POLITECNICO DI MILANO
Corso di Laurea Specialistica in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

DEVELOPMENT OF A FRAMEWORK FOR EVALUATING PERFORMANCE OF EXPLORATION STRATEGIES OF AUTONOMOUS ROBOTS

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Advisor: Prof. Francesco Amigoni
Co-Advisor: Prof. Marina Indri
Tutor: Dott. Nicola Basilico

M.Sc. Dissertation of:
Alberto Quattrini Li, Student ID 749858

Academic Year 2010-2011

Ai miei nonni Lilia ed Enzo ...

Ringraziamenti

Desidero innanzitutto ringraziare il professor Francesco Amigoni, che con la sua costante supervisione ha saputo accompagnarmi nella realizzazione di questa tesi, ma che ha saputo anche darmi sostegno morale e umano. Spero di poter imparare ancora molto da lui.
Un caloroso ringraziamento va anche al dottor Nicola Basilico, che ha saputo darmi preziosi consigli quando ho incontrato difficoltà durante questo lavoro.

Un sentito ringraziamento va anche al professor Piero Fraternali, perchè ha saputo sostenermi nel progetto dell'Alta Scuola Politecnica ed ha contribuito a rendere migliore la mia preparazione grazie a una profonda multidisciplinarietà.

Ringrazio anche l'ingegner Roberto Tedesco, perchè lavorare al suo fianco mi ha permesso di crescere sia professionalmente che umanamente.

Ringrazio a tutta la mia famiglia che ha continuato a supportarmi in tutti i modi possibili durante questi anni di studio lontano da casa.

Rivolgo un grazie particolare a a Claudia che ha saputo sia aiutarmi nella revisione finale di questa tesi, sia incoraggiarmi e sostenermi durante i momenti difficili di questo intenso percorso.

Ringrazio i miei compagni di corso e amici, in particolare Andrea e Iacopo, per tutti i momenti passati insieme su progetti e non.

Un sentito ringraziamento va anche ai miei ex compagni di corso dell'università di Pisa, Andrea, Alessandro e Francesco con i quali ho continuato ad avere una forte amicizia nonostante la lontananza.

Infine, ringrazio tutti i miei più cari amici, per essermi stati vicino in questi due anni.

Abstract

Despite the fact that engineering fields usually have assessed standards for evaluating system performance, in the exploration problem, i.e., in the problem of mapping an initially unknown environment with autonomous mobile robots, there is not yet any standard evaluation methodology for comparing different exploration strategies. The lack of such a standard evaluation framework is probably due to the youth of the approaches to this problem.

One of the elements of a standard evaluation framework should be the comparison of the performance of an exploration strategy with the optimal performance attainable in a given environment. However, the problem of calculating the optimal exploration path in a given environment has not yet been adequately addressed in literature. Although several fields (e.g., robotics, graph theory, computational intelligence) dealt with this issue, results allow to calculate the upper or lower bound of the performance (e.g., number of perceptions, distance traveled) of a given strategy, for generic classes of environments. These results do not answer to the question of what is the best performance in a given, specific, environment. The answer to this question has relevance in practically applying exploration strategies.

In this thesis, we contribute to build up a framework that allows to compare strategies performance with the optimal exploration path in a given environment. The innovative aspect of this work is that, given an environment, the comparison of different exploration strategies can be performed with respect to the optimal behavior (which is computed by our framework), instead of considering lower or upper bounds. The approach to compute the optimal exploration is based on the use of off-line seach algorithms, such as A* or branch and bound. The quality of the optimal solution greatly depends on initial parameters and assumptions we make (e.g., objective test, representation of the map, perception and locomotion models of the robot). Experimental results obtained in simulation in some realistic environments have shown the validity of our approach to solve the problem of determining the optimal solution for the exploration problem in a specific environment.

From the scientific point of view, the contribution of our work is twofold: on the one hand, it contributes to the definition of a standard evaluation framework for exploration strategies, allowing to compare exploration strategies not only between each other in a relative way (which is the current approach), but also against the optimal exploration in an absolute way. On the other hand, our framework could be useful for developing better exploration strategies.

Sommario

Nonostante nell'ingegneria ci siano solitamente standard condivisi per valutare le performance di un sistema, per il problema dell'esplorazione, cioè per il problema di mappare un ambiente inizialmente sconosciuto con robot mobili autonomi, non è ancora stata definita una metodologia standard di valutazione per confrontare diverse strategie di esplorazione. La mancanza di una metodologia generale di valutazione è dovuta, molto probabilmente, al fatto che gli approcci per risolvere questo problema sono recenti.

Uno degli elementi di un framework di valutazione standard dovrebbe essere il confronto delle performance di una strategia di esplorazione rispetto alla performance ottima ottenibile in un dato ambiente. In letteratura, il problema del calcolo del percorso ottimo per esplorare un dato ambiente non è stato ancora studiato in modo approfondito. Anche se molte discipline scientifiche, come la robotica, la teoria dei grafi, l'intelligenza computazionale, hanno trattato questo problema, i risultati ottenuti consentono di calcolare il limite superiore o inferiore delle performance ottenibili con una data strategia di esplorazione, per classi di ambienti generici. Questi risultati non permettono però di capire quale sia la migliore performance in un dato ambiente. Questa informazione è, d'altra parte, rilevante per le applicazioni pratiche.

In questa tesi, contribuiamo a sviluppare un framework che permetta di confrontare le performance delle strategie di esplorazione rispetto al percorso ottimo in un dato ambiente. L'aspetto originale di questa ricerca è che le varie strategie esplorative possono essere confrontate rispetto all'ottimo calcolato dal nostro framework, e non più, quindi, considerando solo il limite superiore o inferiore. L'approccio per determinare l'esplorazione ottima si basa sull'uso di algoritmi di ricerca off-line, come A* e branch and bound. La qualità della soluzione ottima dipende fortemente dai parametri iniziali e dalle ipotesi fatte (ad esempio, il test obiettivo, la rappresentazione della mappa, i modelli di percezione e movimento del robot).
I risultati ottenuti in esperimenti simulati condotti su ambienti realisti-

ci dimostrano che la validità dell'approccio che sta alla base del nostro framework.

Dal punto di vista scientifico, il contributo di questo lavoro è duplice: da un lato, contribuisce alla definizione di un framework standard per la valutazione delle strategie di esplorazione, permettendo di confrontare strategie di esplorazione rispetto all'ottimo e non più soltanto in modo relativo; dall'altro questo lavoro è interessante per studiare l'uso del framework sviluppato in questo lavoro per definire migliori strategie di esplorazione.

Contents

Ringraziamenti **I**

Abstract **III**

Sommario **V**

1 Introduction **1**

2 State of the art **5**
2.1 Exploration strategies 6
2.2 System performance evaluation 7
2.3 State of the art in exploration problem 9

3 Optimal exploration as a search problem **17**
3.1 Preliminary definitions: search problems 17
3.2 Preliminary hypotheses 22
3.3 Problem formulation in exploration case 23
3.3.1 General formulation 23
3.3.2 Parameters and constraints of the search problem . . . 25
3.4 Examples 32
3.4.1 Formulation 32
3.4.2 Heuristics 33

4 Implementation **35**
4.1 Description of the framework 35
4.2 Core code 36
4.3 Relaxation of the problem 41
4.3.1 Initial parameters 42
4.3.2 Constraints 45

5 Experimental results **49**
5.1 Tests methodology and experimental setting 49
5.2 Analyses and comparisons 53
5.3 Solution quality vs computational time 56
5.3.1 Initial parameters 57
5.3.2 Constraints 66
5.3.3 Further experiments 74
5.4 Other experiments 80
5.4.1 Openspace environment 80
5.4.2 Obstacles environment 84

6 Conclusions **89**

Bibliography **91**

Chapter 1

Introduction

Autonomous mobile robotics has seen a widespread development in recent years, due to applications (surveillance, rescue, etc.) where robots are required to operate without any human supervision, especially in the cases where human beings cannot access environment because of their asperity. There are several challenges that a designer faces during the development of an autonomous robot, from low level issues, i.e., sensors, actuators, etc., to high level issues, i.e., control [22].
One of the most important aspects, that affects autonomous mobile robots performance, is the navigation strategy, which is the set of techniques that allow an autonomous mobile robot to decide the next location to reach, by using its current knowledge.

An important task, which is involved in navigation strategies, is the exploration problem. Exploration problem has been studied in connection to autonomous mobile robots able to explore unknown environment and build a map of it. There are several applications that benefit from this task, e.g., planetary exploration [18] and search and rescue [17].

During last years, there have been several proposals about *exploration strategies* [4, 8, 3]. Exploration strategies determine the path followed by the robot to explore an unknown environment, possibly optimizing it, according to some metrics. However, no standard evaluation framework is available yet for comparing different exploration strategies [2], probably because this field started to be studied only recently. In such a situation, there is no guarantee that an exploration strategy is the optimal one. Exploration strategies are usually tested in given environments and their performance is compared to each other. Hence, their evaluation is relative.

The aim of the work is to contribute to build up a framework for the absolute evaluation of exploration strategies with respect to optimal explo-

ration paths.

We deal with the problem of calculating the optimal exploration path given an environment. With the assumption of the a priori knowledge of an environment, our results originally sets a bound on the performance of exploration methods. In this way, we can compare exploration strategies with the optimal behavior. Nowadays, we just have the upper or lower bounds on the performance (number of perceptions, distance traveled, etc.) of a given strategy in a generic class of environments. All comparisons among different exploration strategies are performed in a relative way, given a specific map [2].

The innovative aspect of this work is that, given an environment, the comparison of different strategies is performed with respect to the optimal behavior (which is computed by our framework). The optimal exploration is calculated by applying search algorithms, such as A* and branch and bound. The quality of the optimal solution greatly depends on assumptions and constraints we impose (e.g., perception and locomotion models of the robot, representation of the map).
Our approach to find out an optimal exploration path is novel, because we change the usual representation of the state in exploration problems studied in computational geometry and we use off-line search algorithms to compute the path to optimally explore the environment.

From the scientific point of view, our work is significant because it contributes to the scientific endeavor aiming at the definition of a general methodology for evaluating and comparing exploration strategies in a standard way. Our framework, moreover, could be a starting point for developing improved exploration strategies, because it can provide some hints for defining better heuristics or methods for choosing the next point to reach.

We performed experimental activities for testing the framework both in terms of computational time and in terms of quality of the solution (considering number of perceptions and traveled distance), in some classes of real environments, and compared those results with results obtained with some real exploration strategies, available in [2]. Furthermore, we showed how the quality of the solution and the computational time change, according to initial conditions and parameters provided to the framework.

This thesis is structured as follows. In Chapter 2, we introduce the current state of the art about exploration strategies and particularly about their evaluation, besides a fundamental background about exploration problem, necessary to develop the framework of this thesis. In Chapter 3, we formally

define the problem as an off-line search problem, reporting some preliminary hypotheses, in order to simplify the problem. In Chapter 4, we describe the implementation of the framework reporting relevant pseudo-code. In Chapter 5, we report our experimental results, by comparing on-line exploration strategies and performances in terms of quality of solutions and computational time. In Chapter 6, we sum up our work and suggest further directions of work.

Chapter 2

State of the art

Exploration is a task that plays a fundamental role in many applicative contexts, which span different domains, from planetary exploration, to search and rescue missions. In this thesis, we considered the case in which a mobile robot has to build the map of a surrounding environment, the so called exploration problem, in an initially unknown environment, by moving around according to some exploration strategies and perceiving surroundings through suitable sensors. In literature, this task is accomplished even by employing several robots (e.g., [7]) that should cooperate in order to build the entire map. Here, we considered just a single robot.

Studies on exploration are relatively recent and not stable yet. This fact is confirmed by experimental activities, that can be found in literature and are carried out for evaluating exploration strategies: now just relative comparisons are performed. In such a situation, designers of exploration strategies are not aware of the performance of an exploration strategy with respect to the optimum, and hence they do not know margins of possible improvements. Furthermore, they have to test all possible exploration strategies, in order to define which one performs better.
The desideratum is to have a general evaluation methodology for comparing different exploration strategies with respect to an optimum, and thus an absolute comparison can be made. Therefore, a framework to solve the problem of determining the optimal exploration path in a specific environment is necessary.

In this chapter, we report the current state of the art in this field, for understanding the current limitations in evaluating exploration strategies and how this work contributed to cover this gap. In addition, some theory necessary for developing the framework is recalled.
Specifically, in Section 2.1, we briefly introduce what is an exploration strat-

egy and why we need a new approach to face the problem of evaluating a strategy. In Section 2.2, we present current work in terms of evaluation of exploration strategies and why they are not sufficient for a general evaluation methodology. In Section 2.3, we describe current state of the art about robot details in exploration problems, because this is the background necessary for the development of this framework.

2.1 Exploration strategies

Apart from telecontrolled robots, most of the research focused on developing exploration strategies, that are deployed on robots and make them able to autonomously move around an initially unknown environment, in order to optimally discover it. Several exploration strategies have been developed during these years, from the simplest ones – i.e., those with a predefined trajectory [16] – to the most complex ones – i.e., those which have to choose the next point on the basis of some criteria [30, 28]. On autonomous mobile robots, on-line algorithms are usually deployed, i.e., the solution is computed while the robot is operating in the physical world. Solutions are found by searching a state space, that is the set of all possible states that can be reached by the robot. In exploration problem, states are unknown to the agent, specifically successor states are not known until the action is performed by the robot [20].

The most common approach is to give a destination point, and the robot can choose how to reach that point, without knowing what there is between its current position and the destination point. In [13], it is presented Learning Real-Time A* (LRTA*[1]), applied to the problem of getting to a given destination in an initially unknown environment. The state is represented by the point where the robot finds itself. Indeed, as shown in Figure 2.1, states are represented as vertices visited or not.

[1]LRTA* is a popular control method that interleaves planning and plan execution and usually used to solve search problems in known environments efficiently.

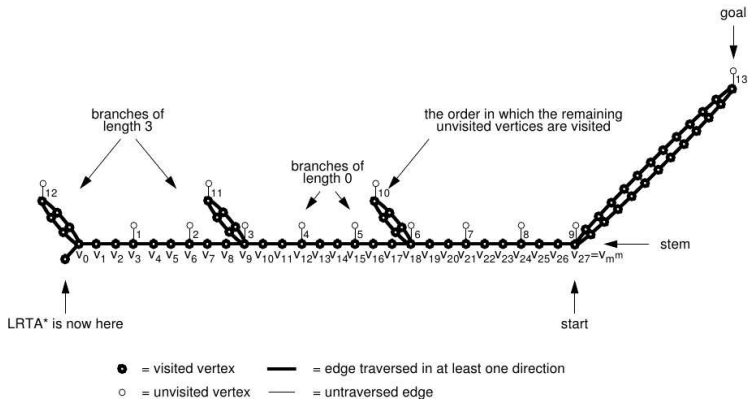


Figure 2.1: An undirected graph used by LRTA* for computation [13].

This approach, in which a state represents the point where the robot finds itself, is not significant in our context, since we are focusing on the exploration problem, which has the goal of building the map of an environment and which does not have a natural destination point. In exploration, states correspond on how much percentage of the map is discovered, and a destination point is not known a priori. The state should represent the partial knowledge of the map, because we are dealing with the problem of mapping an environment. So, we need a new approach to deal with the problem of determining the optimal exploration path in a specific environment.

