the exploration, without making any prediction about the environment.

- Early stop: it indicates when the early stop function is called, we indicate it as "*ES*". It is used only during the explorations performed using the *with* $\mathcal{L}$ method.

6.4 Results

In this section we illustrate the results obtained in our simulations. For each environment we report the amount of explored area as a function of time and the amount of explored area as a function of distance. We also report the goal failure ratio for both "with $\mathcal{L}$" and "without $\mathcal{L}$" exploration strategies. We illustrate the maps divided by their complexity.

6.4.1 Low-complexity environments

Environment 1

The Environment 1 represents a simple environment composed of a single corridor with many rooms attached to it. Figure 6.3 shows the starting point of the exploration.

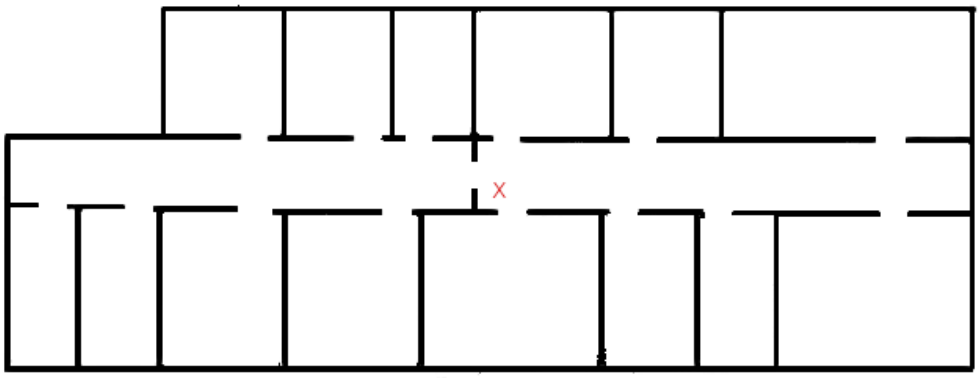


Figure 6.3: The figure shows the starting point of the exploration (in red).

Evaluation In the Environment 1, *with* $\mathcal{L}$ produces a speed up of the exploration. This can be seen in Figure 6.4 in which the explored area is reported as a function of time. This is particularly evident when the explored area reaches around the $\simeq 96\%$, where the *with* $\mathcal{L}$ strategy is constantly better an finish sooner, with a time gain of 19.1% over the *without* $\mathcal{L}$ method. At the same time, the *ES* is triggered allowing the exploration to finish earlier with a gain over time of 30.5%.

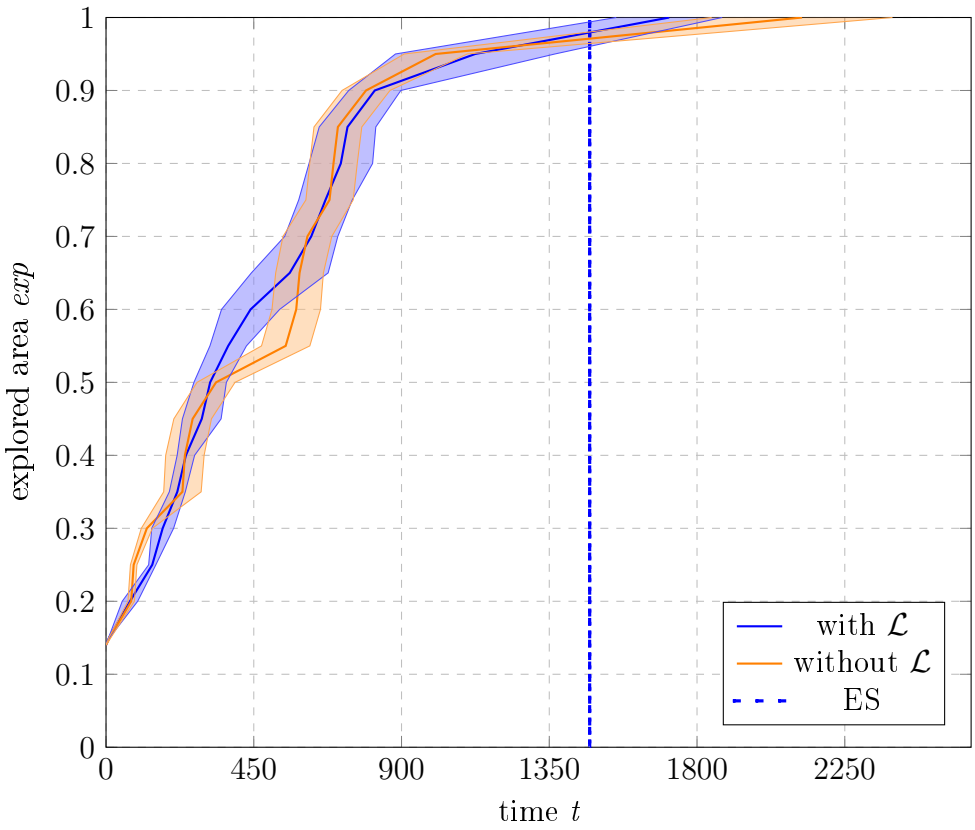
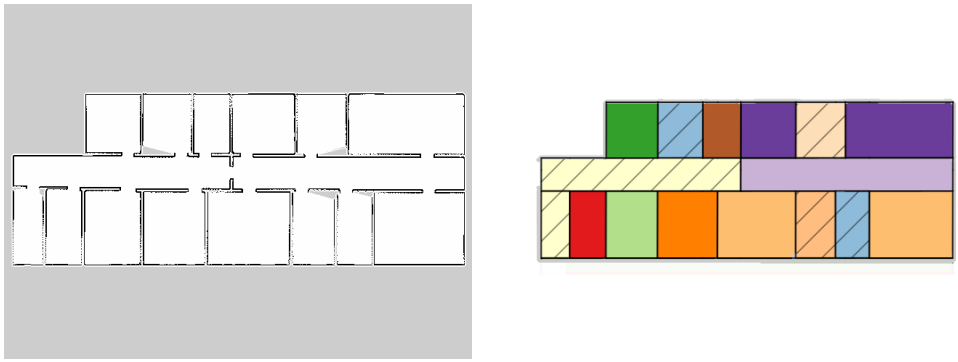


Figure 6.4: The explored area of the Environment 1 as a function of time.

Moreover, Figure 6.6 shows that even the distance travelled by the robot using the *with* $\mathcal{L}$ method is smaller than the one travelled with the *without* $\mathcal{L}$ algorithm. The gain in term of distance travelled is only 6.1%, meanwhile the trigger of *ES* brings a gain of 19.3%. The reconstructed layout, when the *ES* is triggered, is shown in Figure 6.5.

The Table 6.2 shows that the exploration method *without* $\mathcal{L}$ has a failure



(a) (b)

Figure 6.5: Figure 6.5a shows the image of the map of Environment 1 when the early stop is triggered. Figure 6.5b shows the reconstructed layout when the early stop is triggered.

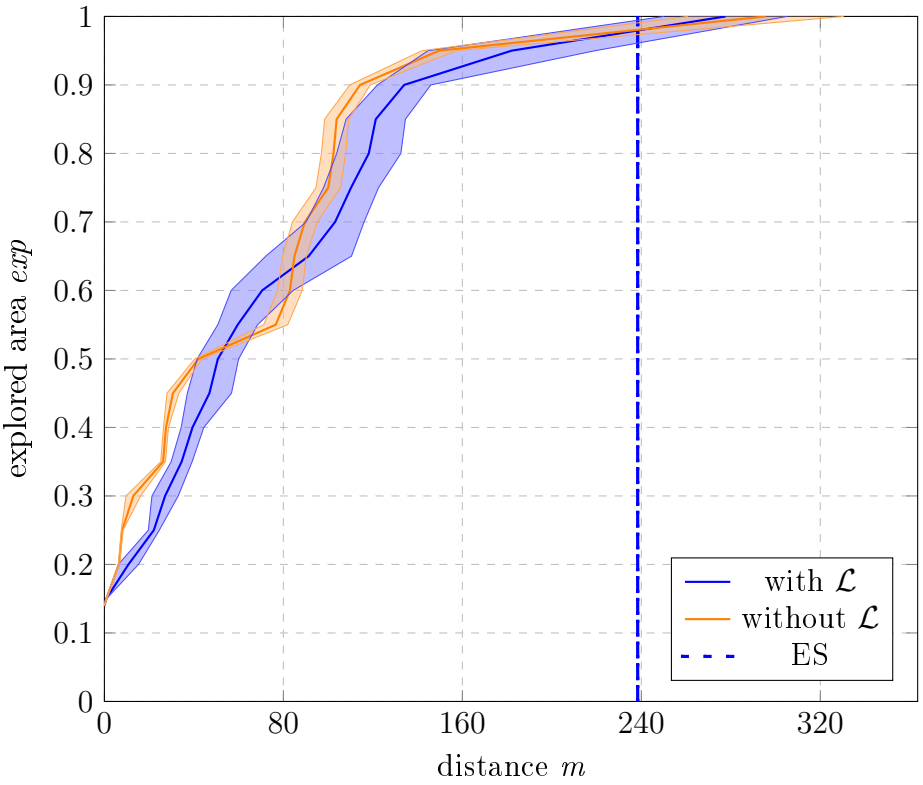


Figure 6.6: The exlored area of the Environment 1 as a function of distance.

rate that is almost 4 times the failure rate of *with* $\mathcal{L}$. A failure can potentially bring the robot to travel an higher distance, however what typically happens

for *without* $\mathcal{L}$ is that it selects a frontier close to its actual location that doesn't bring any new information and that leads to a goal failure (and time lost). Thanks to the smaller goal failed ratio, we can affirm that the *with* $\mathcal{L}$ method is more robust and able to select informative frontiers with more precision with respect to the *without* $\mathcal{L}$ method.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
7.8 %	5.2	27.1 %	13.3

Table 6.2: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 1.

Environment 2

Environment 2 is similar to Environment 1 and is composed of a single corridor with all the rooms attached to it. Figure 6.7 shows the starting point of the exploration.

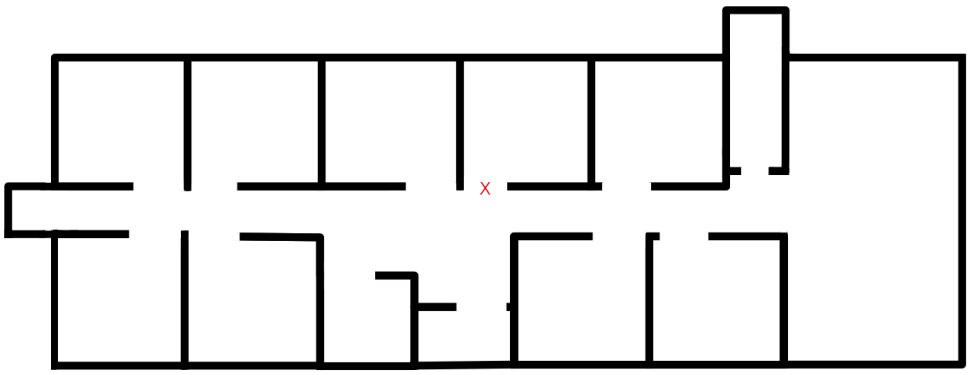
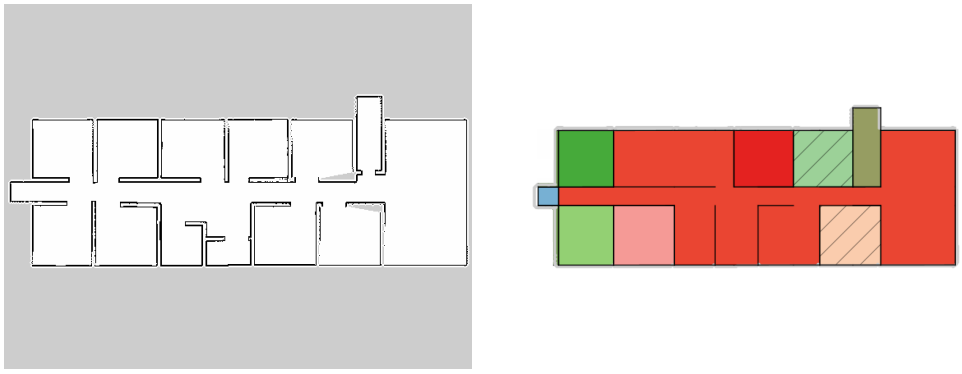


Figure 6.7: The figure shows the starting point of the exploration (in red).

Evaluation Environment 2 is smaller than Environment 1, however the *with* $\mathcal{L}$ strategy still performs better in terms of time requires to explore the 100% of the map respect to the *without* $\mathcal{L}$ algorithm. Figure 6.9 shows that the two algorithms have a similar behaviour along all the exploration

process. The speed up in time is around the 5.5% that increases to 9.6% with the intervention of *ES*. The reconstructed layout, when the *ES* is triggered, is shown in Figure 6.8.



(a) (b)

Figure 6.8: Figure 6.8a shows the image of the map of Environment 2 when the early stop is triggered. Figure 6.8b shows the reconstructed layout when the early stop is triggered.

In terms of distance travelled, the situation is a little tricky: the travelled distance of the *with* $\mathcal{L}$ algorithm is slightly higher than the travelled distance of *without* $\mathcal{L}$ algorithm. The increase in term of distance travelled is around the 5.5%, that decreases to 1.1% with the intervention of *ES*.

This behaviour can be explained by looking at the Table 6.3 of the goal failure ratio. The *without* $\mathcal{L}$ method has a goal failure ratio that is more than twice the goal failure ratio of *with* $\mathcal{L}$ algorithm. This leads the robot that use *without* $\mathcal{L}$ algorithm to lose time in order to recover from the goal failure, without however causing an increase in term of distance travelled.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
15.8 %	6.6	36.5 %	14.3

Table 6.3: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 2.