2.2 System performance evaluation

The evaluation of the performance of a system is an important and required step in engineering fields. For the exploration problem, the definition of a standard evaluation framework has not been addressed yet in literature. Indeed, there is not any standard evaluation methodology for comparing different exploration strategies [2] and different evaluation metrics are considered. For example, in [15], the quality of the map obtained by a strategy and the computational time to produce the map are taken into account to compare different exploration strategies. Other researches have focused on comparing each algorithm with respect to others, in terms of traveled distance or number of perception actions required to map the environment [1]. This relative comparisons could affect the development of exploration strategies, because researchers do not know when the exploration algorithm performs well or not and how much effort they have to spend on improving such strategies. Furthermore, it is not possible to check whether an exploration strategy is the best one, unless all methods are tested and then compared,

because there is not an absolute result to compare to.

In addition, although several fields (e.g., robotics, graph theory [5, 6], computational intelligence) dealt with optimality in exploration strategies, results allow to calculate the upper or lower bound of the performance (e.g., number of perceptions, distance traveled) of a given strategy, for generic classes of environments. These results do not answer to the question of what is the best performance in a given, specific, environment, because they are too generic and so the optimum in a specific environment can be very different from a computed bound. For example, [29] presents an improvement on greedy mapping, which is a simple mapping method for mobile robots, in terms of upper and lower bound on its worst-case traveled distance. The environments they used for testing this strategy are very unrealistic, since one is created randomly and another is a maze, as shown in Figure 2.2.

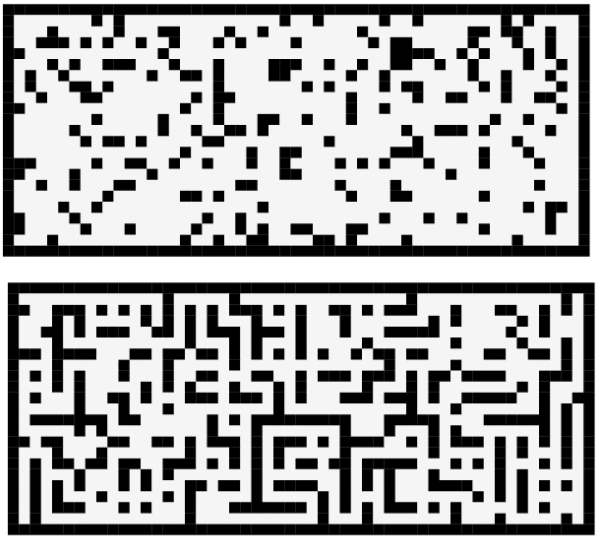


Figure 2.2: Environments adopted for testing greedy mapping [29].

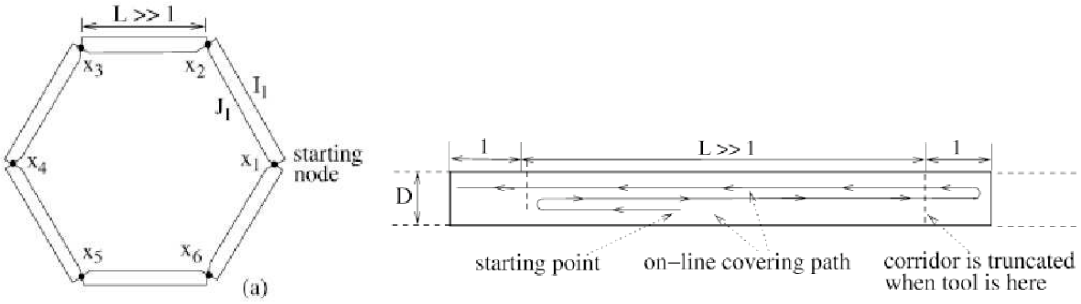
The result found by [29] is the new upper bound in the worst case:

$$|V| + 2\,|V| \ln |V| \tag{2.1}$$

where $|V|$ is the number of vertices of a graph $G = (V, E)$.

Another work [9] describes two on-line algorithms for covering planar areas, i.e., the mobile robot has to move along a path such that every point of the area is covered. This paper demonstrates that any on-line coverage algorithm has a fixed lower bound in any bounded planar environment.

Actually, this result is obtained in very simple environments, as depicted in Figure 2.3.



(a) A Double-ring environment (b) A Corridor environment

Figure 2.3: Environments considered in [9].

Even here, we have an upper bound of a covering path in the worst case:

$$(2 - \varepsilon)l_{opt} \tag{2.2}$$

where l_{opt} is the length of the shortest off-line covering path and ε is an arbitrarily small positive parameter.

Both works find and demonstrate that these on-line algorithms have a lower or an upper bound. This means that an algorithm cannot perform better or worse than the bounds found. With these results, it is not possible to compare different exploration strategies in a specific environment in an absolute way, since these findings are general and applicable to classes of environments.

So, as emerged from the analysis of the state of the art of the exploration strategies evaluation, an absolute comparison lacks. Specifically, there is no answer to what the best realization of an exploration in a given environment is.

The objective of this thesis is to contribute to the scientific endeavor of defining a standard evaluation framework, by building up a system that allows to compare different exploration strategies performance with respect to the optimal solution for exploring a specific environment, so that the limitations of current evaluation methodologies are overcome.

2.3 State of the art in exploration problem

Exploration problem is usually solved by performing steps represented in Figure 2.4.

$$perceiving \longrightarrow mapping \longrightarrow planning \longrightarrow executing$$

Figure 2.4: Exploration steps.

During the first phase, a robot perceives the surrounding thanks to its on-board perception system. There are several perception systems available, each of one with different features and characteristics.
Mobile robots can have several sensors [27]:

- contact sensors, that are able to detect impact with obstacles (e.g., bumpers);
- internal sensors, that measure some parameters of the robot, as acceleration, inclination, etc. (e.g., accelerometers, gyroscopes, compasses, inclinometers);
- proximity sensors, that detect obstacles when they are in the range of a sensor (e.g., sonar, radar, laser range-finders, infrared);
- visual sensors, which visualize objects when they are in the focal area (e.g., cameras);
- satellite-based sensors, that can position a robot in the space with respect to an absolute reference system (e.g., GPS).

The choice of a perception system at design time will affect exploration, because they have different ranges, different errors, etc., and thus a different capability of perceiving surroundings.

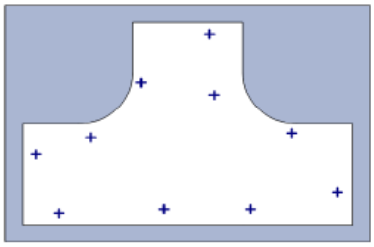
In simulation, there are three approaches to model perception systems [14]. The first one is a deterministic model, which simulates a perfect perception, i.e., there are no errors due to noise or sensor errors.
The second one is nondeterministic, in which noise and errors are taken into account, so the perception is random and could be not exact.
The last one is probabilistic, which finds a probability density function to model the perception. These approaches have different level of complexity, from the simplest one (i.e., deterministic) to the most difficult one (i.e., probabilistic). It is important to remark that a higher level of complexity corresponds a modelization closer to the real world. However, there is a trade-off between computational complexity and realism.

At the second step, the robot represents the environment perceived at the step before, by merging previous map with new information deriving from previous step. There are three different types of maps [11], that carries different types of information:

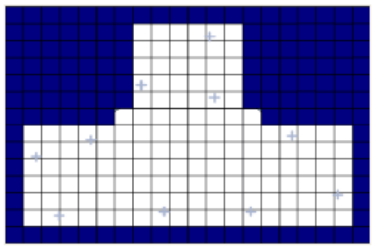
- free space map: only free space is represented;
- object oriented map: even obstacles are represented;
- composite map: a mix of previous two type of maps.

As we can see from Figure 2.5, different ways of representing maps mentioned above exist [26]:

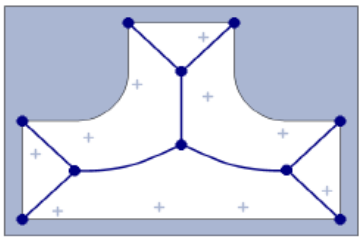
- feature-based map (see Figure 2.5a): objects are represented on the basis of their geometrical features with respect to an absolute reference;
- grid-based map (see Figure 2.5b): the environment is divided into cells, whose size can be chosen;
- topological map (see Figure 2.5c): just relevant points in the environment are represented.



(a) Feature-based map



(b) Grid-based map



(c) Topological map

Figure 2.5: Three types of map [19].

As we can see from Figure 2.6, each of these maps has different properties, which determine how to perform computations and operations on maps. Furthermore, they have different complexity and so different details of the world.

	Feature-based	Grid-based	Topological
Construction	Kalman filter	Occupancy grids	Navigation control laws
Complexity	Landmark covariance (N^3)	Grid size and resolution	Minimal complexity
Obstacles	Only structured obstacles	Discretized obstacles	Defined by the safest path
Localization	Arbitrary	Discretized	Nodes
Exploration	No inherent exploration	Frontier-based exploration	Graph exploration

Figure 2.6: Properties of each map type [19].

Focusing on grid maps, they could be implemented in different ways.
A way is to discretize environments with fixed size cells, each of which can have value of 0 (free space), or 1 (obstacle).
Another way is to have different grain levels, as we can see in Figure 2.7, where the grid map have cells with different size.

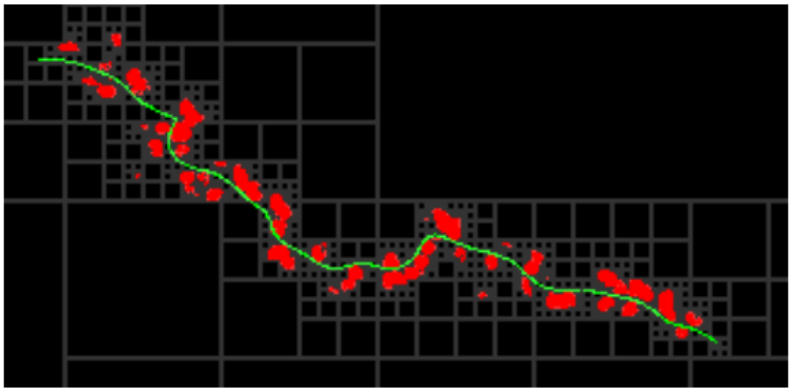


Figure 2.7: A map where quadtree is used [24].

The method to apply the quadtree is the following: at first the environment is discretized with a pre-defined cell size, then, depending on the value of each cell (that can be a real number between 0 and 1), the cell could be expanded, by using a quadtree. In this way, it is possible to have a fine-grained details level, where it is needed, and to have an improvement of efficiency in memory and computational resources [24].
It is also important to decide how cells are connected, in order to model the robot movement. There are three types of connection, as shown in Figure 2.8:

- 8-adjacency;
- orthogonal 4-adjacency;
- oblique 4-adjacency.

This decision affects exploration, because it limits possibility of robot movement, and therefore the performance and the solution of an exploration strategy.

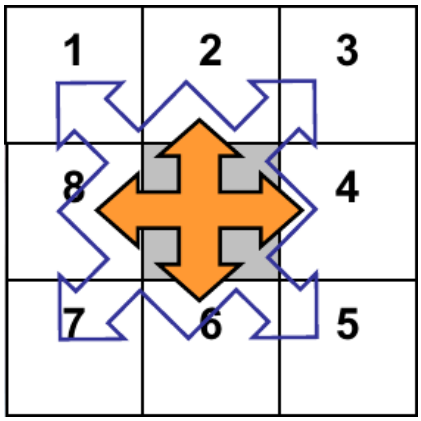


Figure 2.8: Different connections of cells in a grid map.

Another taxonomy of maps exists [27]. A map could be world-centric, i.e., coordinates of the map are represented in a global coordinate space and entities do not have any information about sensor measurements. Or a map could be robot-centric, i.e., a collection of sensor measurements at different locations, since everything is relative to the robot.
The choice of how to represent an environment determines algorithms and approaches that can be used to handle these maps.

At the third step, on the basis of the previous and new information acquired during perception phase, the robot localize itself in the new map, and computes and decides, according to an exploration strategy, the next action. About this step, for the sake of the thesis, it is not relevant to report algorithms used for integration and localization, such as scan matching [21], odometry techniques or Kalman filters [10], since, as we will see in Chapter 3, in the problem of searching an optimal exploration path, we suppose that the robot is able to integrate the map and localize itself in a deterministic way. Instead, it is important the decisional part, which is the core of an exploration strategy.
In a certain state, an exploration strategy has a set of successors, that are possible points reachable from the current state of the robot. This set is

called *frontier*. For example, an exploration strategy can consider as points in the frontier, those close to unknown part of the environment [19]. The definition of successors set distinguishes an exploration strategy and it is defined at design time. It is clear that the more successors a robot have, the more time it has to spend to compute the one to choose. In order to reduce the size of the frontier, a designer can define some constraints to discard some cells that could be inserted in the frontier. For instance, a strategy can discard points too close to obstacles for safety purpose. However, if too many constraints are imposed, it could happen that the robot does not find a solution, because frontier is an empty set. Hence, it exists a trade-off between constraints and computational time.
After having found a set of successors, an exploration strategy picks one successor according to some criteria and perform the action to reach the corresponding point. Below, we describes three simple exploration strategies examples, highlighting the criteria for choosing next successors.

One simple example is proposed by [25], in which the robot chooses points that are on a geometrical pattern, so that the robot can explore all points of interest in a map. This kind of approach defines a pre-computed trajectory, that a robot should follow. However, this could be not applicable in most of the cases, because the environment is unknown, and thus a designer cannot know what path to design. Furthermore, a robot could block itself, if it follows a pre-computed trajectory, due to unexpected obstacles. Indeed, they are usually used for representing landmarks for localization (i.e., visual maps) and not the environment geometry [23].

Another approach randomly picks cells from the frontier, without any particular logic (e.g., [8], which developed a randomized strategy for cooperative robot exploration). This method makes explorations non deterministic, since choice of next point is random.

Finally, there are the so called *Next Best View* strategies, which are the most complex ones, and make a mobile robot more autonomous.
This type of strategies defines an evaluation function, which is used for choosing one successor in the current frontier. For example, exploration strategy B-L [12] defines the following evaluation function:

$$h(q) = A(q) \exp(-\lambda L(q)) \tag{2.3}$$

where q is the successor point, λ is a positive constant, $L(q)$ is the length of the path from current robot position to q and $A(q)$ is an estimation of how much part of the map can be perceived from q. The intuition is to foster successor points where the robot can perceive more, but this value is weighted according to the length of the path.

Another example is the one proposed by [3], which uses different *utility* functions, that are different evaluation functions, and combine them with Multi-Criteria Decision Making (MCDM). In this case, different evaluation functions can be defined, e.g., the travelling cost from the current robot's position to a successor, the area-based information gain estimate, as the area of unknown space potentially visible by the robot if it goes to the successor and the segments-based information gain as the length of the frontier between mapped and unknown space the robot can perceive at a successor. Then, a weight is assigned to each criterium and these criteria are combined by using MCDM (e.g., weighted average can be computed).
It is clear that the best the evaluation function is, the best choice the robot can make.

Finally, the robot performs the action planned during the step before. Even during the action, there could be errors, due to asynchronicity of wheels, motors, etc., that could bring the robot in another point with respect to what it has planned. In addition, as depicted in Figure 2.9, the error tends to accumulate along the path and could become very distant from the actual plan, because every error is correlated to the previous ones.

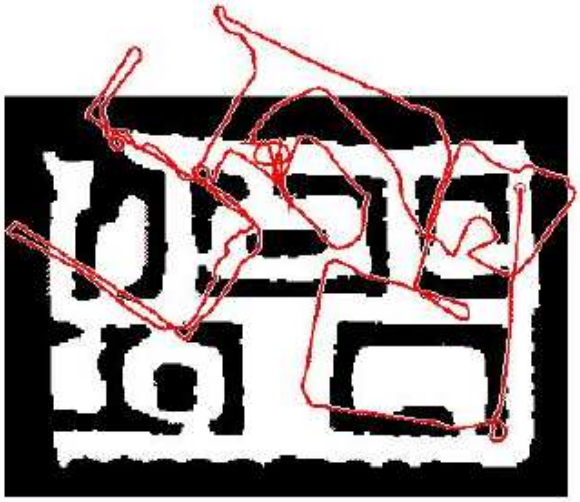


Figure 2.9: Errors accumulated along the path [27].