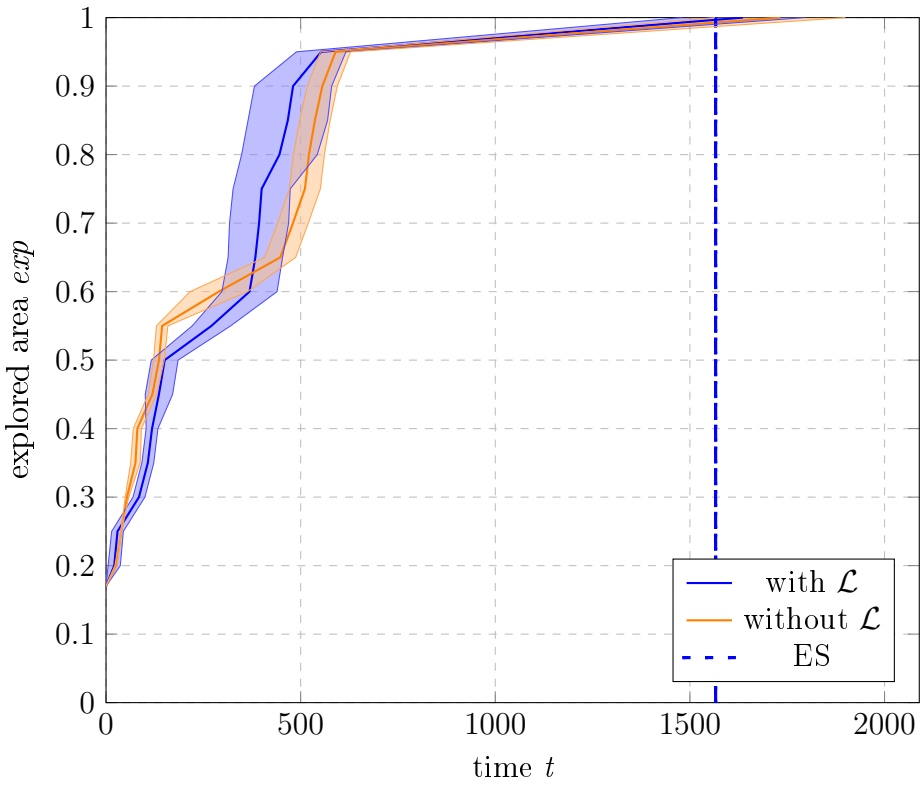


Figure 6.9: The explored area of the Environment 2 as a function of time.

6.4.2 Medium-complexity environments

Environment 3

Environment 3 is a medium-complexity environment composed of a single corridor with many rooms attached to it. The rooms however have some other rooms connected, making the prediction of the environment a little more challenging. Figure 6.11 shows the starting point of the exploration.

Evaluation In Environment 3, the *with* $\mathcal{L}$ algorithm shows to produce a speed up of the exploration. As shown in Figure 6.12, the two algorithms have a similar behaviour along all the exploration process except for the final part, namely when the explored area is around the $\simeq 98\%$ where the *with* $\mathcal{L}$ strategy has better performance and reaches the 100% of the explored area sooner with a speed up of the 11.9%. In this case, the *ES* is triggered almost

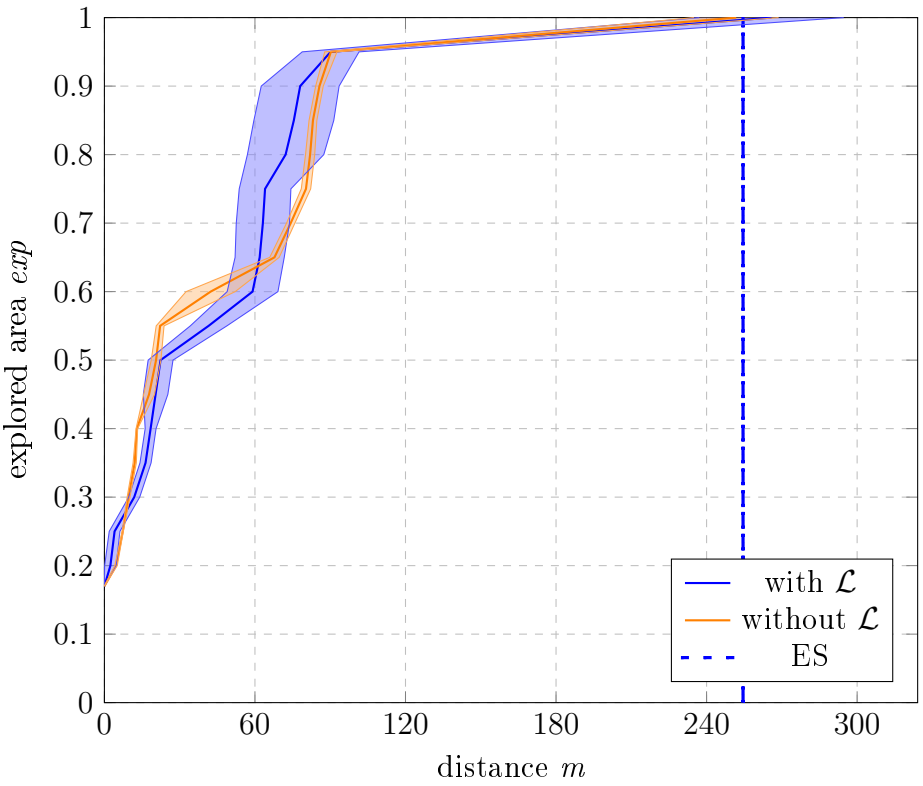


Figure 6.10: The exlored area of the Environment 2 as a function of distance.

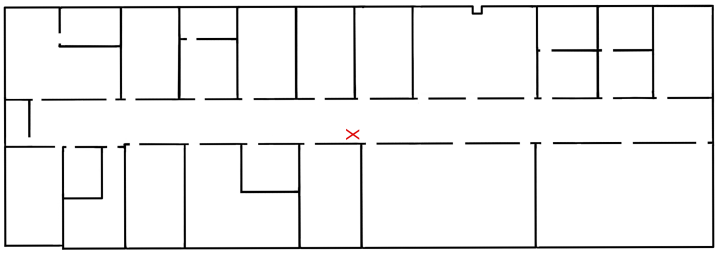


Figure 6.11: The figure shows the starting point of the exploration (in red).

at the same time with the completion of the exploration, without bringing any appreciable improvement.

As can be seen from Figure 6.13, the gain in terms of distance travelled is around the 8.5%.

Also, in terms of failed goal ratio the *with* $\mathcal{L}$ strategy performs much better with respect to *without* $\mathcal{L}$. As can be seen from Table 6.4, our algorithm is

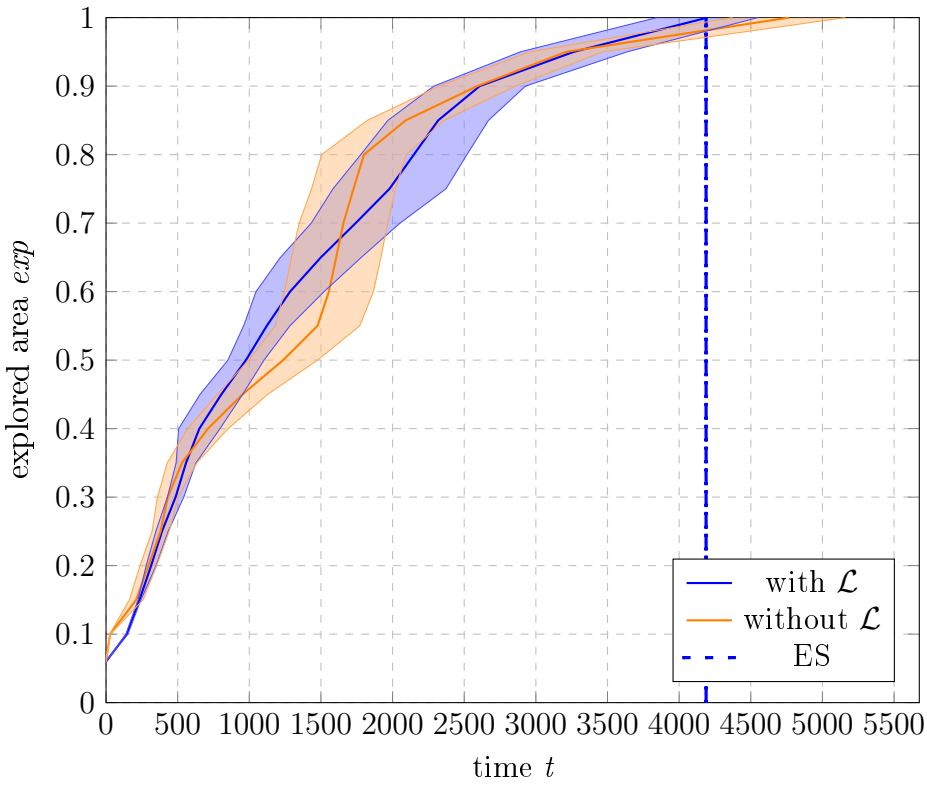


Figure 6.12: The explored area of the Environment 3 as a function of time.

4 times more robust than the other algorithm.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
11.0 %	8.0	42.1 %	14.0

Table 6.4: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 3.

Environment 4

Environment 4 is a medium-complexity environment with many open areas. This environment represents the floor of a home building. Figure 6.14 shows the starting point of the exploration.

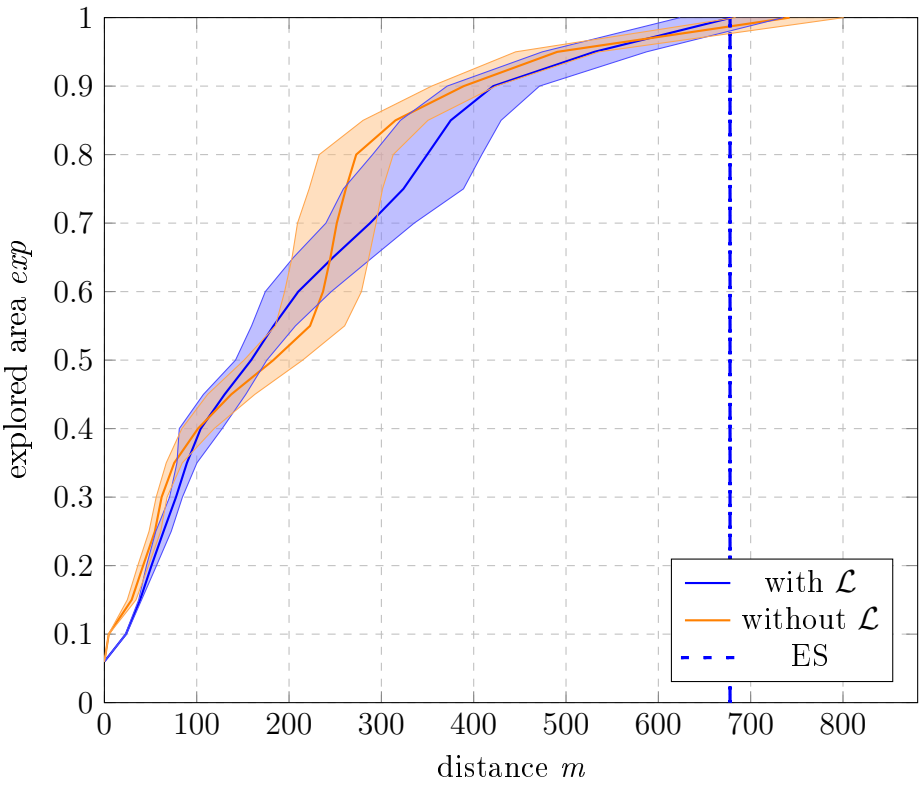


Figure 6.13: The explored area of the Environment 3 as a function of distance.

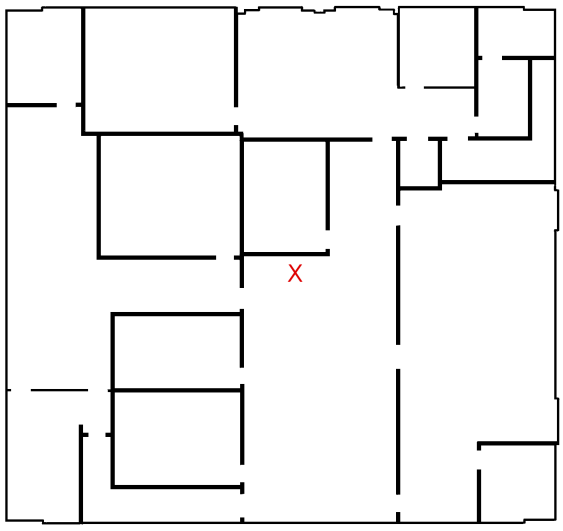


Figure 6.14: The figure shows the starting point of the exploration (in red).

Evaluation In Environment 4, the *with* $\mathcal{L}$ method produces a speed up of the exploration with respect to the *without* $\mathcal{L}$ method. As shown in Figure 6.15, the two algorithms have a similar behaviour up to $\simeq 82\%$ of the explored area, where our method perform strictly better and reach the 100% of the explored area sooner with a speed up of 14.5% of exploration gain. In this case, the *ES* is triggered almost at the same time with the completion of the exploration, without bringing any measurable improvement.

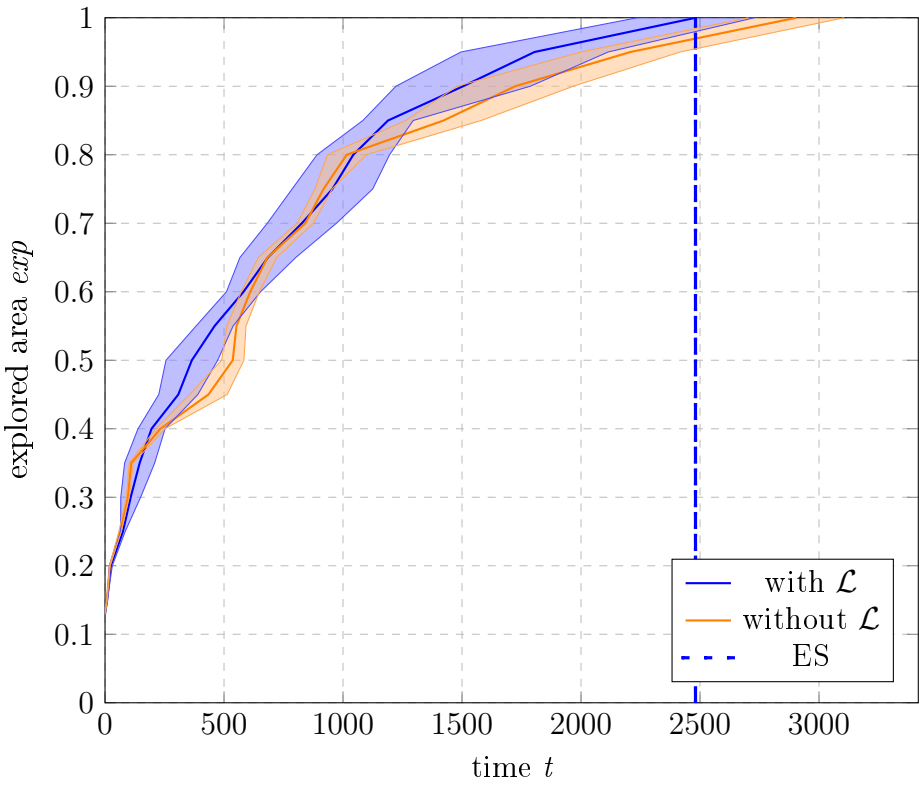


Figure 6.15: The explored area of the Environment 4 as a function of time.

By looking at Figure 6.16, the *with* $\mathcal{L}$ algorithm reaches the 100% of explored area by travelling a shorter distance compared to the *without* $\mathcal{L}$ algorithm, with a gain of 4%.

Finally, by looking at the comparison in terms of goal failure ratio, Table 6.5 reports that the *with* $\mathcal{L}$ method is more robust than the *without* $\mathcal{L}$ method whose goal failure ratio is 6 times higher with respect to our algorithm.

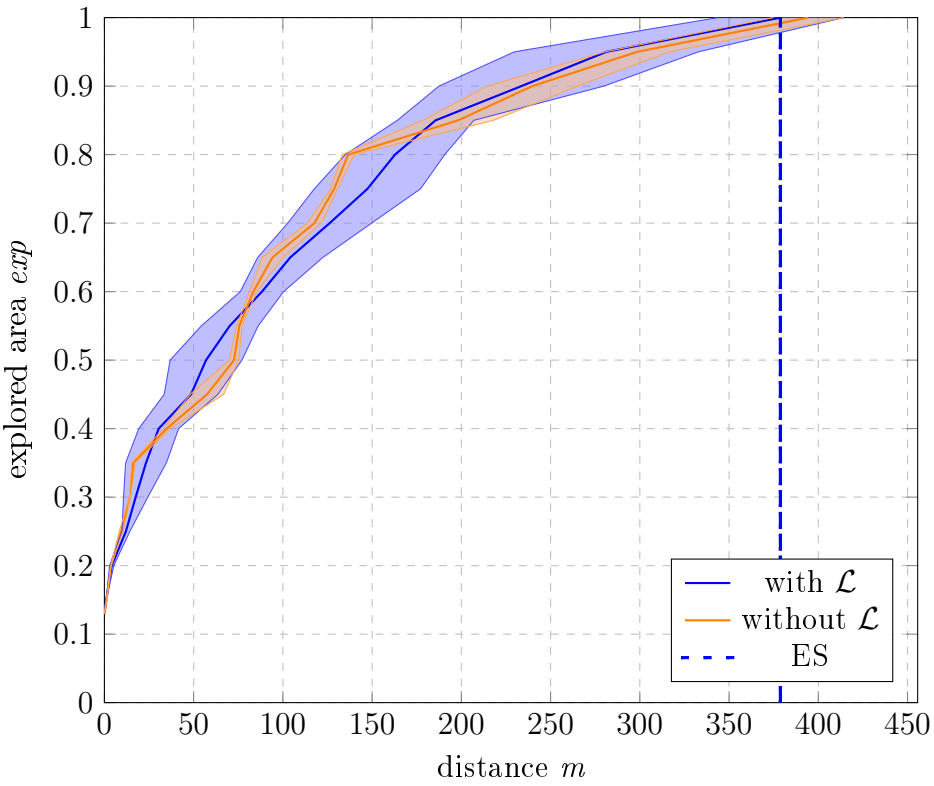


Figure 6.16: The explored area of the Environment 4 as a function of distance.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
4.2 %	3.3	26.3 %	5.1

Table 6.5: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 4.

Environment 5

Environment 5 has a very particular structure: it is composed of a very large central area with a very particular layout that connects to different smaller rooms. It is considered as a medium-complexity environment due to its particular layout. Figure 6.17 shows the starting point of the exploration.

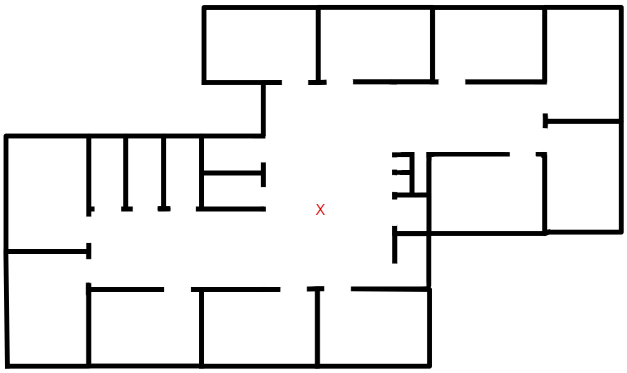


Figure 6.17: The figure shows the starting point of the exploration (in red).

Evaluation In Environment 5, the *with* $\mathcal{L}$ method is able to produce a speed up of the exploration. However, as it can be seen from Figure 6.18, in this particular environment the performance of our algorithm is strictly below the baseline method for almost the exploration process. There is a change of behaviour when the exploration has covered the 99% of the area, when our algorithm is able to complete the exploration sooner with a speed up of the 6.4%. In this map the *ES* is triggered when the exploration is practically ended and the entire area has been covered, and for this reason it does not bring any appreciable improvement.

Figure 6.19 confirms the difficulties of the *with* $\mathcal{L}$ algorithm in this environment. By using the *with* $\mathcal{L}$ method, the robot needs to travel an higher distance compared to the *without* $\mathcal{L}$ method. The increase of distance travelled is equal to the 3.5%.

However, our algorithm is able to reach the 100% of the explored area before the baseline method. This behaviour can be explained by looking at Table 6.6. The goal failure ratio of the *without* $\mathcal{L}$ algorithm is twice the goal failure ratio of *with* $\mathcal{L}$ method. This demonstrates that our algorithm is more robust at the end of the exploration, allowing to select the correct frontiers, and for this reason, to reach the 100% of the explored area sooner.

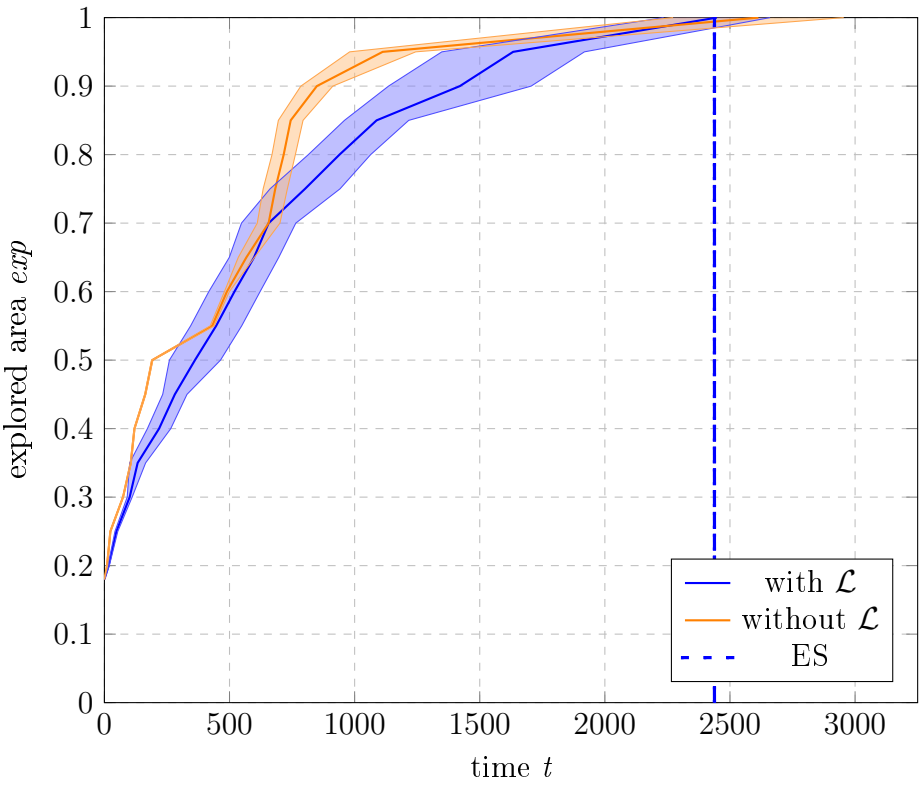


Figure 6.18: The explored area of the Environment 5 as a function of time.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
14.8 %	9.0	33.3 %	6.1

Table 6.6: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 5.

Environment 6

Environment 6 is an environment composed of many open areas. This environment represents the floor plan of a laboratory building. Figure 6.20 shows the starting point of the exploration.

Evaluation In Environment 6, exploration runs made using the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ algorithms have very similar performance during most of time of

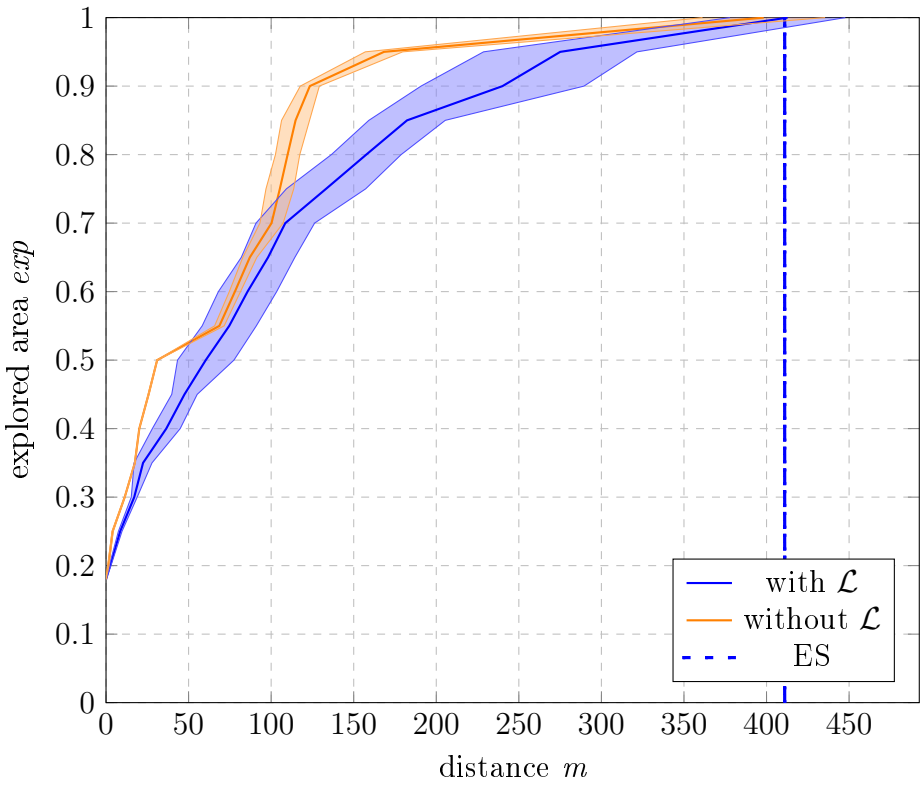


Figure 6.19: The explored area of the Environment 5 as a function of distance.

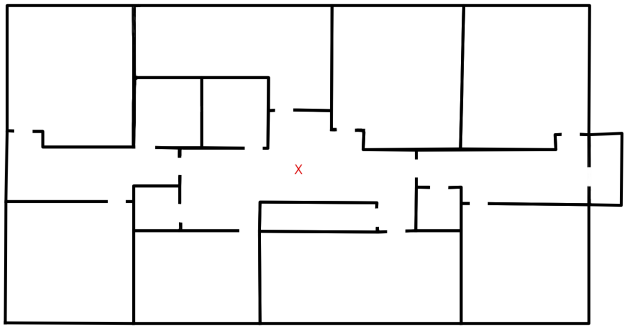


Figure 6.20: The figure shows the starting point of the exploration (in red).

the exploration process. There is a change of behaviour when the exploration process reaches the 97% of explored area, where our algorithm is able to perform strictly better and finish the exploration earlier, with a speed up of the 6.0%. In this environment the *ES* is triggered pretty close to the end of the exploration, however the speed up with respect to the *without* $\mathcal{L}$ strategy

is equal to the 6.7%.