In simulation, it is possible to model the movement without errors or according to a classification as perception model, i.e., deterministic, non deterministic and probabilistic.

One element that can add complexity to a model is the dinamicity of the world. In that case, the knowledge of the robot could be different to the current state of the world, because it could change. Instead, the staticity of

the world means that it preserves same conditions over time.

All the elements described above should be taken into account when we design the behavior of a robot and its representation of the state. In the next chapter, we formulate the problem of determining an optimal exploration path in a specific environment and design its model, considering all the elements reported in this chapter.

Chapter 3

Optimal exploration as a search problem

As reported in Chapter 2, there is no standard methodology for evaluating and comparing different exploration strategies, apart from the lower and upper bounds provided for different on-line exploration algorithms.

The objective of this research is to develop an evaluation framework for comparing exploration strategies with respect to an optimum, given a specific environment. In this way, we contribute to the scientific endeavor of defining a standard evaluation framework, as available in other engineering fields.

In the next sections, we describe our approach and how we modeled the problem of determining an optimal exploration path in a specific environment. Specifically, in Section 3.1 we report the basics of a general search theory from artificial intelligence, that is useful for designing our model. In Section 3.2, we introduce assumptions we made in order to model the problem. In Section 3.3 we describe our general search problem formulation and discuss about relations between time complexity and parameters and constraints. Finally, in Section 3.4 some simple examples are reported in order to understand and verify the formulated search problem.

3.1 Preliminary definitions: search problems

Our approach to the problem of determining an optimal exploration path is to suppose that the environment is known and to formulate this problem as a *search problem*. It is worth to introduce some definitions and notations about search problems from artificial intelligence theory.

As [20] states, a search problem is defined by 6 elements:

- states s in which an agent could find itself;
- initial state s_0, from where an agent starts;
- actions that an agent can perform, when in a state s;
- transition function, which is a function that, given a state and an action, returns a new state:

$$T(s, a) = s' \tag{3.1}$$

- objective test, that determines whether a state s is the goal and can be explicit, i.e., an explicit list with states that are goal, or implicit, which is defined by a condition:

$$o(s) = \{\text{true, false}\} \tag{3.2}$$

- path cost, which assignes a cost to each path from s_0 to any other state.

After the formulation of the search problem, we have to find a solution, by exploring the state space, composed of all possible states that could be reached by an agent. The tool used by search algorithms for finding a solution is the *search tree*, which is an explicit tree, generated starting from the initial state and the transition function, that define the state space. The basic element of a search tree is a *search node*. A search node can be represented in several ways. A possible data structure for a search node is composed by five elements:

- state, which is the state in the state space represented by the node;
- parent node, which is the node in the search tree that generated this node;
- action, which is the action necessary to arrive from the parent node to this node;
- path cost $g(n)$, which is the path cost from the initial node to this node;
- depth, which is the number of steps from the initial node.

All nodes generated but not expanded yet (also called *leaf nodes*, because they still don't have any successor) are inserted in a set, called *search frontier*. It is important to remark that states are not equivalent to nodes, because there could be more than one node corresponding to the same state, for example when there are multiple paths from s_0 to that state.

From now on, we consider nodes instead of states, because they are the basic element used by search strategies.
There exist two classes of search strategies that could be applied to search problems in order to find the solution, by using a search tree:

- uninformed search strategies (e.g., breadth-first, uniform-cost, depth-first), which do not exploit any additional information about the problem;
- informed search strategies (e.g., A*, branch and bound), which, conversely, exploit specific information about the problem and, therefore, are able to find a solution more efficiently.

As we can see from Figure 3.1, the basic version of these search strategies is that they take a node from the frontier, according to some criteria specific to each strategy, they perform the objective test to the state corresponding to the node and, if the objective test returns false, they expand this node, by generating successor nodes of this node, and insert generated nodes in the frontier. These steps are performed in a loop until a solution is found.

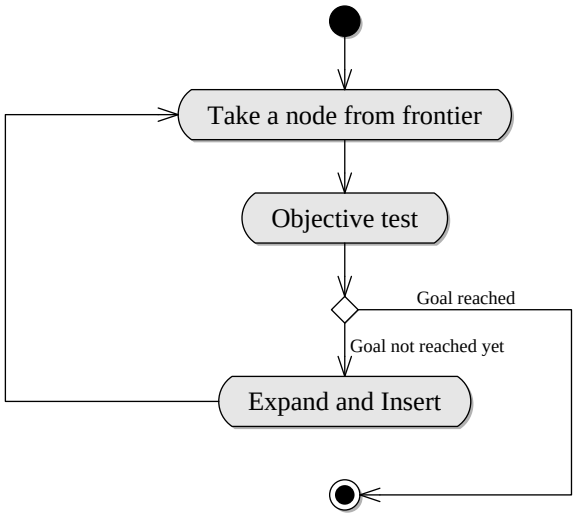


Figure 3.1: Basic algorithm of a search strategy.

One of the most challenging issues is the states repetition, namely the chance to expand multiple nodes with the same state. This issue might make

a search problem unsolvable, because the search algorithm cannot terminate due to the huge amount of nodes to expand. Hence, search strategies can be modified so that they are able to avoid to expand nodes that corresponds to states that have been already expanded, by adding a *closed list*, a structure that keeps in memory all expanded states. When a node in the frontier is chosen, before its expansion, the search algorithm checks whether a node with the same state has been already expanded. If this is the case, the search algorithm discovered two different paths that lead to the same state and so it can discard one of them. In this way, time complexity is reduced. However, in some cases, it could happen that an optimal path is discarded.

In this work, we used informed search strategies, which use heuristic as additional information for finding a solution. In this way, the time and memory complexity ought to be reduced, since it is not necessary to explore all possible paths to obtain the optimal one [20]. This class of search strategies looks at the sum of the current cost to reach a node n and the heuristic in that node:

$$f(n) = g(n) + h(n) \tag{3.3}$$

where $g(n)$ is the path cost from the initial node to the node n and $h(n)$ is the estimate of the cost to reach the goal from the node n.

The difference between A* and branch and bound is that the former expands nodes with the minimum $f(n)$ until it finds a solution, whereas the latter firstly finds a solution (e.g., using a depth-first search, which goes in depth of the tree, til it finds a solution), and then compares the solution found and prune other paths that are expected to cost more than the found solution.

It is significant to remark that heuristic is defined as a function $h(n)$ and represents an estimation of cost from a node n to the closest goal. It should have two properties, so that search strategies can find an optimal solution:

- admissibility, namely heuristic $h(n)$ never overestimates the cost to reach the goal:

  $$h(n) \leq C^*(n) \tag{3.4}$$

  where $C^*(n)$ is the actual cost to reach the goal node from node n;

- consistency, namely heuristic $h(n)$ is never greater than the cost to reach node n' (successor of n) summed to the heuristic from the new node $h(n')$:

  $$h(n) \leq c(n, a, n') + h(n') \tag{3.5}$$

It has been proved that consistency implies admissibility, but the converse is not true. So, it is sufficient to design an heuristic that is consistent. Still, most admissible heuristics that are useful in practice happen to also be consistent.

Each search strategy can be evaluated according to the following four evaluation criteria:

- completeness, namely a strategy is complete when it is able to always return a solution;
- time complexity, namely time spent to find the solution;
- space complexity, namely amount of memory used to keep the search tree;
- optimality, namely a strategy is optimal when it always returns an optimal solution.

Figure 3.2 reports properties of A* and branch and bound search algorithms.

	A*	Branch&Bound
Complete	Yes (unless there are infinitely many nodes with $f \leq f(G)$)	Yes (as A*)
Time	$O(b^m)$ Exponential in [relative error in h $\times$ length of solution	$O(mb)$
Space	$O(b^m)$ (Keeps all nodes in memory)	$O(b^m)$
Optimal?	Yes (if heuristic is consistent)	Yes

Figure 3.2: Evaluation of search strategies (b is branching factor, m maximum depth of paths in space state, h is heuristic).

Both algorithms fit with our requirements. Indeed, they return an optimal solution, obviously if the heuristic is correctly designed. In terms of space, they are equivalent, even if branch and bound can prune some paths and thus remove them from memory, whereas A* should keep all nodes in memory til the end. In terms of time, A* greatly depends on the definition of the heuristic. If heuristic is very close to the actual cost, A* search algorithm perform very well, while if it is not, the search algorithm have to check all possible paths,degenerating to breadth-first search algorithms. Branch

and bound, besides the definition of the heuristic, depends on the first solution found, because the first solution affects the pruning steps performed by branch and bound.

3.2 Preliminary hypotheses

To formulate the problem of determining an optimal exploration path in a specific environment, we have to make some assumptions.

First of all, we assume that the environment is known, so that only computational operations are performed. In this way, as said before, off-line search algorithms are applicable.
Furthermore, the world is deterministic, observable and static, so that it is not necessary to model changes that could happen in the world.

Secondly, we consider a single robot, that is represented as a point, and thus it has not a dimension.

Then, we assume that the perception model and the movement model are perfect, that is there is not any error during perception or localization, due to sensors or movement errors.
The movement model follows 8-adjacency connection of cells, so the robot is able to move in every direction.
Moreover, its movement within the map is independent of the ground, namely the effort of a movement action is not considered.

Finally, the map of the world is modeled as a 2D map (see Figure 3.3), with a world-centric reference, so that it is not important the 3D shape of obstacles, but just their projection to the ground.

Figure 3.3: An example of map, where each pixel represents a cell of a predefined size.

These hypotheses are fixed and are justified since we are searching an optimal exploration path in a specific environment. Other parameters, that define the initial conditions of the search problem and constraints can be tuned, so that we can relax the search problem (they are presented in Section 3.3.2).

3.3 Problem formulation in exploration case

In this section, firstly, we introduce the general formulation of optimal exploration as a search problem. This general formulation allows to find the optimal exploration path in a specific environment. Then, since this search problem has exponential complexity, we analyze parameters and constraints that can reduce its complexity, by highlighting relations between time complexity and optimality.

3.3.1 General formulation

As already asserted in Section 2.1, classical formulation for on-line algorithms is not applicable to the problem of determining the optimal exploration path in a specific environment, because of state representation, i.e., a point where a robot could find itself. Moreover, these formulations require to specify a destination point, but, in most cases, the destination point of an exploration is not known a priori and it is relevant to observe that a robot cannot know successors of a state, unless it tries all possible actions.

From the point of view of determining the optimality of an exploration in a specific environment, off-line search algorithms should be applied, so that the process of determining the optimal path is just computational. It is possible to apply them, because we assumed that the map of an environment is already known, and the world is deterministic and static. In this way, it is possible to know successors of a state, that is the results of an action performed in a specific state. Furthermore, state can and should be represented as the partial knowledge of the map, instead of the destination point.

Following theory about search problems recalled in Section 3.1, the exploration problem can be formulated as follows:

- state s: it includes the map discovered until a generic time t, which can be represented in several ways (e.g., as a grid), and the pose of the robot in the map, which considers coordinates of the position of the robot in the map and the orientation of the robot, thus identified by a pair:
$$s = \text{<discovered\_map, robot\_pose>} \tag{3.6}$$
- initial state s_0: it is an element of the space of states, i.e., where the robot is deployed in the environment at the beginning and the initial knowledge of the map (e.g., null);

- actions: from a given pose, the robot can perceive the surroundings and move to another pose, which is taken from the boundary (i.e., successor points in the physical space frontier, called with another name just not to confuse with the frontier used by search algorithms) in some way:

$$\text{Ac(pose)} = \{\text{perceive(pose), moveTo(pose)}\} \tag{3.7}$$

- transition function: it is a function that returns the state resulting from an action:

$$T(s, a) = s' \tag{3.8}$$

  where $s =$ <discovered_map, robot_pose>
  and where, if perception action is performed, the new state s' has the same robot pose and an updated map,
  $a =$ perceive
  $s' =$ <updated_map, robot_pose>
  and where, if movement action is performed, the state resulting from moving to another point is the change of the robot pose but the same discovered map,
  $a =$ moveTo
  $s' =$ <discovered_map, robot_pose$'$>

- objective test: is the percentage of the map discovered til now greater or equal than the objective?

$$o(n) \geq G \tag{3.9}$$

  where $o(n)$ is a function that returns the current percentage of discovered map in node n and G is the objective threshold;

- path cost: a function $g(n)$ that computes the cost of the path to arrive to n; we consider two cases to find the best path, one minimizing number of perceptions and the other one minimizing the distance, i.e., we just considered the case in which perception cost is 1 and cost of traveled distance is 0, and viceversa, otherwise more complex functions should define a combination of movement and perceptions costs.

An important aspect that has to be designed carefully is the definition of an appropriate heuristic, in order to use informed search strategies. Obviously, heuristic is related to what we are optimizing and it ought to be different, when we consider either the number of perceptions or the traveled distance as path cost. As already said, it is relevant to ensure the property

of consistency in order to always find the optimum. So the heuristics defined for the two cases are:

- expected number of perceptions to fully map the environment, that is the number of free unknown cells in the map of the state s of the node n divided by maximum number of perceivable cells, depending on sensor range of the robot:

$$h_1(n) = \frac{\text{free\_unknown\_cells}(n)}{\text{perceivable\_cells}} \tag{3.10}$$

- expected traveled distance to fully map the environment: the furthest point in unknown part of the map from the point in the boundary (note that perception capability should be considered for determining the furthest point):

$$h_2(n) = \text{furthest\_point}(n) - \text{sensor\_range} \tag{3.11}$$

They are admissible and consistent, because, for the first, we need at least the number of perceptions returned h_1, so it is less than the actual cost to reach the goal; for the second, we just consider one part of unknown map, but there could be other unknown parts of the map (this is just an intuition; a formal proof is not provided here).

3.3.2 Parameters and constraints of the search problem

The formulation reported in Section 3.3.1 is a general formulation, that is able to find the optimal exploration realization. However, since solving the search problem has exponential complexity, we ought to define some simplifications, which lead to a reduction in time complexity, but could worsen the optimal exploration path that can be found.

First of all, we have to define how the boundary set is generated. The method to generate next possible robot poses, from a state s, is the boundary-based one, namely those cells near, according to 8-adjancency, to unknown cells are cells in the boundary. The idea is that, when the robot is close to boundary cells, it is able to perceive more. Figure 3.4 shows this method. In particular, lighter gray cells are all boundary cells, since they are near to unknown black cells.

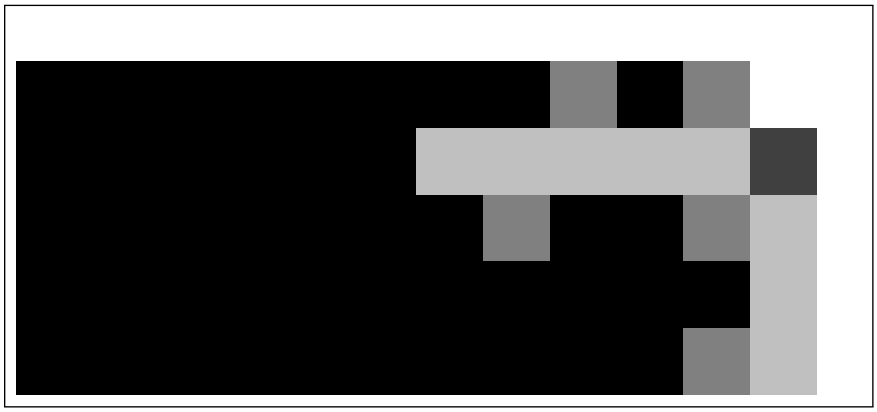


Figure 3.4: Boundary cells in a simple environment. Legend: Black=unknown, lighter grey=boundary cells, darker grey=robot, white=discovered free cells.