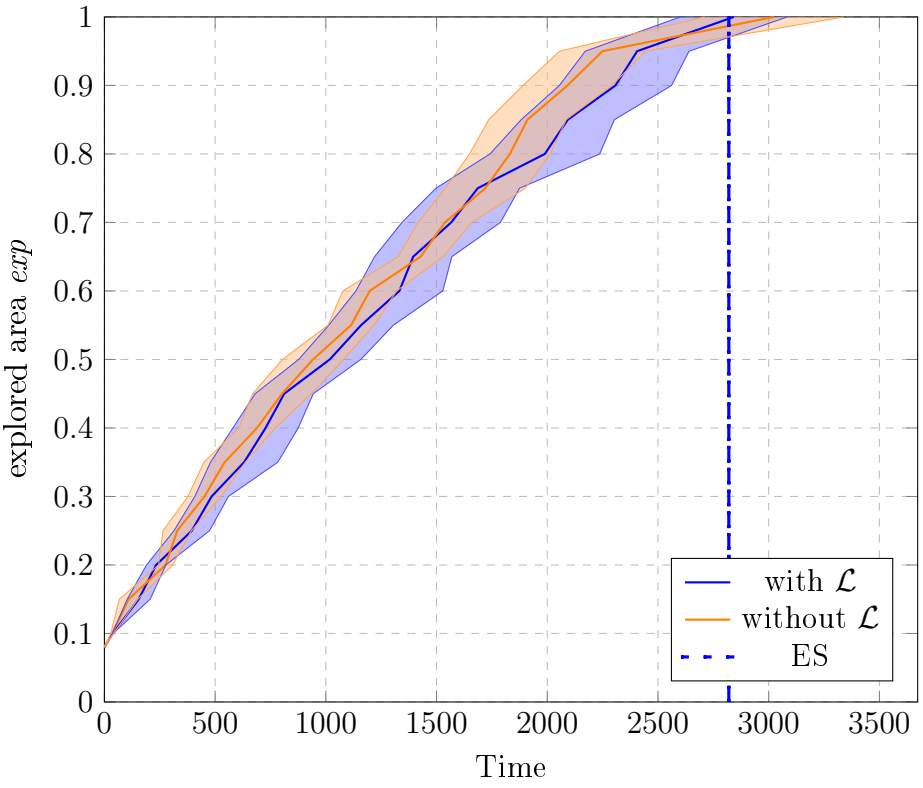


Figure 6.21: The explored area of the Environment 6 as a function of time.

From the distance travelled point of view our algorithm performs slightly worse than the *without* $\mathcal{L}$ algorithm, with an increase of distance travelled of the 4.9% that decreases to 4.0% with the *ES*. This behaviour can be seen in Figure 6.22.

By taking a look to the Table 6.7, we can see that the *with* $\mathcal{L}$ algorithm is more robust than the *without* $\mathcal{L}$ algorithm, with the last one that has a goal failed ratio that is 4 times the goal failed ratio of our method. This behaviour allows our algorithm to have a speed up in terms of exploration time, even if the distance travelled is greater than the one of the *without* $\mathcal{L}$ method.

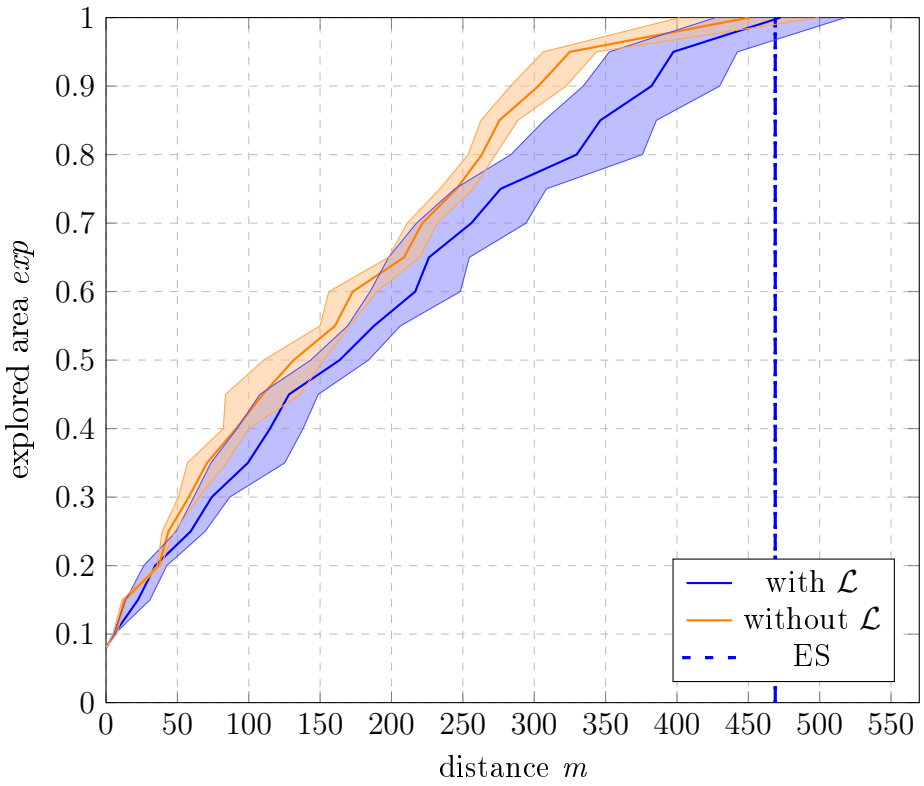


Figure 6.22: The explored area of the Environment 6 as a function of distance.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
9.0 %	7.6	37.6 %	11.7

Table 6.7: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 6.

6.4.3 High-complexity environments

Environment 7

Environment 7 represents an office environment composed of many rooms and open areas. This map has been classified as hard because there are many rooms that are hidden by different walls and are pretty difficult to be predicted using our approach. Figure 6.23 shows the starting point of the exploration.

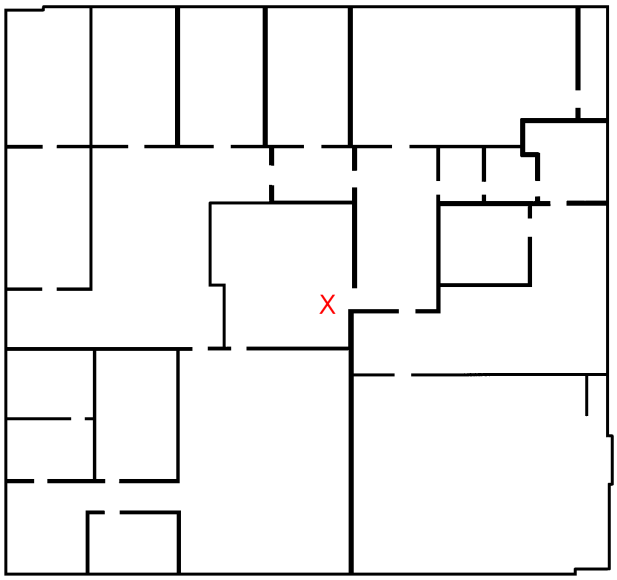


Figure 6.23: The figure shows the starting point of the exploration (in red).

Evaluation As it can be seen in Figure 6.24, the two exploration strategies have a very similar behaviour, that changes only at the very end of the exploration, with the *with* $\mathcal{L}$ strategy that is able to reach the 100% of the exploration a little sooner, with a speed up of the 3.0%. The *ES* is triggered at the end of the exploration, so it does not bring any improvement in terms of speed up.

If we consider the distance travelled, as it can be seen from Figure 6.25, the two algorithms have the same behaviours and both reach the 100% of explored area with the same distance travelled.

As it can be seen from Table 6.8 which reports the failure goal ratio, the *with* $\mathcal{L}$ algorithm is more robust in comparison with *without* $\mathcal{L}$ method.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
9.2 %	9.1	25.3 %	16.8

Table 6.8: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 7.

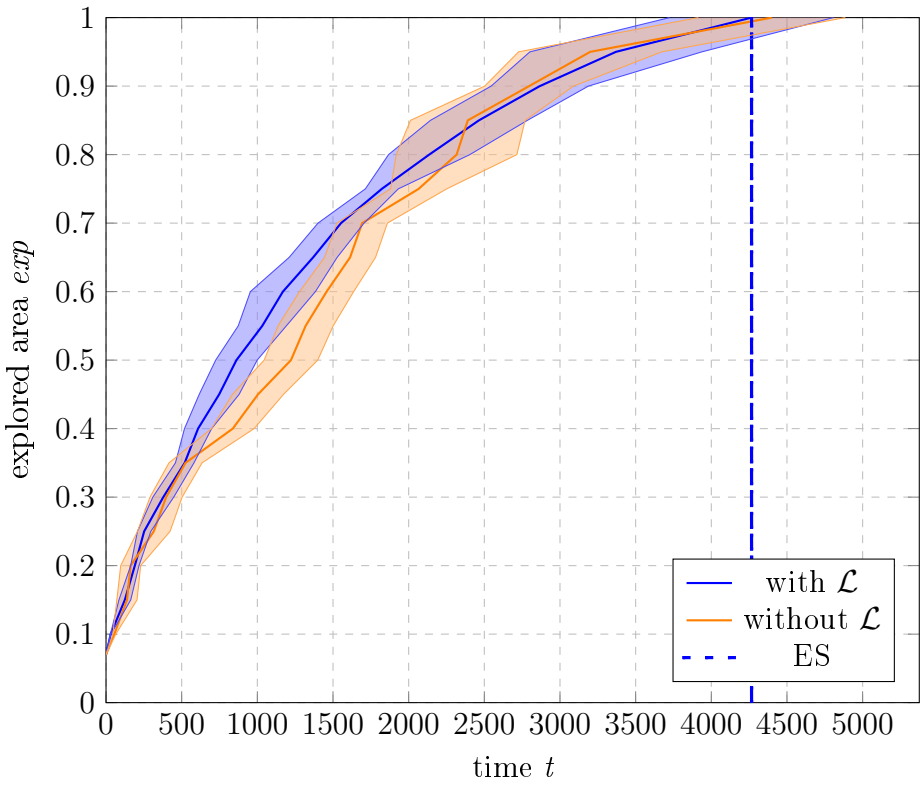


Figure 6.24: The explored area of the Environment 7 as a function of time.

Environment 8

Environment 8 represents a particular environment of a school. It has been categorized as high-complexity environment due to its very particular layout. Figure 6.26 shows the starting point of the exploration.

Evaluation As it can be seen from Figure 6.28 which reports the explored area as function of time, our algorithm performs strictly worse at the start of the exploration, however when the exploration reaches around the 96% of the explored area, our algorithm performs strictly better and is able to reach the 100% of the explored area with a speed up of the 18.2%. The *ES* is triggered before the exploration reaches the 100% of the exploration, leading to a speed up respect to the *without* $\mathcal{L}$ strategy of the 24.4%. The reconstructed layout, when the *ES* is triggered, is shown in Figure 6.27.

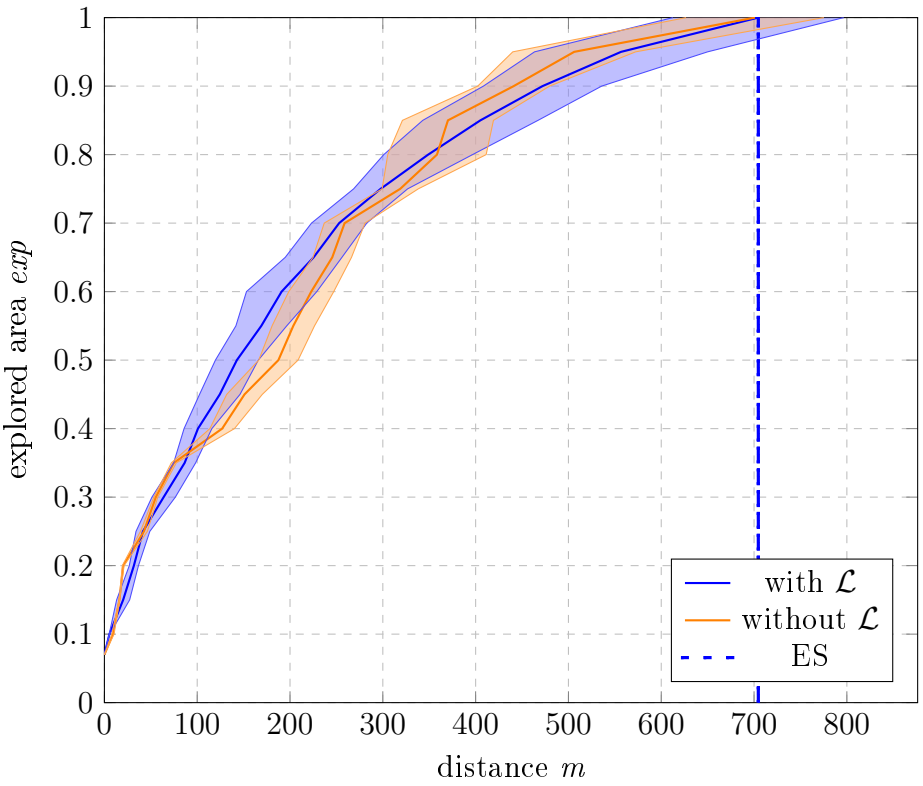


Figure 6.25: The explored area of the Environment 7 as a function of distance.

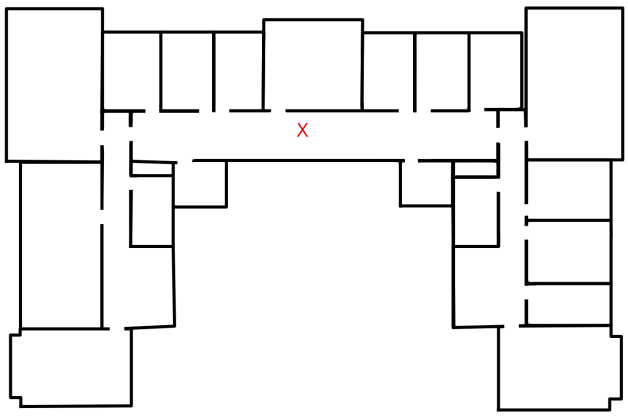
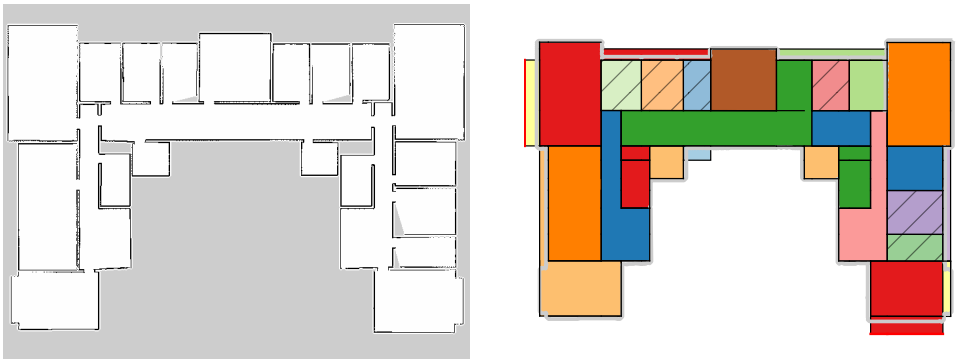


Figure 6.26: The figure shows the starting point of the exploration (in red).

As shown in Figure 6.29, our algorithm is able to enhance the exploration even from the point of view of the distance travelled, travelling a shorter distance with a gain of 15.7% with respect to the *without* $\mathcal{L}$ strategy. If we



(a) (b)

Figure 6.27: Figure 6.27a shows the image of the map of Environment 8 when the early stop is triggered. Figure 6.27b shows the reconstructed layout when the early stop is triggered.

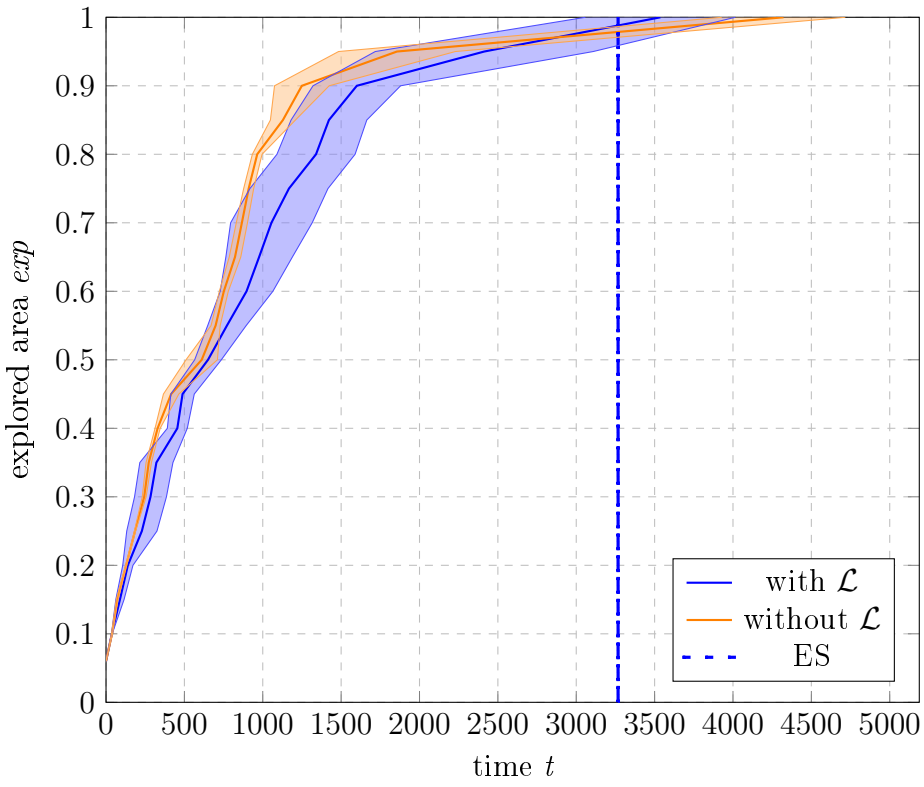


Figure 6.28: The explored area of the Environment 8 as a function of time.

consider the *ES*, the enhancement is increased up to the 22.6%.

Finally, looking at the Table 6.9, we can note how our algorithm is more

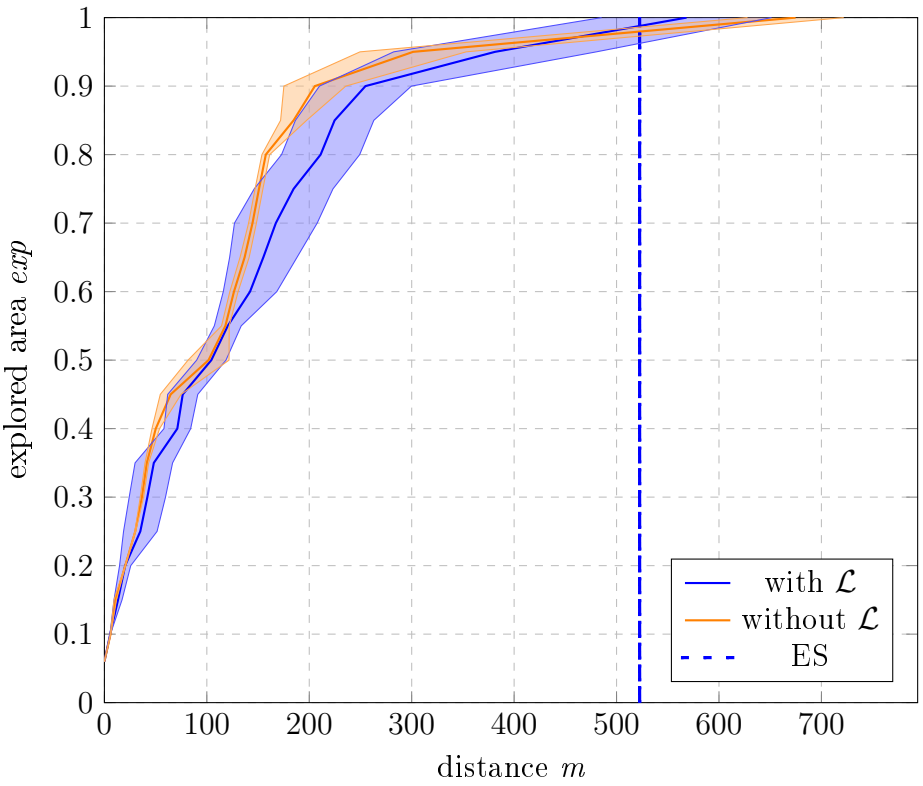


Figure 6.29: The explored area of the Environment 8 as a function of distance.

robust in terms of goal failed ratio, which lead to an increase of the performance.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
23.8 %	9.0	41.4 %	6.4

Table 6.9: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 8.

Environment 9

Environment 9 is a very challenging environment with many different rooms connected trough two main corridors. It represents a typical office environment. Figure 6.30 shows the starting point of the exploration.

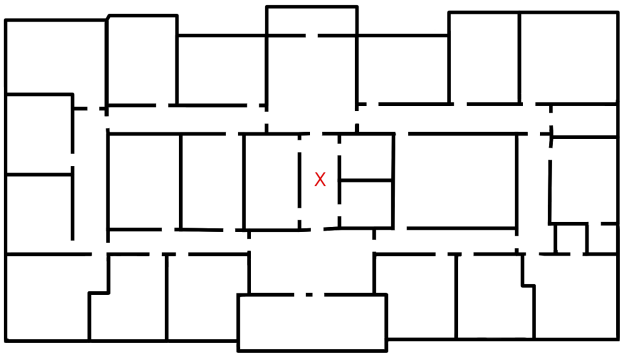
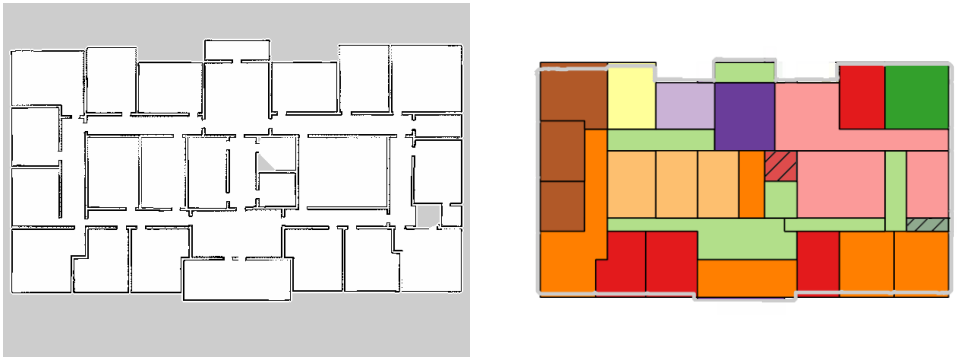


Figure 6.30: The figure shows the starting point of the exploration (in red).

Evaluation As it can be seen from Figure 6.32, our algorithm performs strictly worse for most of the exploration process, however when the the exploration reaches around the 96% of explored area, the behaviour changes and our algorithm is able to reach the 100% of explored area sooner with respect to the other algorithm, with a speed up of the 7.8%. The *ES* is triggered before the end of the exploration, leading to a speed up with respect to the *without* $\mathcal{L}$ algorithm of the 13.3%. The reconstructed layout, when the *ES* is triggered, is shown in Figure 6.31.



(a) (b)

Figure 6.31: Figure 6.31a shows the image of the map of Environment 9 when the early stop is triggered. Figure 6.31b shows the reconstructed layout when the early stop is triggered.

If we look at Figure 6.33, we can see that the travelled distance at the end of the exploration for both the exploration algorithm is almost equal. If

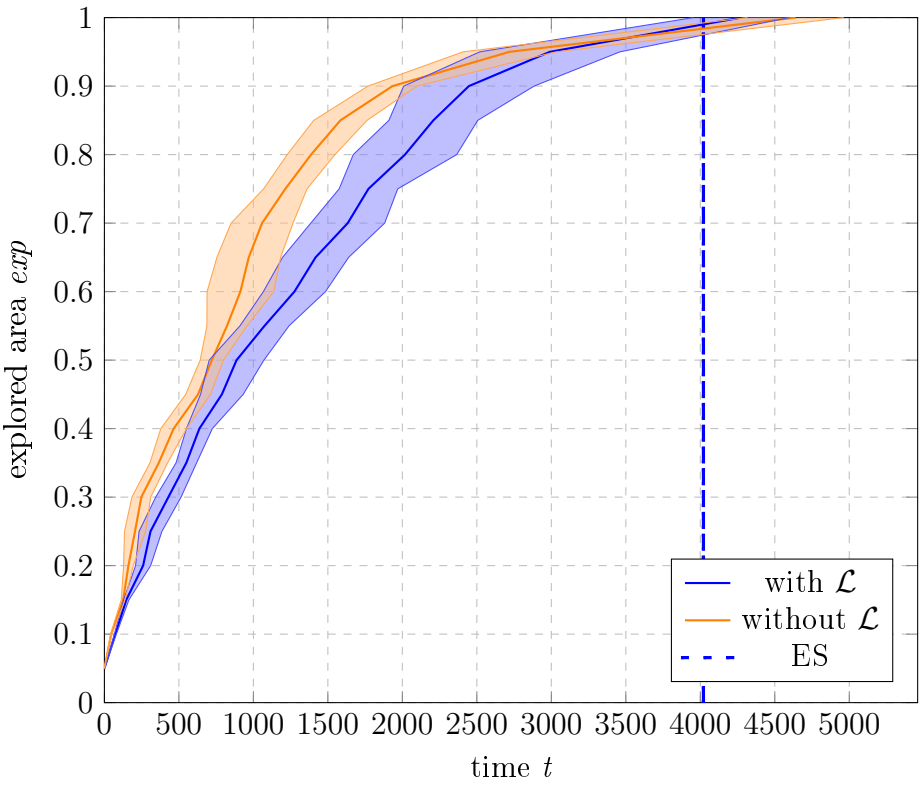


Figure 6.32: The explored area of the Environment 9 as a function of time.

we consider the *ES*, there is an enhancement in terms of travelled distance equal to the 5.2%.

Finally, in Table 6.10 the failure goal ratio for both the algorithms is reported. By looking at it, we can see that our algorithm is more robust respect to the *without* $\mathcal{L}$ algorithm.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
17.2 %	3.3	37.1 %	9.9

Table 6.10: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 9.