The boundary-based method does not affect optimal solution. Suppose that we take a cell c adjacent to a boundary cell c', which is found with the method above. If a perception action is performed in c, then the mapped area is always included in the mapped area of a perception performed in c' as Figure 3.5 shows.

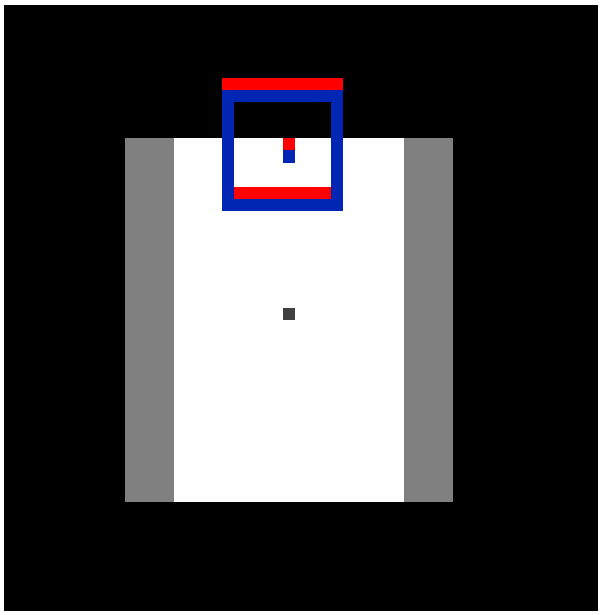


Figure 3.5: Mapped area in a boundary cell and in an adjacent one (red cell is the boundary cell, blue cell is the adjacent cell).

In the initial state, we can suppose that the robot has already perceived the surroundings (if perception cost is the cost function, then in the initial

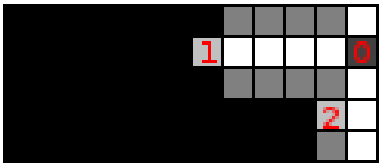
node the cost is initialized to 1), and actions of perception and movement could be merged together, in order to reduce branching factor and depth of the tree. In such a case, the new set of actions becomes:

$$\text{Ac(pose)} = \text{move\&perceive(pose)} \tag{3.12}$$

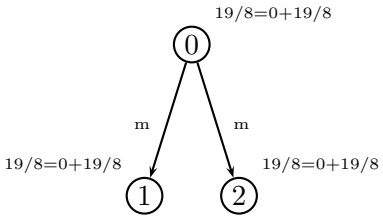
and the new state s' returned by the transition function $T(s, a)$ is:

$$s' = <\text{updated\_map, robot\_pose}'> \tag{3.13}$$

This assumption is not restrictive and does not eliminate possible optimal solutions, because the robot needs to perform both actions to explore and map the entire environment. Consider the case where the robot is in a certain state, that is not the goal, and can choose a boundary cell: if the robot performs just the movement action to a boundary cell, still it does not find itself in the goal, because the mapped percentage of the environment is still the same. The robot needs to perform the action of perception, to possibly reach the goal.
This simplification really cuts the complexity. Indeed, if the actions are separated, there would be a huge amount of repetition of states, since we should consider, apart from the perception action in a node, even the movement action to all others boundary cells that have been found in the parent node. The repetition of states due to movement action, in case the cost function is the number of perceptions, could cause the search algorithm to end up in a loop, resulting from the movement between two points, because the cost of a movement is 0.
Furthermore, about the heuristic h_1 (see Equation 3.10) - i.e., if we consider again as cost function the number of perceptions - we can observe that in case the two actions are separated, when a child node n' is a result of a movement action, the heuristic of n' (actually all child nodes of n) is equal to the heuristic of the parent node n, i.e., $h_1(n) = h_1(n')$. Consider again the environment depicted in Figure 3.3; the number of free cells is 30 and the number of occupied cells is 30. Figure 3.6a shows the initial discovered map, the robot pose, indicated by 0, and boundary cells, indicated by 1 and 2. Suppose that the robot sensor range is 1, i.e., it can discover 8 cells around itself. Figure 3.6b illustrates the search tree in this situation, with function f related to each node (current path cost summed to heuristic h_1 of each node) .



(a) Considered state.

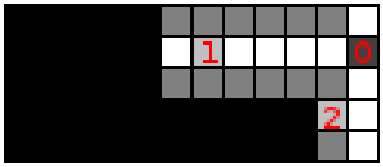


(b) Heuristic in node 1.

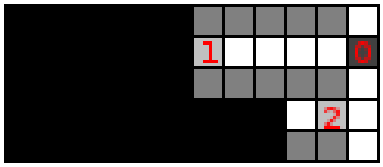
Figure 3.6: Example that shows that movement action alone does not change heuristic h_1.

As we can see, $f(0) = f(1) = f(2)$, because movement action costs 0 and there is not any change in the discovered map (indeed, the heuristic h_1 is the same in all nodes).

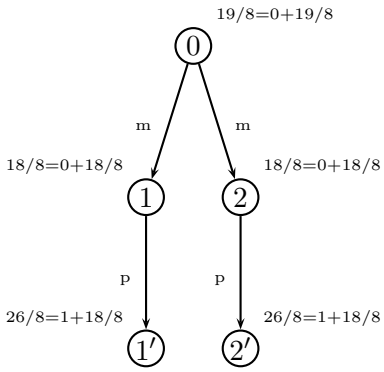
Since we assumed that the environment is a priori known, the heuristic can be computed as if the perception action is performed in the boundary cell. Even in this case, we have the problem that other two heuristics are equal, namely, the child node n' and the other child node n'' deriving from a perception action in n' ($h_1(n') = h_1(n'')$). Figure 3.6b illustrates this situation. We can see that before expanding the nodes at depth level 2, the search algorithm expands all the nodes at depht level 1, because heuristics are the same, but the nodes generated from a movement action do not have any cost in terms of number of perceptions. As a matter of fact, $f(1) = f(2) < f(1') = f(2')$. So the search algorithm expands firstly nodes 1 and 2 and then $1'$ and $2'$.



(a) New state if perception is performed in pose 1.



(b) New state if perception is performed in pose 2.



(c) Heuristic in node 1.

Figure 3.7: Example that shows that even adding prospecting level of 1 does not change heuristic h_1.

This could lead to a degeneration of A* algorithm in a breadth-first algorithm.
Therefore, we can consider both actions performed together, without eliminating any optimal solution.
Figure 3.8 illustrates that depth and branching factor of a tree are reduced and so time complexity for search algorithms is lower.

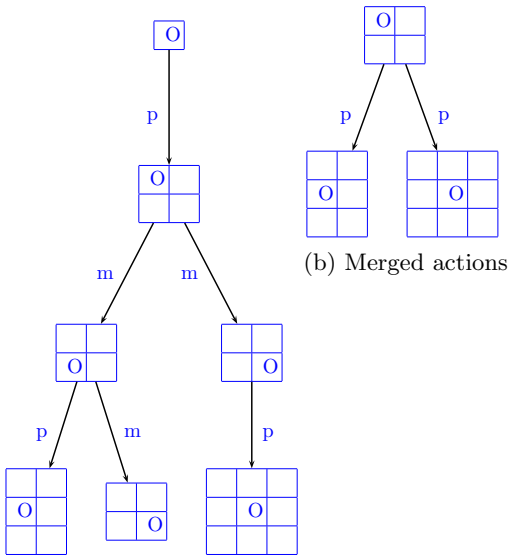


Figure 3.8: Search trees considering both actions merged and not.

It is important to note that nothing changes for function that calculate the path cost. For example, if we consider number of perceptions as cost function, then, when the robot performs an action, just the perception action is considered for computing the path cost. Instead, if we consider traveled distance as cost function, then only the movement action is considered for computing the path cost.

Besides these simplifications that do not affect solution optimality, there are a series of initial conditions (parameters of the exploration problem formulation) that can affect the search.
First of all, it is possible to define different objective tests, according to how much percentage G of the map should be discovered, so that search algorithm terminates as soon as percentage G of the map has been discovered. Even in this case, optimality of the solution is not affected in terms of exploration path, but simply changes according to the objective to accomplish.
Moreover, it is possible to set various initial states, namely the initial robot poses. Some robot poses could lead to an increase of time complexity, because they could have more choices to analyze.

Other parameters related to specific model and implementation of the robot and the environment can be taken into consideration for reducing the time complexity. Specifically, we have:

- the perception model of the robot, because each perception model

returns a different part of environment perceived;

- resolution of the map (if grid maps are used), i.e., the bigger the size of a cell, the less amount of memory is used and the less boundary cells the robot finds.

However, there is a trade-off between realism and time complexity.
In the case of perception model, if the robot use a naive one (e.g., footprint sensor), we reduce time complexity because the sensor is able to see even through obstacles, but the sensor is not realistic. Instead, if the perception model is very complex (e.g., laser sensor), it is very realistic, but the time complexity increases. By changing the perception model, the solution optimality is preserved, because the boundary set is not restricted, even if the exploration problem could be less realistic and almost impossible to reach in real settings.
In the case of the resolution of the map, with lower map resolution, the boundary set is smaller, because there are less cells to consider, but details of environment are lost, whereas with higher map resolution, the boundary set is bigger, and details of environment are kept. So, there is a trade-off between the environment details and the number of boundary cells. Figure 3.9 shows the same map with two different cell size.

(a) Cell size=0.5 (b) Cell size=0.25

Figure 3.9: Map with different resolution.

In this case, the optimality of a solution is affected, in the sense that, with same initial conditions, if we have less details (i.e., lower resolution), the exploration solution can be better than the exploration solution, if we have more details (i.e., higher resolution), because some actions cannot be performed in a more realistic environment.

Finally, a series of constraints in the selection of boundary cells can be imposed, in order to reduce the size of the boundary set and therefore to simplify the search problem, namely the search algorithm does not consider all boundary cells, but just a subset of them. Section 4.3 shows how these constraints have been designed.

3.4 Examples

In the following sections, we provide some simple examples that test our general formulation of the search problem for determining the optimal exploration solution in a specific environment, and the heuristics.

3.4.1 Formulation

This example verifies the formulation of the problem provided in the Section 3.3. In particular, we consider a simple environment, which can be represented as a grid (see Figure 3.10). The starting point of the robot is marked in the top left cell in the grid.

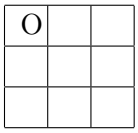


Figure 3.10: Map of the environment

We assume that the robot can perceive cells distant 1 from the robot pose in all directions (i.e., horizontally, vertically and diagonally). In general, we assume that the robot can move from its pose to a boundary (i.e., more than 1 step). Here, we run breadth-first search algorithm to test the formulation, and we can see the tree (see Figure 3.11) that results from the run of the search algorithm. The creation of nodes is as follows: new nodes are placed from left to right. We suppose that if there are nodes on equal level, the robot chooses the left-most node. As we can see, after two expansions of nodes, the algorithm finds the optimal solution that is the one circled by a green line; the cost is just 1, if we consider both traveled distance and number of perceptions.

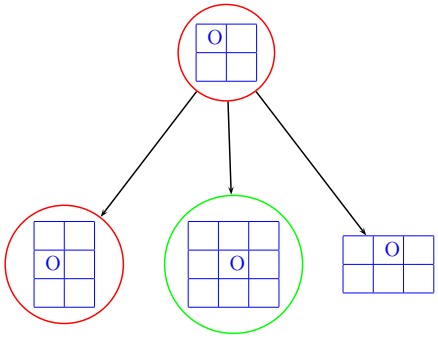


Figure 3.11: Tree constructed by breadth-first search. Legend: red circle=expanded but not goal node, green circle=expanded goal node.

3.4.2 Heuristics

These examples verify whether the heuristics previously defined in Section 3.3 do actually make sense or not. We consider the environment depicted in Figure 3.12, where white cells are known cells, grey cells are obstacle cells and black cells are unknown cells.

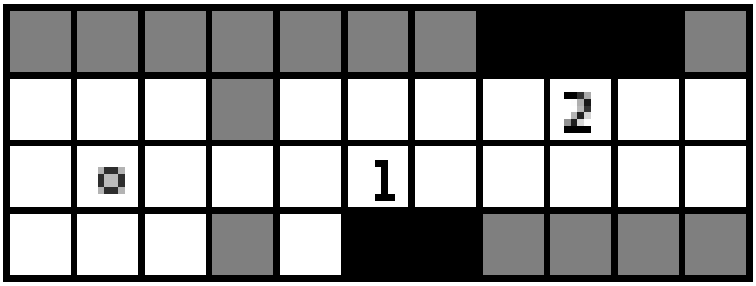


Figure 3.12: Environment considered for testing heuristics. Legend: grey=obstacle cells, white=free known cells, black=unknown cells.

The cell with a 0 is the robot pose, while cell with 1 and 2 are two boundary cells (for simplicity, we do not consider any other boundary cells), that are considered by the search algorithm for generating successors.

About the heuristic h_1 (see Equation 3.10), we can see that our heuristic definition does make sense, as it is shown in Figure 3.13, which illustrates the execution of A*, reporting the tree with expanded nodes (red nodes), goal node (green node) and not expanded nodes (black nodes).

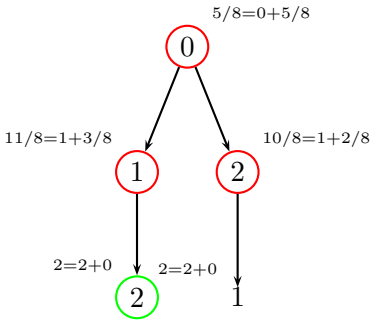


Figure 3.13: Execution of A* to minimize perceptions. Legend: red circle=expanded but not goal node, green circle=expanded goal node.

On each node is reported its $f(n)$ value, namely the path cost summed to the heuristic in the considered node. As we can see, heuristic h_1 defined is admissible and consistent.

About the heuristic h_2 (see Equation 3.11) - i.e., the one that minimizes the traveled distance - it worked as expected. We measured the furthest point with Euclidean distance.

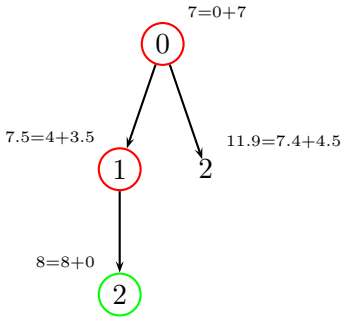


Figure 3.14: Execution of A* to minimize traveled distance. Legend: red circle=expanded but not goal node, green circle=expanded goal node.

As we can see in Figure 3.14, which shows the execution of A*, reporting the search tree, heuristic h_2 is admissible and consistent.
The solution returned by A* algorithm, by applying these two heuristics, is the actual optimal path.

Chapter 4

Implementation

In this chapter, we present the implementation of the algorithms for solving the search problem formulated in Section 3.3. We also discuss what parameters and constraints can be added/modified in order to relax the search problem. All constraints that reduce the size of the boundary set influence also the optimal exploration realization, since some possible optimal paths could be discarded.

In Section 4.1, we describe the framework in terms of main modules and of the main functionalities they provide. In Section 4.2, we show the basic code. In Section 4.3, we report how it is possible to relax the problem, introducing further assumptions to increase efficiency.

4.1 Description of the framework

The framework is designed and developed to be very modular.

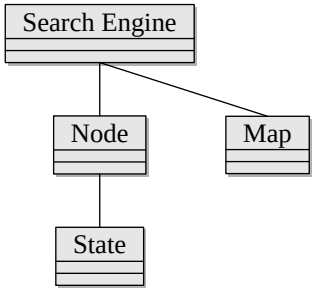


Figure 4.1: Architecture of the framework.

As shown in Figure 4.1, the core class is the *Search Engine*, which starts the search over the search tree, by starting from the initial state. It is possible to use different algorithms to search over the search tree. Here, we

implemented two search algorithms, that is A* and Branch and Bound. The other three classes are:

- *Map*: which keeps in memory the map of the environment;
- *Node*: which obviously represents a node in the search tree, with the same data structure as presented in Section 3.1;
- *State*: which represents a state of a node.

When in a state, the robot can perform an action that, as already introduced in Section 3.3.2, combines perception and movement.

The perception is performed according to the perception model implemented. In this implementation, we provided two perception models, even if it is possible to implement others (more details are provided in Section 4.3):

- laser sensor, which can sense surroundings within a pre-defined range and realistically cannot see beyond an obstacle;
- footprint sensor, which is a dummy sensor that is capable of perceive everything within the range (even through obstacles).

4.2 Core code

The framework core activities are shown in Figure 4.2.

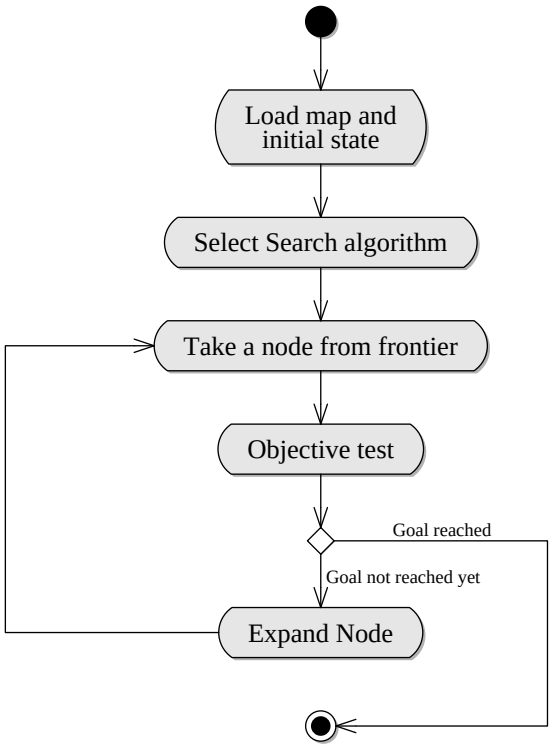


Figure 4.2: An overview of framework activities flow of search engine.

Steps are those usually employed in search algorithms. Specifically, at first, the Search Engine loads the initial state, that is the initial robot pose, and the map, and the root node of the search tree is created. Then, the search algorithm selected by the user is started. The search algorithm takes a node from the frontier and tests whether the objective function is fulfilled or not, namely it checks whether the whole free space of the environment is mapped or not. If this is the case, then a solution is found, otherwise it expands the node, generating its successors, and picks another node from the frontier, until the solution is found.

When a node is expanded, transition function, as defined in Section 3.3.1, finds successor nodes, and each generated node n is inserted in an ordered frontier by the function $f(n)$ (see Equation 3.3), i.e., the estimated cost of a solution that passes from the expanded node n. Figure 4.3) illustrates steps performed when a node is expanded.

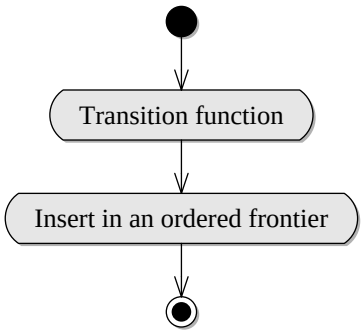


Figure 4.3: Expansion of a node.

Successor nodes are basically those states, where the robot is in a boundary cell, defined in the map corresponding to the parent node, and has perceived the environment from that cell. In transition function, as shown in Figure 4.4, the framework ought to find boundary cells, defined as those adjacent to unknown cells, according to 8-adjacency. Then, robot performs actions, i.e., it moves to boundary cells and perceives surroundings from there, and so nodes are created with those new states. It is relevant to remark that perception action is performed only when the robot reaches the destination, and not continuously, while it moves around. The cost of the path is updated on the basis of the weights passed as input to each action.

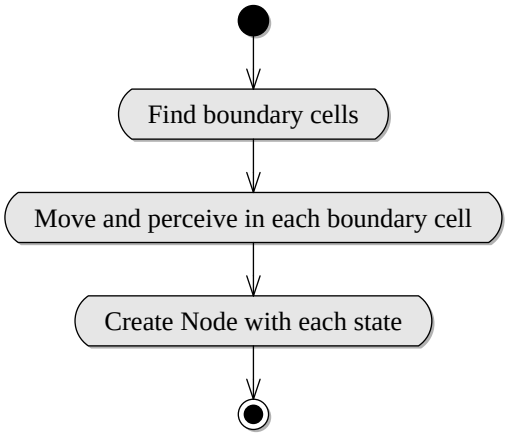


Figure 4.4: Transition function activity diagram.

In the general implementation of this framework, all boundary cells are generated and considered as possible destinations for the robot, according to the boundary-based method, introduced in Section 3.3.2. In this way, the framework is modular and it is possible to add constraints that can restrict the boundary, as presented in Section 4.3.

In the following listings, the core of A* algorithm is presented, with all functions that are needed to the A* algorithm to perform the search, i.e., expansion of a node, node insertion in the frontier, and transition function.

```
1  function Astar-search(problem, strategy) return realization of
       steps, or failure
2    frontier ← Insert(Create-Node(Initial-State[problem]),
         frontier)
3    loop do
4      if Empty?(frontier) then return failure
5      node ← Pop-First(frontier)
6      if Goal-Test[problem](State[node])
7        then return Solution(node)
8      frontier ← Insert-All(Expand(node, problem), frontier)
```

Listing 4.1: A* pseudo-code.

```
1  function Expand(node, problem) return set of nodes
2    successors ← empty set
3    for each <action, result> in Transition-Function[problem](
         State[node]) do
4      s ← new Node
5      Parent-Node[s] ← node
6      Action[s] ← action
7      path_cost[s] ← path_cost[node] + step_cost(node, action, s)
8      depth[s] ← depth[node] + 1
9      add s to successors
10   return successors
```

Listing 4.2: Node Expansion.

```
1  function Insert-All(successors, frontier)
2    for each new_node in successors do
3      for i=1 to size[frontier] do
4        node ← i[frontier]
5        if (path_cost[node] + heuristic[node] <= path_cost[
             new_node] + heuristic[new_node])
6          then  Insert new_node before node and break
```

Listing 4.3: Implementation of insertion of generated nodes in the frontier.

```
1  function Transition-Function(state) return set of states
2    successors ← empty set
3    for each cell in Find-Frontier-Cells(state)
4      new_state ← CreateNewState(cell)
5      Perceive(new_state)
6      add new_state to successors
7    return successors
```

Listing 4.4: Transition Function.

In the Listing 4.5, it is presented the branch and bound pseudocode and in the Listing 4.6 is presented the insertion of nodes in the frontier (other functions used by branch and bound are the same presented above).

```
1  function BranchAndBound-search(problem, strategy) return
       realization of steps, or failure
2    frontier ← Insert(Create-Node(Initial-State[problem]),
       frontier)
3    sol_node ← NULL
4    while (not Empty?(frontier))
5      node ← Pop-Last(frontier)
6      if (sol_node == NULL || Path-Cost(node)+Heuristic(node) <
         Path-Cost(sol_node))
7        then
8            if Goal-Test[problem](State[node]) then sol_node =
                 node
9            else frontier ← Insert-BB-All(Expand(node, problem),
                 frontier, elements)
10   if sol_node not NULL then return Solution(node)
11   else  return failure
```

Listing 4.5: Branch and bound pseudo-code.

The elements parameter is the displacement, where the insertion of new nodes starts. It is necessary, since branch and bound performs a depth-first search algorithm in order to find a solution.

```
1  function Insert-BB-All(successors, frontier, elements)
2    for each new_node in successors do
3      for i=elements to size[frontier] do
4        node ← i[frontier]
5        if (path_cost[node] + heuristic[node] >= path_cost[
             new_node] + heuristic[new_node])
6          then  Insert new_node before node and break
```

Listing 4.6: Implementation of insertion of generated nodes in the frontier for branch and bound.

In the Listing 4.7, we show how the heuristics h_1 and h_2 are computed.

```
1  function computePerceptionHeuristic(state)
2    for each free cell in a map do
3      count number of cells Perceived in that state
4      take maximum_perceivable
5    return free_cell_still_not_visible / maximum_perceivable
6
7  function ComputeDistanceHeuristic(state)
8    fc ← find the furthest cell in a not visible area
9    return distance(robot_pose, fc)
```

Listing 4.7: Heuristics.

About h_1 (i.e., perception heuristic), its computation can be updated at each state, because in each state, the maximum number of perceivable cells can be calculated, so that we have a heuristic near to the actual solution cost. However, if the maximum number of cells perceivable is equivalent to the whole area that could be mapped with the sensor, then we can consider a part of it, since it is impossible that the robot find itself in a totally unknown part. Specifically, the maximum number of cells perceivable is given by the following formula:

$$\text{maximum\_perceivable} = \frac{((r * 2 + 1)^2 - 1)}{2} + r^2 \tag{4.1}$$

This is the case when the robot is in a vertex of a square, as we can see in Figure 4.5. Indeed, if we assume a reference system where the robot is the origin, the robot is able to see three quadrants of unexplored area.

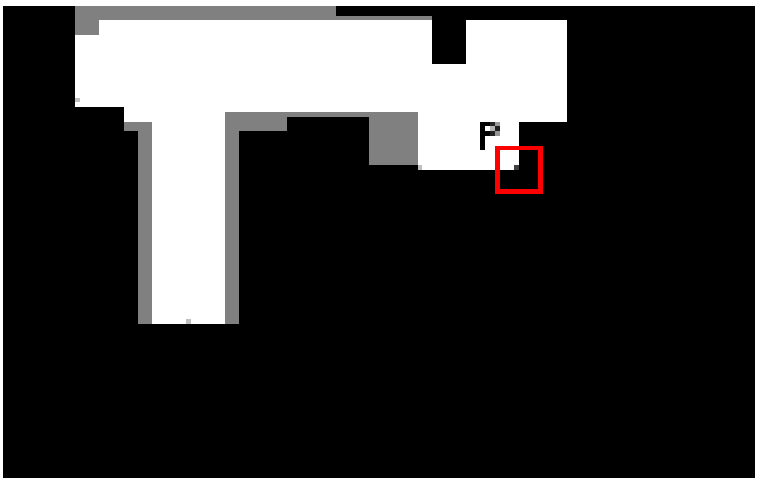


Figure 4.5: Perceivable area (red square) in point P (center of the square).

4.3 Relaxation of the problem

As already discussed in Section 3.3.2 and as we can see from the code, we can intervene at different steps, to relax the search problem and find its solution more efficiently. Specifically, if we want to modify the search algorithm we can modify various components:

- objective test, that is changing the percentage of required mapped area;
- nodes expansion, that is choosing not all boundary cells, but some of them;

- frontier ordering, by changing the function that determines the order.

In the following, we see some initial parameters that can be tuned and some constraints on boundary cells that can be activated.

4.3.1 Initial parameters

Objective test

Regarding the objective test (see Equation 3.9), it is possible to tune the constant G, namely the percentage that the robot should map of the environment. By diminishing G, the robot should perform less actions in order to explore the environment, and so the tree depth is reduced. Therefore, the time complexity is reduced.

(a) $G = 0.85$ (b) $G = 0.95$

Figure 4.6: Mapped area with different constant G.

Figure 4.6 shows two maps of the same area with different constant G. The tests with different G are run with same initial conditions, i.e., same initial robot pose, same perception model and range, same constraints and same cost function. In the case of Figure 4.6a, the optimal exploration path, by considering number of perceptions as cost function, costs 26, and the search algorithm needs 2.55 seconds for finding the solution. In the case of Figure 4.6b, the optimal exploration path costs 29 and the search algorithm needs 34.71 seconds for solving the search problem. The time complexity is reduced, because in the first case the search algorithm visits nodes at maximum depth level of 26, while in the second case the maximum depth level is 29.

Due to the parametric objective test, even the heuristics h_1 and h_2 ought to be defined accordingly, to ensure consistency. In such a case, the new definitions of the heuristics are the following:

$$h_1'(n) = \frac{\text{missing\_free\_unknown\_cells}}{\text{perceivable\_cells}} \tag{4.2}$$

where missing_free_unknown_cells are the number of cells that still has to be explored, depending on the value of percentage of the map G, that has to be mapped;

$$h_2'(n) = (\text{furthest\_point(n)} - \text{sensor\_range}) * (G - \text{current\_percentage}(n)) \tag{4.3}$$

where G is the percentage of the map to be discovered and current_percentage is a function that returns is the current percentage of the map already explored in n.

Perception Model

The perception model adopted are the footprint sensor and the laser sensor, and they can be chosen by the user. Here, we simplify perception models by assuming that the robot can sense at 360 degrees, so that orientation can be neglected. As seen in Figure 4.7, footprint sensor is able to discover more cell than laser sensor. The former perception model is dummy, in the sense that it perceives a footprint of the environment within the range of the sensor, even though there is any obstacle. The latter is more realistic, since it cannot see through an obstacle. Indeed, the robot with the laser sensor requires more perception actions to map the same percentage of the environment as with the footprint sensor, hence, footprint sensor has lower time complexity. In Figure 4.7, results of a perception of the two sensors are shown.

(a) Footprint Sensors (b) Laser Sensor

Figure 4.7: Results of a perception of the implemented sensors.

We can observe that there are several cells that laser sensor is not able to discover, while fooprint sensor is. In the initial settings of Figure 4.7 and in the environment of Figure 3.3, considering as cost function the number of perceptions, in the case of footprint sensor, the optimal exploration path costs 15, while, in the case of laser sensor, the exploration solution costs 22. In the Listing 4.8, we present the code of the laser sensor model. It is very sophisticated and very close to reality, indeed it uses shadow cones deriving from obstacles to determine whether a cell is visible or not, namely, each

obstacle cell create a shadow cone, and if a cell falls in that shadow cone, then it is not visible.

```
1  function Perceive(state)
2    for i=1 to range do
3      for each cell in radius(range) do
4        if notVisible(cell) then
5          if cell center not in a shadow cone then
6            makeVisible(cell)
7            if obstacle then
8              update shadow cones
9        else
10         if obstacle then
11           update shadow cones
```

Listing 4.8: Laser perception model.

Due to the modularity of this framework, even other perception models could be developed and added to this framework.

Closed list

About the constraints for reducing the size of the search frontier, we can recall the concept of *closed list*, already introduced in Section 3.1, namely, search algorithm can avoid repeated states.

```
1  function Astar-search(problem, strategy) return realization of
       steps, or failure
2    frontier ← Insert(Create-Node(Initial-State[problem]),
         frontier)
3    closed_list ← empty
4    loop do
5      if Empty?(frontier) then return failure
6      node ← Pop-First(frontier)
7      for each expanded_node in closed_list do
8            if(node >= expanded_node) goto end
9      if Goal-Test[problem](State[node])
10       then return Solution(node)
11     frontier ← Insert-All(Expand(node, problem), frontier)
12       end:
```

Listing 4.9: A* with closed list pseudo-code.

In Listing 4.9, modified A* pseudo-code is reported. Before testing the node with the objective test, it is checked whether the considered node is greater or equal than nodes in closed_list. A node n_1 is greater or equal than a node n_2 when $g(n_1) \geq g(n_2)$ and n_1.state == n_2.state. The equivalence of states is verified when the perceived maps in both states are perfectly

the same, when number of steps is considered. Additionally, the robot pose equivalence must hold in the case of traveled distance, because two nodes can be very different in terms of solution, if robot poses are different. With this definition of equivalence, the optimality is preserved in both cases, because, on the forme case, if n_1, whose path cost is greater than the path cost of the other node, is discarded, we can reach the same solution from node n_2, though, because boundary cells are the same in both nodes states, because they are equivalent in terms of mapped area; in the latter, it is deleted paths that lead to the same state, considering even the robot pose equivalence in the states. This modification can be activated through a Boolean variable, that can be passed to the Search Engine.