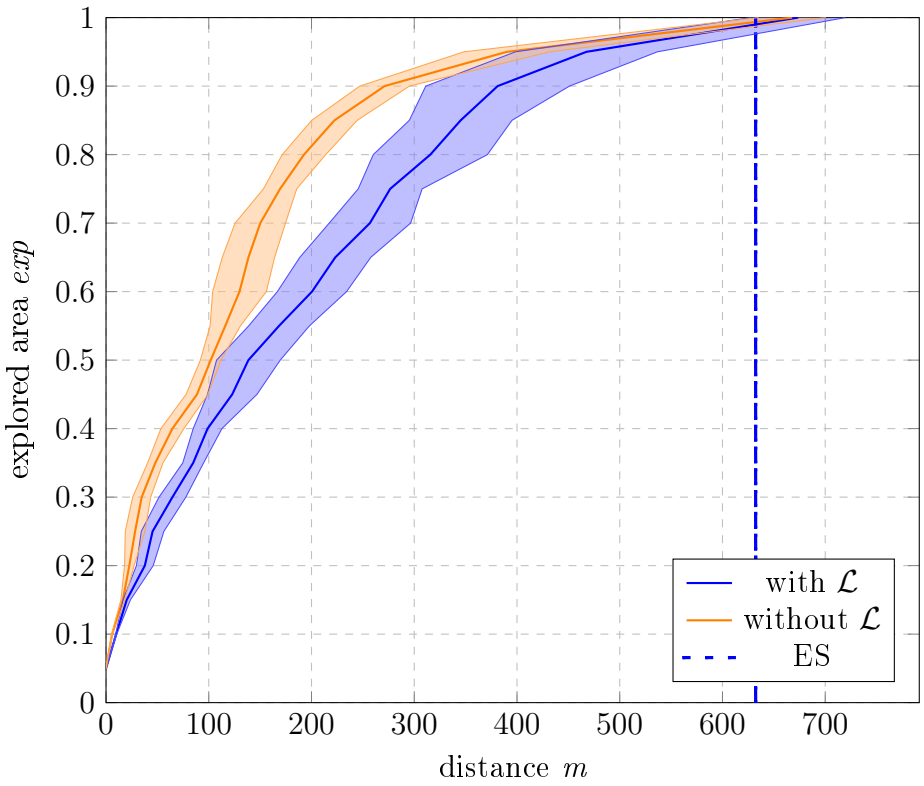


Figure 6.33: The explored area of the Environment 9 as a function of distance.

Environment 10

This map represents a school environment with a single corridor and many rooms attached to it. This environment has been categorized as hard due to the particular rooms that can be found in it, making the prediction pretty challenging. Figure 6.34 shows the starting point of the exploration.

Evaluation As we can see from Figure 6.35, which represents the explored area in function of time, the *with* $\mathcal{L}$ strategy performs strictly better than the *without* $\mathcal{L}$ strategy, and is able to reach the 100% of the explored area with a speed up of the 9.5%. The *ES* is triggered at the end of the exploration process, for this reason it does not provide any improvement in terms of speed up.

Further, the Figure 6.36 shows that our algorithm travels a shorter path

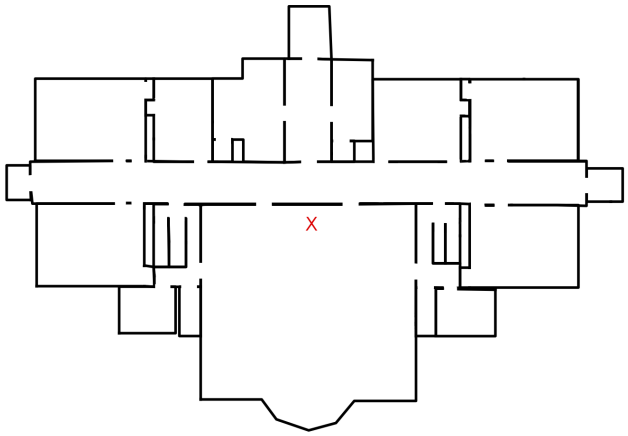


Figure 6.34: The figure shows the starting point of the exploration (in red).

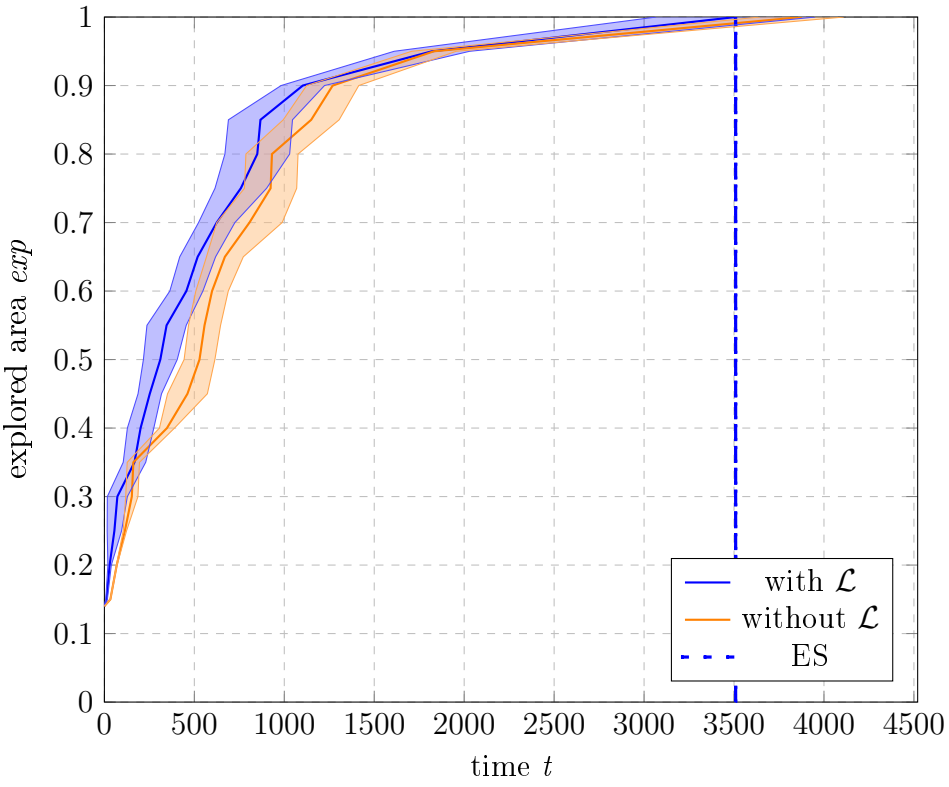


Figure 6.35: The explored area of the Environment 10 as a function of time.

to reach to 100% of the explored area.

Finally, in Table 6.11 it is reported the goal failure ratio for both the

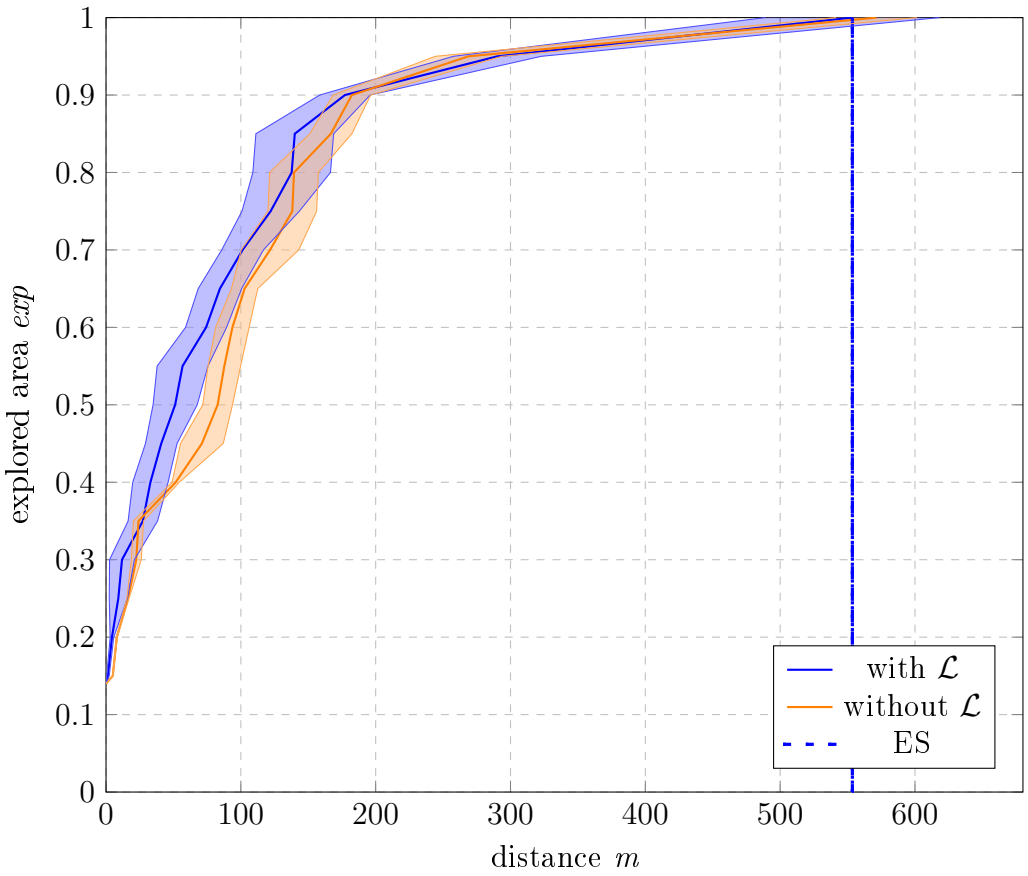


Figure 6.36: The explored area of the Environment 10 as a function of distance.

algorithms. It shows that *with* $\mathcal{L}$ is more robust compared to *without* $\mathcal{L}$.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Ratio	Std. Dev	Ratio	Std. Dev.
24.2 %	11.4	38.4 %	16.7

Table 6.11: Goal failure ratio for the *with* $\mathcal{L}$ and *without* $\mathcal{L}$ strategies in Environment 10.

General results

In Table 6.12 a recap of the general results previously described are displayed.

Env.	*without* $\mathcal{L}$			*with* $\mathcal{L}$			Gain		*ES*		Gain	
	Time	Distance	G. f. r.	Time	Distance	G. f. r.	Time	Distance	Time	Distance	Time	Distance
Env. 1	2119 s	295 m	27.1 %	1715 s	277 m	7.8 %	19.1 %	6.1 %	1473 s	238 m	30.5 %	19.3 %
Env. 2	1732 s	251 m	36.5 %	1636 s	265 m	15.8 %	5.5 %	−5.5 %	1566 s	254 m	9.6 %	−1.1 %
Env. 3	4766 s	742 m	42.1 %	4197 s	679 m	11.0 %	11.9 %	8.5 %	4188 s	677 m	12.1 %	8.7 %
Env. 4	2904 s	394 m	26.3 %	2482 s	379 m	4.2 %	14.5 %	4.0 %	2481 s	378 m	14.5 %	4.0 %
Env. 5	2613 s	398 m	33.3 %	2447 s	412 m	14.8 %	6.4 %	−3.5 %	2438 s	410 m	6.7 %	−3.0 %
Env. 6	3024 s	450 m	37.6 %	2842 s	472 m	9.0 %	6.0 %	−4.9 %	2820 s	468 m	6.7 %	−4.0 %
Env. 7	4399 s	701 m	25.3 %	4265 s	704 m	9.2 %	3.0 %	−0.4 %	4265 s	704 m	3.0 %	−0.4 %
Env. 8	4322 s	674 m	41.4 %	3537 s	568 m	23.9 %	18.2 %	15.7 %	3267 s	522 m	24.4 %	22.6 %
Env. 9	4639 s	667 m	37.1 %	4275 s	673 m	17.2 %	7.8 %	−0.9 %	4021 s	632 m	13.3 %	5.2 %
Env. 10	3876 s	571 m	38.4 %	3508 s	553 m	24.2 %	9.5 %	3.1 %	3508 s	553 m	9.5 %	3.1 %
Average	3440 s	514 m	34.5 %	3090 s	498 m	13.7 %	10.1 %	2.2 %	3003 s	483 m	12.8 %	5.4 %

Table 6.12: A a recap of the results previously described. "G. f. r." is the abbreviation of Goal failure ratio. The results correspond to the explored area equal to 100%.

6.5 Discussion

An interesting, although intuitively expected, result of our experimental analysis is that using the *with* $\mathcal{L}$ algorithm produces a speed up in the exploration process. From the results described in Section 6.4, the fact that is remarkably evident is that the algorithm produces a speed up, which is not uniformly distributed over all the exploration process but concentrated at the end of the exploration process. At the start the exploration strategy performs similarly to the *without* $\mathcal{L}$ method until, approximately, the 90% of the total area has been explored. From that moment on, our work performs consistently better and lead to a speed up of the exploration.

In Figure 6.37 a recap of the time required to reach the 100% of the exploration is shown. The average final gain of the *with* $\mathcal{L}$ algorithm is shown in Table 6.13.

The results we obtained can be explained by the fact that the robot starts with a very little knowledge of the environment (map), making it very difficult for the algorithm to make correct prediction. With the exploration going on and incrementally building the map, however, the algorithm makes more and more precise predictions, that lead to a speed up at the end of the exploration. In conclusion, the inaccurate prediction made at the start of the exploration does not jeopardize the gain obtained at the end of the exploration. An example of an incrementally built map and the predicted layout is shown in Figure 6.38 and in Figure 6.39.