4.3.2 Constraints

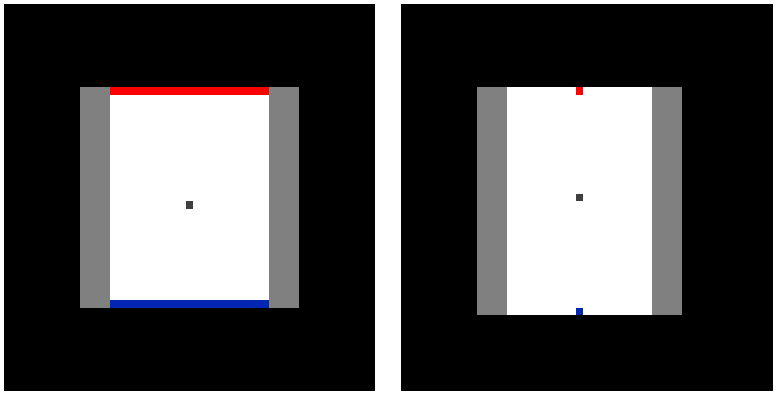
In addition, we can make assumptions on the boundary set, so that some boundary cells are discarded and, thus, the frontier is smaller.

Centroid

The first idea is to group boundary cells in clusters, that are sets of homogeneous elements. In this case, we consider as cells in a cluster the boundary cells that are adjacent. Then, *k-means clustering* can be applied (by activating this constraint with a Boolean variable) for choosing a cell in each cluster. Specifically, the algorithm calculates the centroid of a cluster, which is determined in the following way: Given a finite set of points $x_0, \ldots, x_k \in \mathbb{R}^2$ that belong to the cluster,

$$C = \frac{\sum_{i=0}^{k} x_i}{k} \tag{4.4}$$

Then, the nearest cell to the centroid is chosen, whose distance is calculated using Euclidean distance. Figure 4.8 illustrates which cells are in the same cluster and the related centroids.



(a) Two identified clusters (b) Two computed centroids

Figure 4.8: Results of applying clustering function.

Listing 4.10 shows the pseudocode for clustering function.

```
1 function SelectCentroid(boundary_list)
2   clusters ← empty
3   for each cell in boundary_list do
4     clusters ← InsertInCorrectCluster(cell)
5   for each cluster in clusters do
6     centroid ← ComputeCentroid(cluster)
7     select nearest cell to centroid of cluster
8     eliminate other cells of cluster
9   return boundary_list
```

Listing 4.10: Select centroids of found clusters.

Cluster size

In addition, it is possible to reduce the number of clusters, by discarding all clusters that are too small, i.e., a cluster should contain a minimum number of cells k:

$$\forall c \in \text{clusters} \mid \text{size}(c) \geq k \tag{4.5}$$

where c is a cluster, size is a function that returns the number of cells in cluster c and k is a positive integer, that can be set as desired. If the value of k is 0, then this constraint is not considered. The idea behind this constraint is that from cells belonging to small clusters negligible information about unknown areas is perceivable. It is relevant to remark that, depending on k value, this constraint could become too restrictive. For example, in the case of an indoor environment, where there are several rooms and the robot

find itself in the hallway, if k is equal to the number of cells representing the door space, the search algorithm cannot find a solution. In Figure 4.9, it is shown an example of environment, where this constraint does not allow the search algorithm to find a solution.

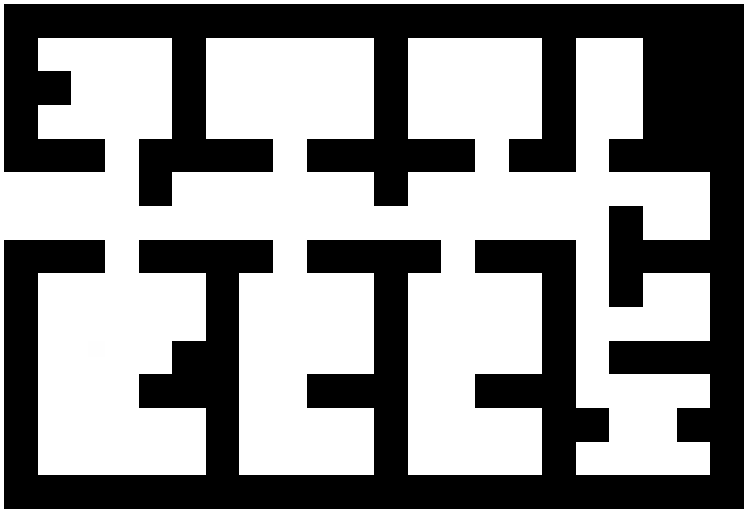


Figure 4.9: Indoor environment where the constraint on clusters size is a problem.

Safeness

The second idea is to define constraints on single boundary cells.
We can introduce the concept of *safeness*, that is boundary cells too close to obstacles are discarded. In this way, there is a guarantee that the robot does not hit any obstacle. Hence, the constraint imposed is, $\forall c \in$ boundary, $\forall o \in$ obstacles:

$$\text{distance}(c, o) \geq l \tag{4.6}$$

Distance constant l from obstacles is a positive integer and is a parameter that can be tuned.

Nearness

Furthermore, we can impose a *nearness* constraint, i.e., boundary size can be reduced by discarding boundary cells too far from the robot current position:

$$\text{distance}(\text{robot\_pose}, bc) \geq m \tag{4.7}$$

where bc is a boundary cell and m is the distance. In this way, in the case of cost function as number of perceptions, a possible duplicated path is

eliminated (e.g., first to go to the furthest point and then the nearest, and viceversa). The idea is to avoid that the robot goes too far and then going back, as shown in Figure 4.10.

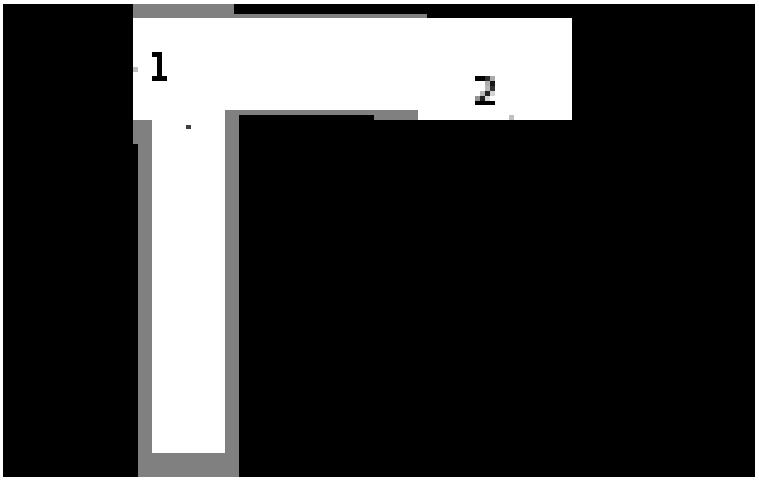


Figure 4.10: Nearness constraints.

If the constraint is activated, then the boundary cell 2 is discarded, while the boundary cell 1 is kept. In this way the robot, firstly, goes to 1 and after that it goes to 2. Nearness distance is a positive integer and can be tuned.

These are some of constraints implemented. However, because of the modularity of this framework, it is possible to add other constraints, in order to reduce complexity of this problem.

Chapter 5

Experimental results

In this chapter, we show experimental results from different perspectives. First of all, we analyze results obtained with our framework and compare them with results of on-line exploration strategies available in [2]. Then, we verify, changing parameters and making additional assumptions:

- how the quality of the exploration solution, as number of perceptions or as traveled distance, changes;
- how the computational time for finding a solution changes.

This chapter is organized as follows. In Section 5.1, we show initial conditions and assumptions used in experiments and we present how we performed tests. In Section 5.2, we analyze and compare optimal solutions obtained with our framework with results obtained in [2]. In Section 5.3, we observe how quality of solution and computational time change, by tuning different parameters.

5.1 Tests methodology and experimental setting

The framework used for testing has been implemented in C++, during the work on this thesis (the architecture and the relevant core code were presented in Chapter 4). The software allows to define different initial conditions and parameters:

- the environment to be explored;
- the initial pose of the robot;
- the size of the cell;

- sensor model and sensor range;
- the percentage of the map that has to be discovered;
- activation of different constraints on boundary cells.

The solution cost can be measured by either the number of perceptions or the traveled distance. Furthermore, about the search algorithms, it is possible to choose between A* and branch and bound, and if closed list is enabled or not.

The framework for determining an optimal exploration path has been initially applied in an indoor environment (see Figure 5.1), called office environment. This environment has been used in [2] for evaluating different exploration strategies.

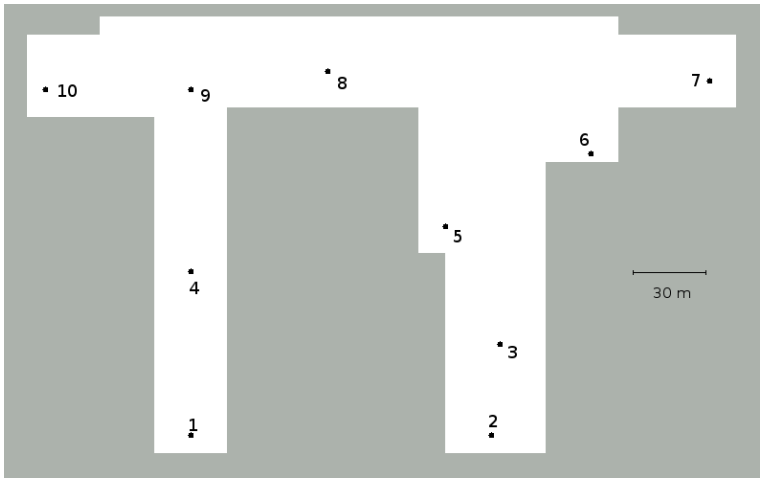


Figure 5.1: Indoor environment.

The reason why we chose this environment is that we want to compare real exploration strategies solutions with the optimal exploration solution calculated by our framework. In addition, we want to prove that this framework is applicable to realistic environments, where a robot can be deployed in.

We performed different trials, by changing initial conditions and parameters defined before, to have results that can be compared with results obtained in [2]:

- initial state (i.e., ten different initial robot poses, as shown in Figure 5.1), so that we can generalize the behavior of this framework, by computing the average and the standard deviation of the solution cost;

- cell size, in order to verify how computational time and solution quality change (we consider 1 m, 2 m and 4 m);
- sensor range (i.e., a range of $r = 20$ m, as in [2], and $r = 25$ m or $r = 30$ m, since $r = 10$ m and $r = 15$ m that appear in [2] require too much time to compute a solution), in order to compare optimal solution with on-line algorithms solutions;
- percentage of mapped environment (i.e., $G = 85\%$, $G = 90\%$ or $G = 95\%$, as in [2]), again to compare results of this framework with on-line exploration strategies;
- frontier pruning criteria: picking cluster centroid, because otherwise experiments would require too much time.

We used A* search algorithm and closed list enabled and the sensor model is the footprint sensor.

About single analyses on a subset of experiments, we changed frontier pruning criteria, i.e., minimum size of a cluster, safeness and nearness. Furthermore, we tested laser sensor and branch and bound search algorithm and we ran a set of trials in which the framework determined the exploration paths for complete explorations, namely goal percentage $G = 1$.

Metrics used to evaluate our framework are the quality of the solution, as number of perceptions (also called steps, according to [2]) or traveled distance, and computational time[1] to find a solution.

The expiration time, i.e., the interval in which the search algorithm is allowed to find the solution, is set to 18000 seconds (5 hours).

Other environments should be tested, such as an open space (see Figure 5.2), and a scattered one, i.e., with many obstacles located in the environment (see Figure 5.3).

[1]Actually, computational time could depend on computer hardware used for testing. Computer hardware specifications used for testing are: 1.60 GHz Intel Core i7-720QM Processor, RAM 8GB DDR3 (laptop *HP dv3-4010sl*).

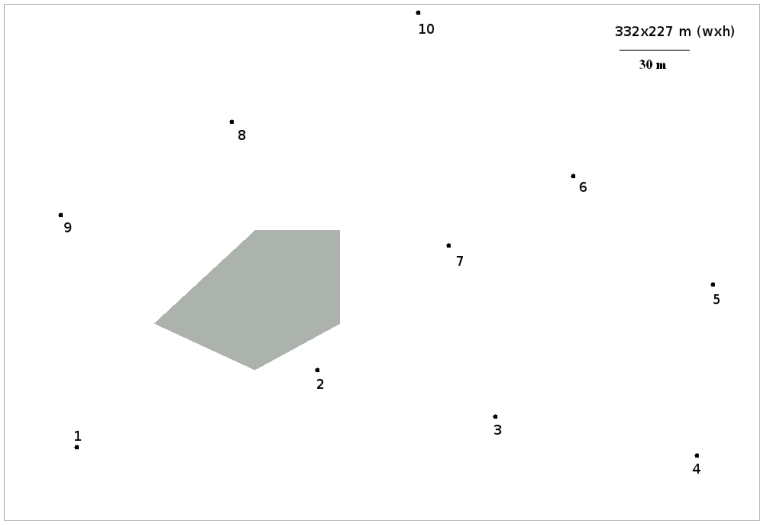


Figure 5.2: Openspace environment.

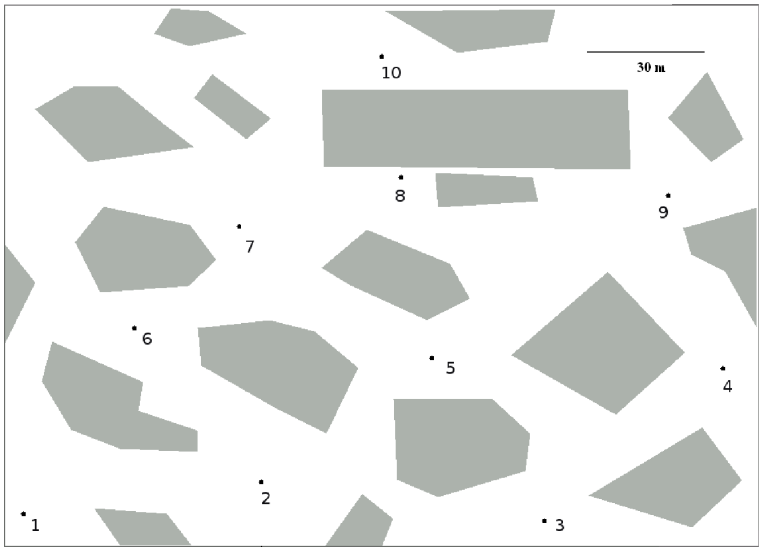


Figure 5.3: Obstacles environment.

Experiments on these two environments have not extensively been done in this work, as for indoor environment, since these environments are very large spaces and require several hours per trial.

5.2 Analyses and comparisons with on-line exploration strategies

Firstly, we analyze results we obtained in an aggregated way. For the office environment, the average solutions quality (over 10 trials) of the framework are reported in Table 5.1. In parentheses, we report the standard deviation. The second column refers to different cell sizes. As we can see in Table 5.1, in the case of number of steps as cost function, the standard deviation is very low (around 3%). So, different solutions cost approximately the same. Indeed, intuitively we can imagine that an environment requires a fixed number of steps to be explored, given a fixed sensor range. In the other case, i.e., traveled distance as cost function, the standard deviation is a little bit larger (around 10%). This is due to the fact that some initial poses require more distance to be traveled in order to explore the environment, e.g., if the robot is in the middle of the environment, it has to visit the left and the right corridors, by covering at least twice one of the two hallways.

		# OF STEPS			DISTANCE		
	SENSOR RANGE	20	25	30	20	25	30
G = 85%	1	26.3 (0.5)	18.8 (0.6)	14.4 (0.5)	674.8 (56.0)	575.4 (49.6)	503.1 (49.4)
	2	25.8 (0.4)	19.9 (0.6)	14.4 (0.5)	670.3 (50.4)	586.3 (44.6)	494.7 (43.6)
	4	25.3 (0.9)	19.6 (0.7)	15.7 (0.7)	657.9 (55.9)	571.3 (38.2)	522.8 (47.2)
G = 90%	1	28.25 (0.5)	20.0 (0.5)	15.8 (0.6)	762.8 (45.2)	636.5 (53.4)	570.5 (57.2)
	2	27.8 (0.4)	21.4 (0.5)	15.4 (0.7)	751.2 (51.1)	666.0 (51.4)	560.4 (50.4)
	4	27.3 (0.9)	21.1 (0.7)	16.8 (0.6)	744.1 (48.8)	638.7 (44.0)	585.9 (46.1)
G = 95%	1	30.0 (0.0)	21.7 (0.5)	17.1 (0.3)	831.4 (56.3)	715.3 (54.8)	648.5 (57.4)
	2	29.7 (0.8)	23.1 (0.6)	16.9 (0.3)	820.9 (47.7)	746.3 (65.1)	633.8 (59.3)
	4	29.5 (0.7)	23.0 (0.5)	18.5 (0.5)	861.7 (76.7)	729.6 (52.8)	663.1 (46.3)

Table 5.1: Results obtained in office environment, with footprint sensor, and as contraint the selection of centroid, looking at different cell sizes.

Now, we analyze the data in the table by showing trends of the quality average of the solutions, according to different initial conditions and parameters. As expected, we can see that cost of the solutions diminishes by increasing the sensor range r, keeping fixed the cell size and the goal percentage G to be discovered, because in one step the robot can perceive more area of the environment. Figure 5.4 and Figure 5.5 show this trend.

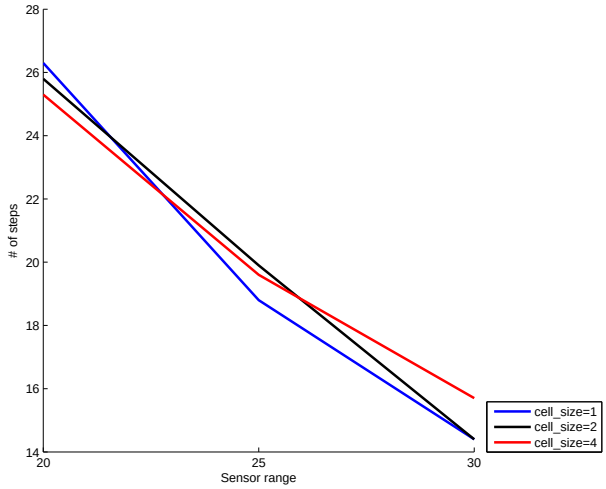


Figure 5.4: Graph that shows the relation between the path cost and the sensor range, looking at different cell size, with # of steps as metric and $G = 0.85$.

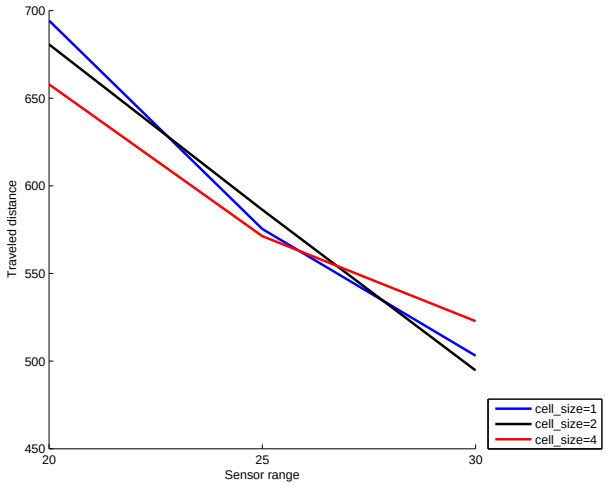


Figure 5.5: Graph that shows the relation between the path cost and the sensor range, looking at different cell size, with traveled distance as metric and $G = 0.85$.

Analyzing how the cost of the solutions scale with the percentage of the environment to be explored, we can see that by increasing the goal percentage G the path cost increases. This result can be intuitively explained by the fact that the robot needs to perform a numer of steps greater or equal to the number of steps performed to map a smaller percentage G of the environment.

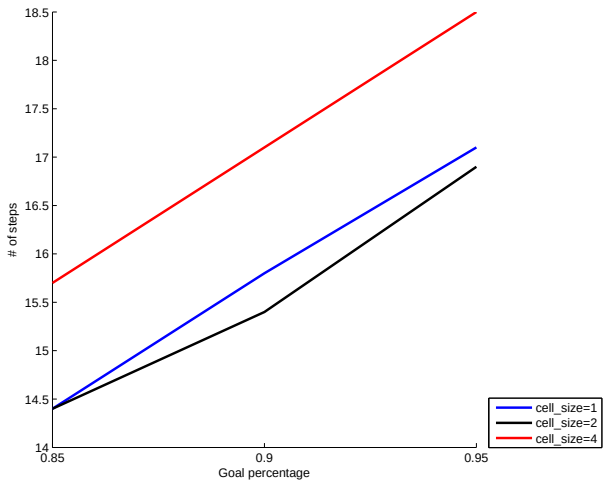


Figure 5.6: Graph that shows a column of the Table 5.1, with # of steps as metric and $r = 30$ m.

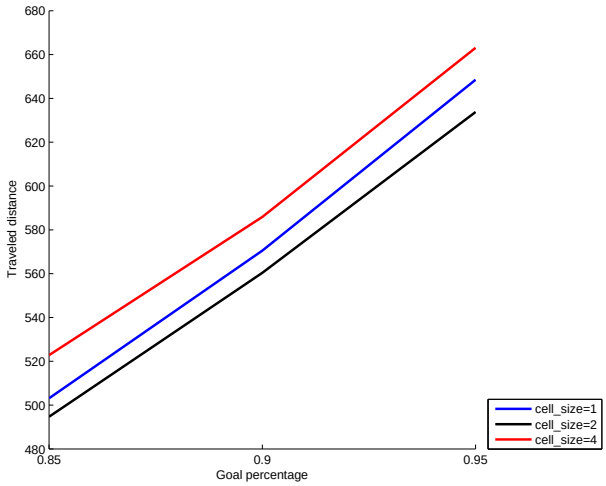


Figure 5.7: Graph that shows a column of the Table 5.1, with traveled distance as metric and $r = 30$ m.

Figure 5.6 and Figure 5.7 show that the number of steps and the traveled distance grow linearly with the percentage of the environment to be explored. This is very interesting, because there is not an explosion of the solution cost if the goal percentage G to be explored is increased.

It is also interesting to analyze how the solution cost changes with respect to cell size: it appears that there is no pattern, even though one may expect that by decreasing the size of a cell (and so increasing the resolution of a map) the cost of the solution grows, because the environment has more details that the robot should take into account when exploring it. This is due

to the fact that computed centroids could be different considering different size of a cell and this could change the boundary cells that the robot can choose and thus the path the robot follows.

Finally, we compare the results in Table 5.1 with results in [2]. In Table 5.2, the average performances (over 10 trials) of the strategies tested in [2] are reported, considering a goal percentage $G = 95\%$. As said before, we did not perform experiments with $r = 10$ and $r = 15$, because they require too much computational time (more than 5 hours per trial).

	# OF STEPS			DISTANCE		
SENSOR RANGE	10	15	20	10	15	20
random	120.6	57.8	33.7	66626.7(2265.8)	3473.0(1158.2)	1990.4(349.3)
greedy	116.0	58.5	37.3	1915.0 (236.6)	2390.0 (874.5)	2359.0 (509.9)
GB-L	127.8	64.8	40.5	1655.4 (203.3)	1121.0 (122.0)	866.9 (74.6)
A-C-G	119.7	54.6	35.2	1673.2 (214.9)	1132.4 (147.3)	909.8 (110.0)

Table 5.2: Results obtained in office environment by different exploration strategies [2].

As we can observe, the framework for determining an optimal exploration path in a specific environment finds (as expected) better solutions compared to those found by real exploration strategies (with $r = 20$ m), considering both number of perceptions and traveled distance as metrics to measure the path cost, proving that our framework actually finds the optimal exploration path. There is very much difference between the solutions, if we consider the number of perceptions. This can be explained by the fact that real exploration strategies usually optimize the traveled distance instead of the number of perceptions. In the case of traveled distance, it is interesting to note that GB-L strategy solution performance is quite close to the performance of the optimal exploration path; instead other exploration strategies performances are very far from the optimal exploration path cost.

5.3 Solution quality vs computational time

It is clear that problem of determining an optimal exploration path has an exponential complexity. So, we analyze how to reduce the computational time, but preserving an acceptable solution quality, i.e., not too different from the optimal one.

As already said in Section 5.1, all experiments have been carried out with the centroid constraint activated, since it required too much time to find a solution without that constraint. This fact can be explained by the huge amount of boundary cells that would create a bunch of nodes that should be checked by the search algorithm. The search algorithm degenerates in a

breadth-first algorithm, because every cell in the same cluster has barely the same heuristic considering both cost functions. So, our reference as optimal solution is the solution obtained with the selection of the centroid activated.

In the following, for the sake of simplicity, we present just a subset of experiments, namely a set of 10 trials (i.e., 10 different initial poses) on the same environment and same initial conditions and parameters, and we show the trend in terms of quality of solutions and computational time for determining an exploration solution. However, we performed several other experiments with different initial conditions and parameters and noticed that the behavior is quite the same, considering the analyzed parameters and constraints.

5.3.1 Initial parameters

Initial poses

Deepening the results obtained in previous trials, it is interesting to note that some initial poses require more computational effort to find a solution. We also took into account the goal percentage G set to 1, namely all the free space of the environment has to be explored, to verify whether it is possible to map all the environment.
Considering each initial pose, in the case of number of steps as cost function, as expected, we can see a regular trend both of computational time and cost of the solutions, that is by increasing the percentage G of the map to be discovered even the computational time and the cost of the solution grow (see Figure 5.8 and Figure 5.9). On x-axis there are initial poses depicted in Figure 5.1, on y-axis computational time to find a solution and number of steps, and each bar represents the percentage of the environment to be explored. If computational time is negative, it means that the search algorithm has terminated, without finding any solution, because of the expiration of the timer, which was set to 18000 seconds; all initial parameters are reported in the caption of the figures.

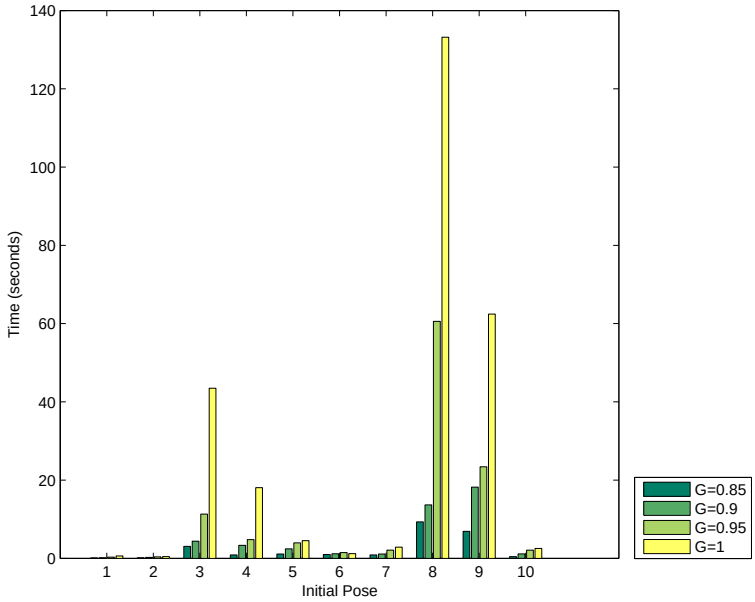


Figure 5.8: Graph bar that shows that some initial poses require more computational time to find the optimal exploration path (if time is negative, it means that no solution has been found within the interval time set). Initial parameters: cost_function=# of steps, cell_size=4, laser_sensor=false, sensor_range=25, pick_centroid=true, interval_timer=18000.

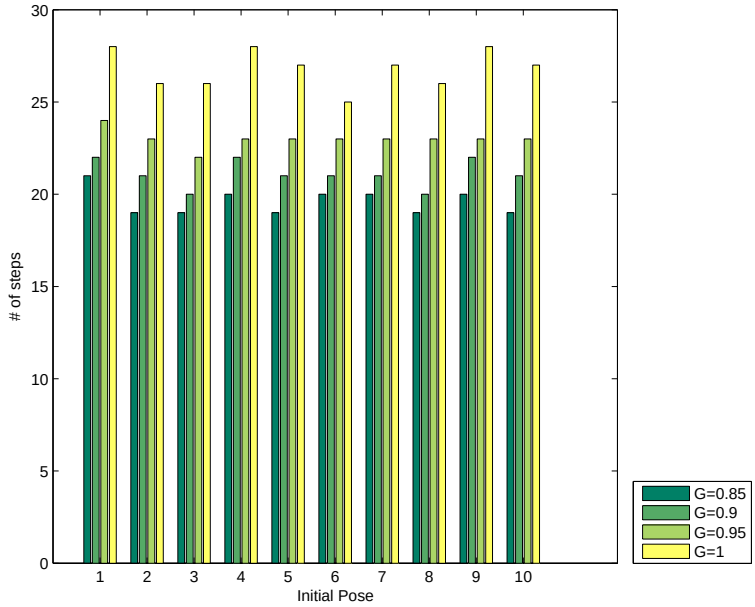


Figure 5.9: Graph bar that shows that number of steps at different goal percentage G. Initial parameters: cost_function=# of steps, cell_size=4, laser_sensor=false, sensor_range=20, pick_centroid=true.

If we compare computational time in different initial poses, we note a huge difference, for example for initial poses 1 and 9. The reason is that in initial pose 1 the search algorithm basically performs a depth-first search, at least at a certain depth level of the search tree, because the robot is constrained to follow a certain path; whereas, in 9, the robot starts in the middle of the environment and can go to the left-most hallway and then to the right-most hallway or viceversa. This fact makes the search tree to exponentially grow, because there are several branches that the search algorithm ought to check.

Figure 5.10 and Figure 5.11 show the computational time and the solutions quality, when traveled distance is considered.

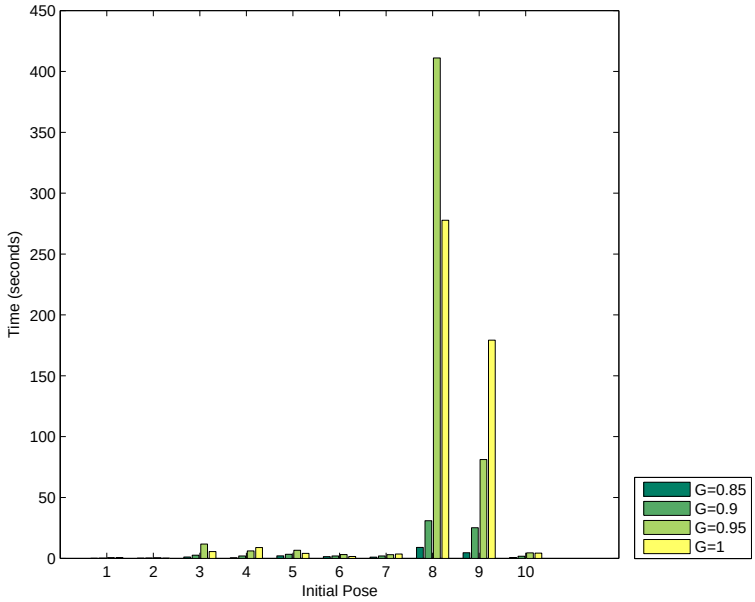


Figure 5.10: Graph bar that shows that some initial poses require more computational time to find the optimal exploration path (if time is negative, it means that no solution has been found within the interval time set). Initial parameters: cost_function=traveled distance, cell_size=4, laser_sensor=false, sensor_range=25, pick_centroid=true, interval_timer=18000.