The results also show that the *with* $\mathcal{L}$ algorithm is more robust with respect to the *without* $\mathcal{L}$ algorithm. This feature is shown by the goal failure ratio, that measures the ratio between a the number of times the exploration selects a goal that is unreachable (typically a frontier that becomes a wall) and the total number of goals selected before reaching the complete exploration. Table 6.14 shows the average goal failure ratio on the 10 environments for both our method and the baseline method. From these data we can observe that our algorithm is more robust than the *without* $\mathcal{L}$ algorithm. This is particularly evident in the late phase of the exploration, where it become more and more important and difficult to select the correct frontier

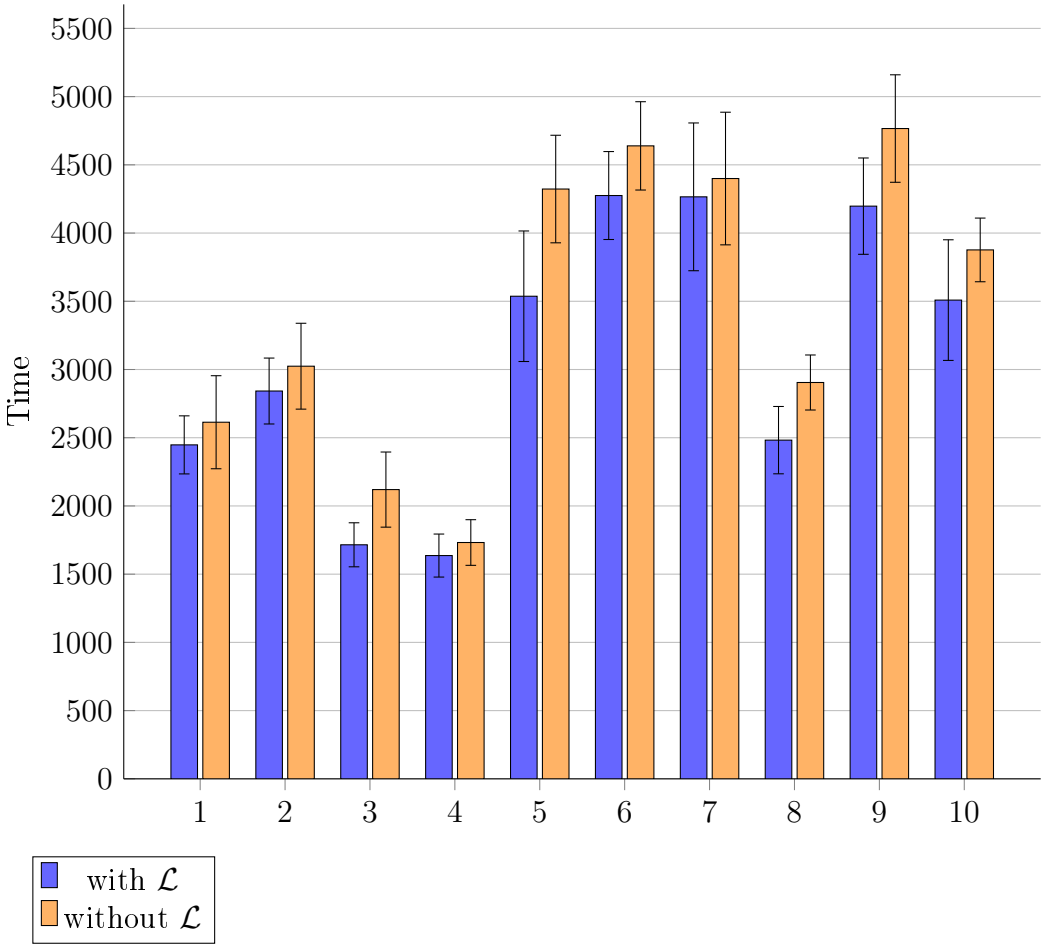
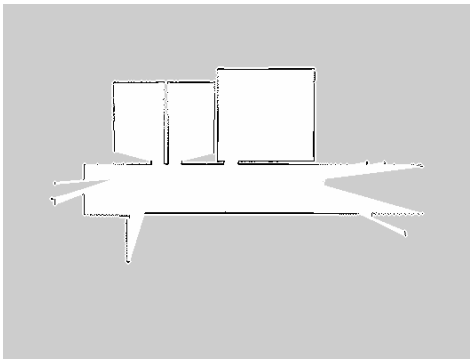
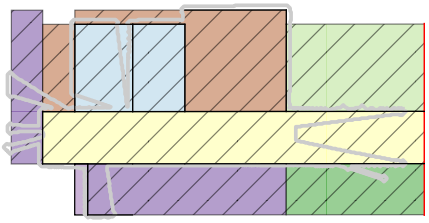


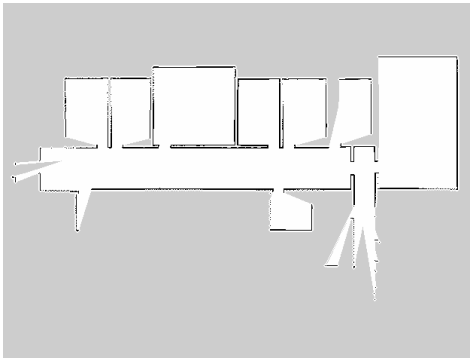
Figure 6.37: The figure shows the average time required in the 10 simulated environments to reach the 100% explored area.



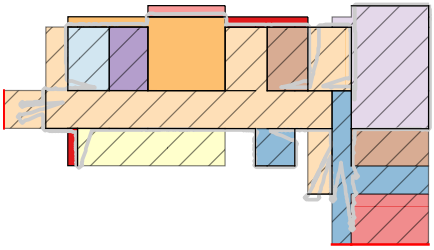
(a) Map at 20%



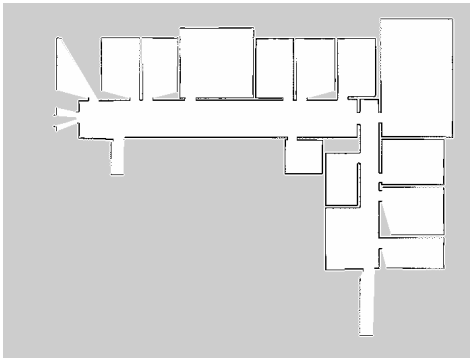
(b) Prediction at 20%



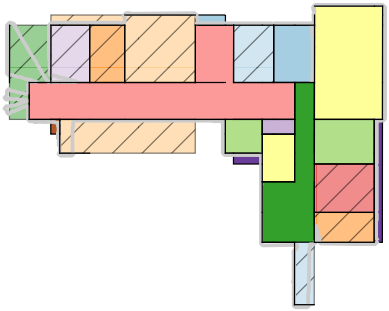
(c) Map at 40%



(d) Prediction at 40%



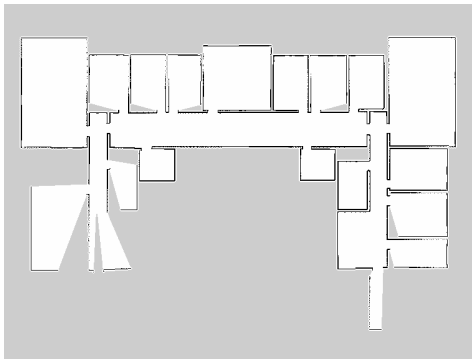
(e) Map at 60%



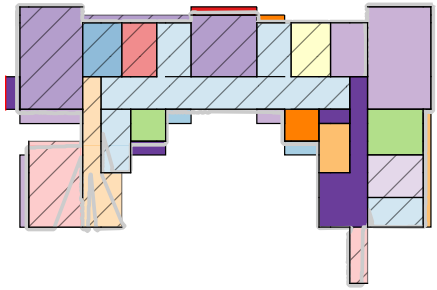
(f) Prediction at 60%

Figure 6.38: The figures show the map and the predicted environment with explored area equal to: 20, 40, 60 %. The complete map of the environment is shown in Figure 6.1h.

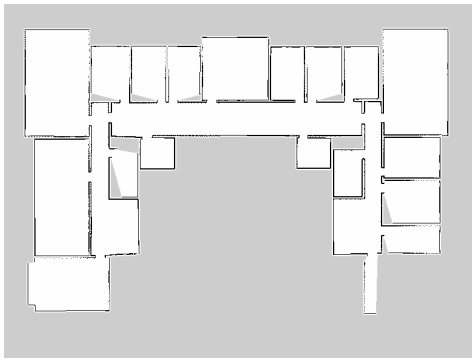
to explore.



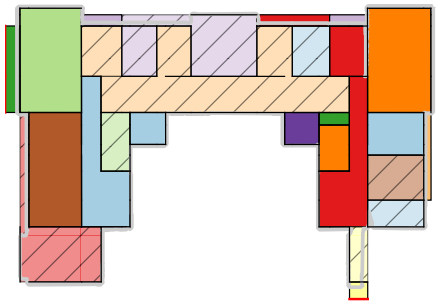
(a) Map at 80%



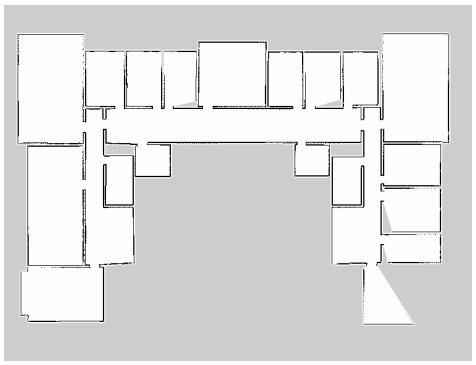
(b) Prediction at 80%



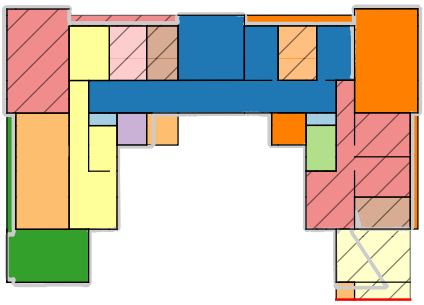
(c) Map at 90%



(d) Prediction at 90%



(e) Map at 95%



(f) Prediction at 95%

Figure 6.39: The figures show the map and the predicted environment with explored area equal to: 20, 40, 60, 80, 90, 95 %. The complete map of the environment is shown in Figure 6.1h.

6.5.1 Consideration on early stop

An important aspect of our work is the capability of exploration strategy to use the predicted layout of the environment to understand whether the

with $\mathcal{L}$		*without* $\mathcal{L}$		
average time [s]	Std. Dev.	average time [s]	Std. Dev.	gain
3090	316	3440	313	10.1 %

Table 6.13: The table shows the average gain of our method on the 10 different environments. "average time" is the average time required to perform the exploration on the 10 different enviroments and "Std. Dev." the corresponding standard deviation. These parameters has been calculated for both the exploration strategies used: *with* $\mathcal{L}$ and *without* $\mathcal{L}$. The gain is the average speed up of our algorithm.

with $\mathcal{L}$		*without* $\mathcal{L}$	
Average G. f. r.	Std. Dev.	G. f. r.	Std. Dev.
13.7 %	7.3	34.5 %	11.4

Table 6.14: The table shows the average goal failure ratio on the 10 environments. "G. f. r." is the abbreviation of Goal failure ratio.

exploration can be stopped since the map representing the environment can be reconstructed from the data collected so far. In our implementation we used a very conservative approach to trigger the *ES*, with the intention of not stopping the exploration of the environment when potentially interested frontiers can still be found. In our experiments, the *ES* has been successfully triggered in 4 maps with an effective gain in terms of speed up. In other 3 maps the *ES* has been triggered, however the speed up is negligible, so we do not count them as successful. From now we indicate as *with* $\mathcal{L}$ + *ES* the exploration strategy that uses the layout predicted combined with early stop. The results of the *with* $\mathcal{L}$ + *ES* are shown in Figure 6.40.

The average final gain of the *with* $\mathcal{L}$ is shown in Table 6.15.

As we can see the *with* $\mathcal{L}$ + *ES* algorithm leads to a small improvement with respect to the *with* $\mathcal{L}$ (from 10.1% to 12.8% as average speed up overall the environments). If we consider only the 4 maps in which the *ES* is successfully triggered, we obtain a gain in term of exploration time of 20.1%. However, if we manually observe the reconstructed map of the environment at different percentages of explored area, we can see that reconstructed envi-

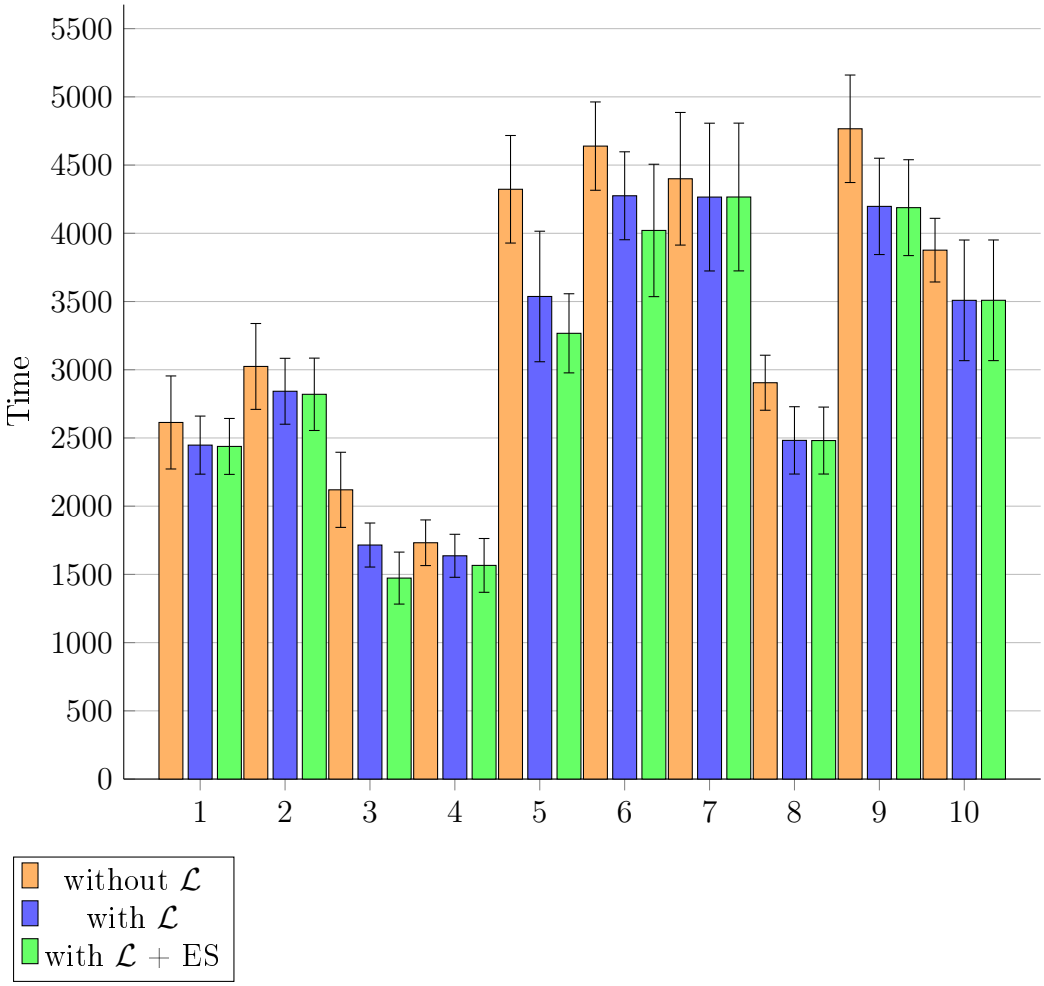
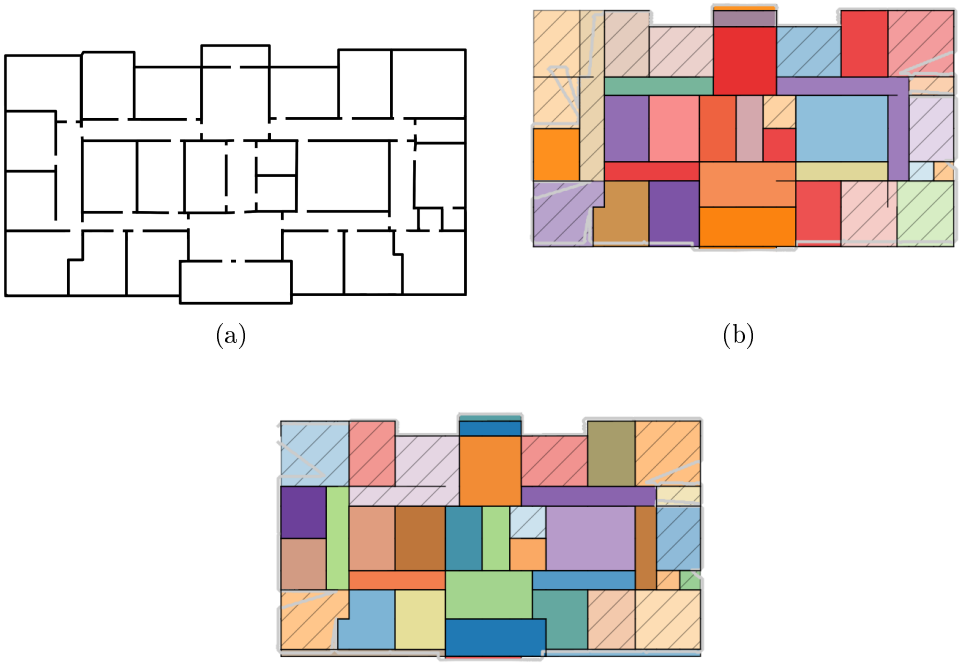


Figure 6.40: The figure shows the average time required in the 10 simulated environments to reach the 100% explored area for the *without* $\mathcal{L}$, *with* $\mathcal{L}$ and *with* $\mathcal{L}$ + *ES* algorithms. For the *with* $\mathcal{L}$ + *ES* we plot the time in which the *with* $\mathcal{L}$ reaches the 100% of the exploration or the instant in which the *ES* is triggered.

with $\mathcal{L}$ + *ES*		*without* $\mathcal{L}$		
average time [s]	Std. Dev.	average time [s]	Std. Dev.	gain
3003	321	3440	313	12.8 %

Table 6.15: The table shows the average gain of our method on the 10 different environments. "average time" is the average time required to perform the exploration on the 10 different enviroments and "Std. Dev." the corresponding standard deviation. These parameter has been calculated for both the exploration strategies used: *with* $\mathcal{L}$ + *ES* and *without* $\mathcal{L}$. The gain is the average speed up of our algorithm.

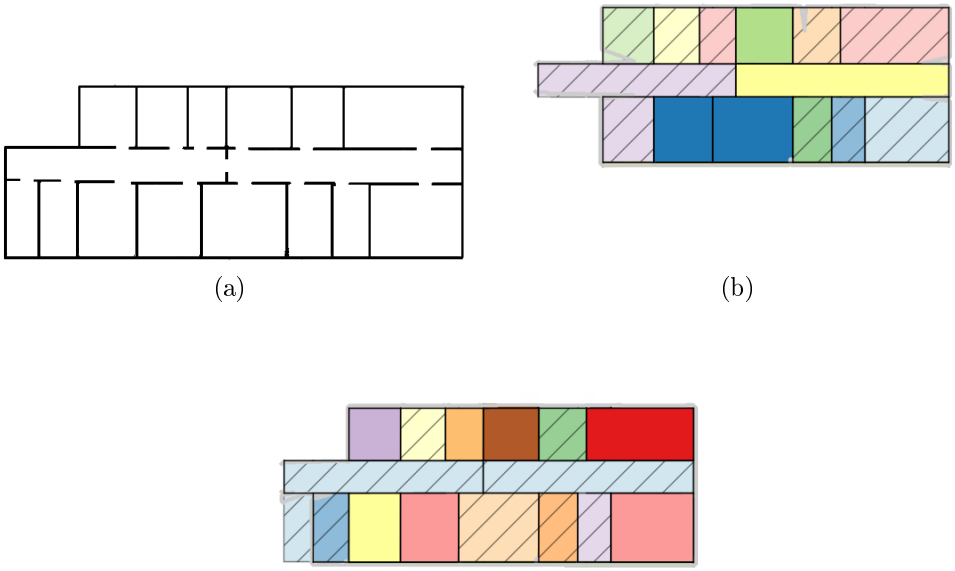
ronments start to have an high reliability when the exploration is around the $80\% - 90\%$ of the total area. If the exploration is manually stopped when the explored area is around the 95%, the layout $\mathcal{L}$ of the environment would be predicted with an high reliability, except for some minor inaccuracies. Then environment could have been considered explored with a speed up around the 38%. Instead, if we manually stop the exploration with the explored area equal to the 90%, we would typically lose a room (in most of the environments) but the speed up would be around the 50%. Two examples of predicted environments with the explored areas equal to 90% and 95% are shown in Figure 6.41 and Figure 6.42.



(a) (b)

(c)

Figure 6.41: Figure 6.41a shows the map of the environment. Figure 6.41b shows the reconstructed environment with the explored area at 90%. Figure 6.41c shows the reconstructed environment with the explored area at 95%.



(a) (b)

(c)

Figure 6.42: Figure 6.42a shows the map of the environment. Figure 6.42b shows the reconstructed environment with the explored area at 90%. Figure 6.42c shows the reconstructed environment with the explored area at 95%.

Chapter 7

Conclusions and future work

In this thesis we have studied and developed a multi-criteria exploration strategy for a mobile robot that explores an initially unknown indoor environment. The selection of the next location is made according to the expected information gain partially-observed rooms in terms of area visible from these locations calculated using a predicted layout of the unseen part of the environment.

We implemented our work in *ROS*, and tested it in 10 indoor environments simulated using *Stage*. The exploration strategy has been evaluated comparing against a baseline exploration strategy that evaluates the information gain as the maximum area visible from a location.

The results showed that a robot using our exploration strategy and one using the baseline method have similar behaviours at the beginning of the exploration process. However, at the end of the exploration process, the robot that uses our proposed method is able to make very accurate predictions of the environment, thus allowing to significantly reduce the time required to complete the exploration.

Moreover, we proposed a novel early stop (*ES*) criterion that uses the predicted layout to stop the exploration when only uninteresting areas are left. This criterion has been successfully triggered in 4 out of the 10 environments tested in the experimental evaluation, further decreasing the time required to complete the exploration.

Future work could address different aspects of our exploration system. Firstly, a more complex prediction about the unexplored part of the environment can be made, in order to create a layout closer to the real one also in the early phases of the exploration. This, for example, can be performed by keeping track of all the poses of the robot during the exploration, by knowing the sensors perception model, and by making a more informative prediction using these new data.

Another possible development could address the integration of our algorithm with a database of previously explored environments, allowing to make more informed predictions about the environment.

Moreover, it could be interesting to implement more aggressive criteria to trigger the early stop, following the reasoning described in Section 6.5.1.

A further interesting future direction includes the development of an exploration strategy that is able to dynamically switch between classical exploration strategies and our method when the prediction becomes more accurate.

Bibliography

[1] F. Amigoni, D. Fusi, and M. Luperto. "Exploiting Inaccurate A Priori Knowledge in Robot Exploration (extended abstract)". *Proc. AAMAS*. to appear. 2019.

[2] F. Amigoni and A. Gallo. "A Multi-Objective Exploration Strategy for Mobile Robots". *Proc. ICRA*. 2005, pp. 3861–3866.

[3] A. Aydemir, P. Jensfelt, and J. Folkesson. "What can we learn from 38,000 rooms? Reasoning about unexplored space in indoor environments". *Proc. IROS*. 2012, pp. 4675–4682.

[4] N. Basilico and F. Amigoni. "Exploration strategies based on Multi-Criteria Decision Making for search and rescue autonomous robots". *Proc. AAMAS*. 2011, pp. 401–417.

[5] J. Caley, N. Lawrance, and G. Hollinger. "Deep learning of structured environments for robot search". *Proc. IROS*. 2016, pp. 3987–3992.

[6] J. Canny. "A Computational Approach to Edge Detection". *IEEE T PAMI* 8.6 (1986), pp. 679–698.

[7] J. Chang, G. Lee, Y.-H. Lu, and C. Hu. "P-SLAM: Simultaneous localization and mapping with environmental-structure prediction". *IEEE T Robot* 23.2 (2007), pp. 281–293.

[8] T.J. Chong, X.J. Tang, C.H. Leng, M. Yogeswaran, O.E. Ng, and Y.Z. Chong. "Sensor Technologies and Simultaneous Localization and Mapping (SLAM)". *IEEE IRIS* 76 (2015), pp. 174–179.

[9] D. Comaniciu and P. Meer. "Mean shift: a robust approach toward feature space analysis". *IEEE T PAMI* 24.5 (2002), pp. 603–619.

[10] 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". *Proc. KDD*. 1996, pp. 226–231.

[11] B. P. Gerkey, R. T. Vaughan, and A. Howard. "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems". *Proc. ICAR*. 2003, pp. 317–323.

[12] *Gmapping*. `http://wiki.ros.org/gmapping`.

[13] H. Gonzáles-Baños and J.-C. Latombe. "Navigation Strategies for Exploring Indoor Environments". *SAGE IJJR* 21.10-11 (2002), pp. 829–848.

[14] G. Grisetti, C. Stachniss, and W. Burgard. "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters". *IEEE T Robot* 23 (2007), pp. 34–46.

[15] K. L. Ho and P. Newman. "Loop closure detection in SLAM by combining visual and spatial appearance". *Rob Auton Syst* 54.9 (2006), pp. 740–749.

[16] N. Kiryati, Y. Eldar, and A.M. Bruckstein. "A probabilistic Hough transform". *Pattern Recognition* 24.4 (1991), pp. 303–316.

[17] B. Kuipers and Y. Byun. "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations". *Rob Auton Syst* 8.1 (1991), pp. 47–63.

[18] A. Quattrini Li, R. Cipolleschi, M. Giusto, and F. Amigoni. "A semantically-informed multirobot system for exploration of relevant areas in search and rescue settings". *Auton Robots* 40.4 (2016), pp. 581–597.

[19] M. Luperto and F. Amigoni. "Extracting Structure of Buildings using Layout Reconstruction". *Proc. IAS-15*. 2018.

[20] M. Luperto, V. Arcerito, and F. Amigoni. *Proc. ICRA*. to appear.

[21] M. Luperto, A. Quattrini Li, and F. Amigoni. "A System for Building Semantic Maps of Indoor Environments Exploiting the Concept of Building Typology". *Proc. RoboCup*. 2013, pp. 504–515.

[22] *Move_ base*. `http://wiki.ros.org/move_base`.

[23] *Nav2d*. `http://wiki.ros.org/nav2d`.

[24] *OpenCv: Contour*. `https://docs.opencv.org/3.3.0/d3/d05/tutorial_py_table_of_contents_contours`.

[25] A. Pronobis. "Semantic Mapping with Mobile Robots". PhD thesis. Stockholm, Sweden: KTH Royal Institute of Technology, 2011.

[26] A. Pronobis and P. Jensfelt. "Large-scale semantic mapping and reasoning with heterogeneous modalities". *Proc. ICRA*. 2012, pp. 3515–3522.

[27] 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". *ICRA Workshop on Open Source Software*. 2009.

[28] *ROS: Robot Operating System*. `http://www.ros.org/`.

[29] C. Stachniss and W. Burgard. "Exploring Unknown Environments with Mobile Robots Using Coverage Maps". *Proc. IJCAI*. 2003, pp. 1127–1132.

[30] D. P. Ström, I. Bogoslavskyi, and C. Stachniss. "Robust exploration and homing for autonomous robots". *Rob Auton Syst* 90 (2017), pp. 125–135.

[31] D. P. Ström, F. Nenci, and C. Stachniss. "Predictive exploration considering previously mapped environments". *Proc. ICRA*. 2015, pp. 2761–2766.

[32] S. Thrun. "Robotic Mapping: A Survey". *Exploring Artificial Intelligence in the New Millenium*. Ed. by G. Lakemeyer and B. Nebel. Morgan Kaufmann, 2003, pp. 1–35.

[33] S. Thrun and A. Bü. "Integrating Grid-based and Topological Maps for Mobile Robot Navigation". *Proc. AAAI*. 1996, pp. 944–950.

[34] C. Tovey and S. Koenig. "Improved Analysis of Greedy Mapping". *Proc. IROS*. 2003, pp. 3251–3257.

[35] B. Yamauchi. "A Frontier-Based Approach for Autonomous Exploration". *Proc. CIRA*. 1997, pp. 146–151.