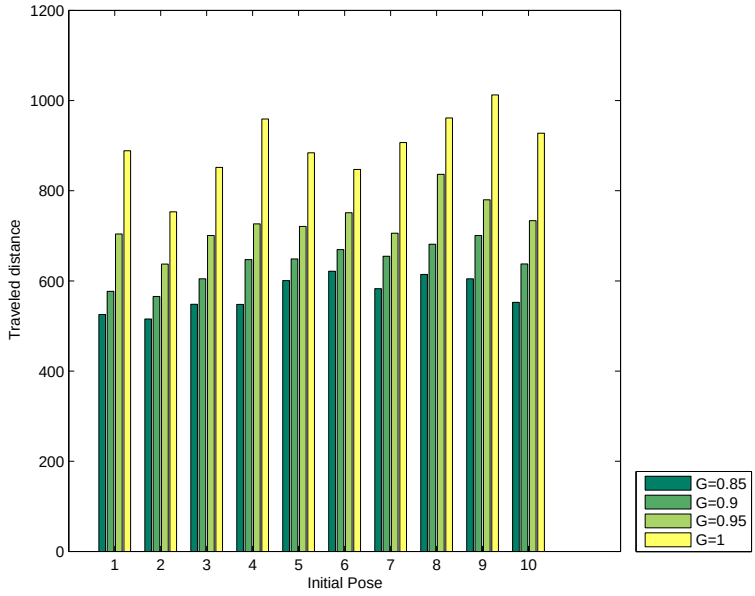


Figure 5.11: Graph bar that shows that number of steps at different goal percentage G. Initial parameters: cost_function=traveled distance, cell_size=4, laser_sensor=false, sensor_range=20, pick_centroid=true.

In the case of traveled distance as cost function, we can observe in the first three goal percentages, i.e., $G = 0.85$, $G = 0.9$ and $G = 0.95$, the computational time trend is the same. However, in some trials it happens that the computational time in case of $G = 1$ is less than the computational time in case of $G = 0.95$ (e.g., initial pose 8). This happens because of the two different heuristics h_2 (used if $G = 1$), which is more precise, and h'_2 (used if $G < 1$), which is more conservative. However, the cost of solutions always increases by augmenting the goal percentage G.
It is relevant to note that, in initial poses 9 and 10, the computational time when number of steps is considered as path cost is less than the computational time when traveled distance is considered as path cost. The reason is that, because of the closed list, in the former case, the search algorithm can discard more nodes thanks to a less tight equivalence among states, whereas in the latter, the search algorithm cannot. It also happens the converse, even if it is less accentuated. For example in initial pose 8, there is a slight difference between computational times when number of steps and traveled distance are considered. The reason is that there is not very much difference among nodes in frontier in terms of evaluation function f (i.e., the sum of current path cost and the cost estimate to the goal), in case of number of steps, whereas in traveled distance, there is: any selected boundary cell is a good view-point over the unexplored area; instead, choosing diverse cells

could be very different in terms of distance.

Figure 5.12 and Figure 5.13 illustrate the number of nodes handled by the search algorithm in the same initial conditions and parameters of Figure 5.8 and Figure 5.10. The amount of nodes processed by the search algorithm is a more objective metric to calculate the computational time, since the latter depends on the hardware on which experiments were run. The trend is the same as the computational time measured in seconds, though.

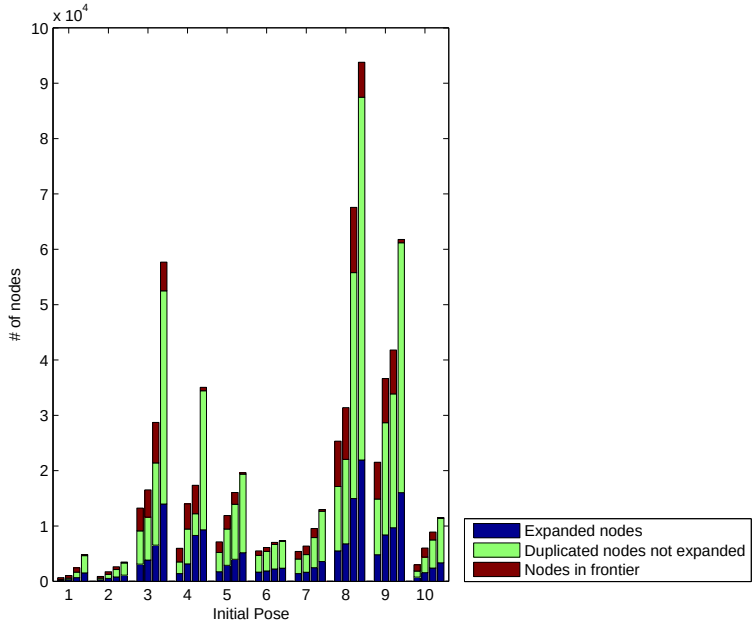


Figure 5.12: Graph bar that shows amount of expanded nodes, discarded duplicated nodes not expanded because equivalent to nodes in closed list and nodes in frontier at the end of the search algorithm, for each initial pose, where each set of 4 bars represent different ascending goal percentage G, i.e., $G = 0.85$, $G = 0.9$, $G = 0.95$ and $G = 1$. Initial parameters: cost_function=# of steps, cell_size=4, laser_sensor=false, sensor_range=25, pick_centroid=true.

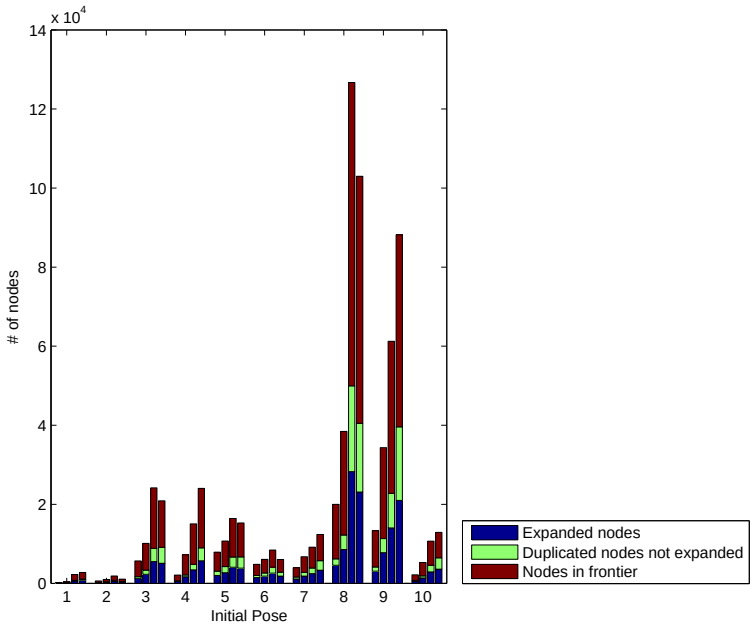


Figure 5.13: Graph bar that shows amount of expanded nodes, discarded duplicated nodes not expanded because equivalent to nodes in closed list and nodes in frontier at the end of the search algorithm, for each initial pose, where each set of 4 bars represent different ascending goal percentage G, i.e., $G = 0.85$, $G = 0.9$, $G = 0.95$ and $G = 1$. Initial parameters: cost_function=traveled distance, cell_size=4, laser_sensor=false, sensor_range=25, pick_centroid=true.

Cell size

By augmenting the resolution of the map, that is by reducing the size of a cell, we have observed that the computational time increases exponentially. This can be explained as the framework has more information to process and more boundary cells to check. Figure 5.14 shows how much time the search algorithm spends for determining the optimal exploration solution, considering different cell sizes. As we can see, computational time in initial poses 5, 6, 8 and 9 is really abated by augmenting the size of a cell.

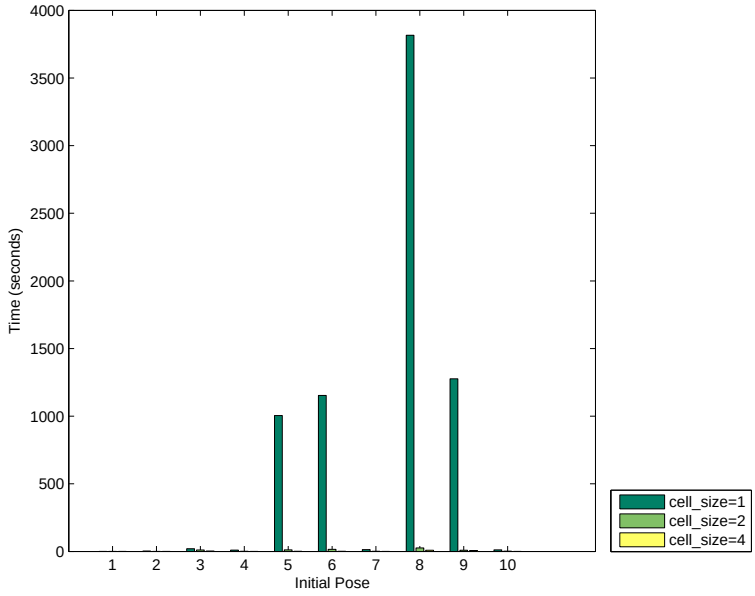


Figure 5.14: Graph bar that shows computational time according to different cell sizes (if time is negative, it means that no solution has been found within the interval time set). Initial parameters: cost_function=# of steps, $G = 0.85$, laser_sensor=false, sensor_range=25, pick_centroid=true, interval_timer=18000 sec.

However, as already assessed before, the quality of the solutions does not change much (see Figure 5.15), because of the discretization of the environment and the computation of centroids, that generate slightly different paths.

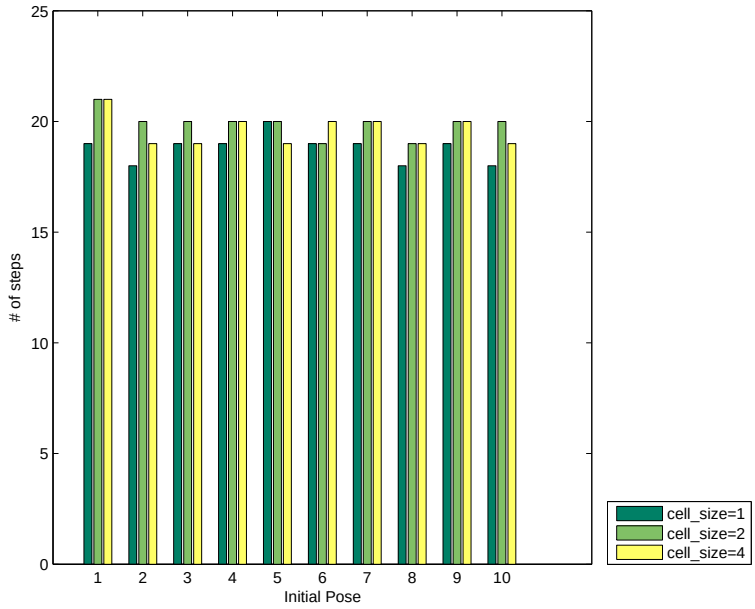


Figure 5.15: Graph bar that shows quality of the solutions according to different cell sizes. Initial parameters: cost_function=# of steps, $G = 0.85$, laser_sensor=false, sensor_range=25, pick_centroid=true, interval_timer=18000 sec.

Figure 5.16 and Figure 5.17 show the computational time and the quality of the solutions, when traveled distance is considered, and the same behavior can be observed.

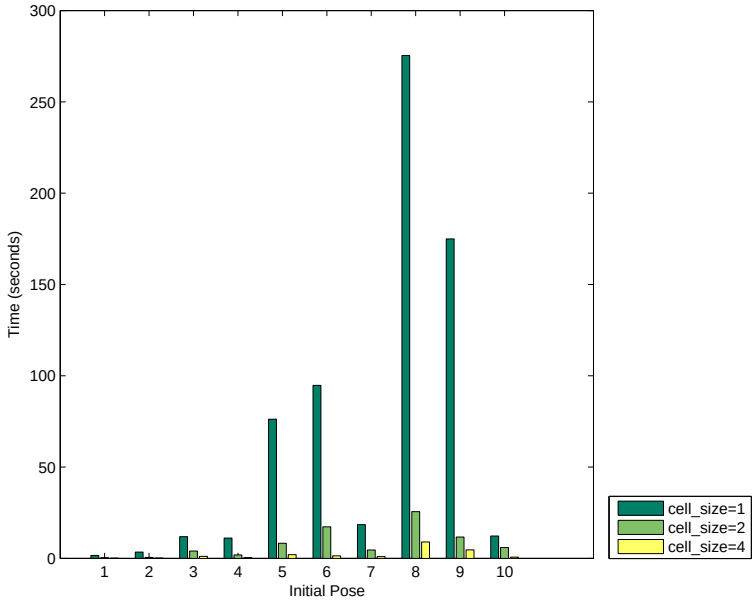


Figure 5.16: Graph bar that shows computational time according to different cell sizes (if time is negative, it means that no solution has been found within the interval time set). Initial parameters: cost_function=traveled distance, $G = 0.85$, laser_sensor=false, sensor_range=25, pick_centroid=true, interval_timer=18000 sec.

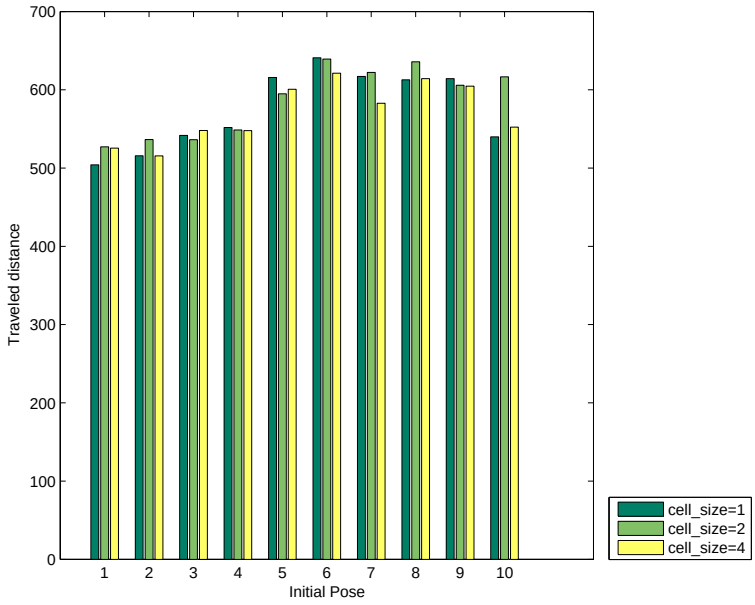


Figure 5.17: Graph bar that shows quality of the solutions according to different cell sizes. Initial parameters: cost_function=traveled distance, $G = 0.85$, laser_sensor=false, sensor_range=25, pick_centroid=true, interval_timer=18000 sec.

5.3.2 Constraints

About constraints, it is intuitive that solution quality remains the same or worsens if constraints are activated, because, by applying them, we take a subset of original boundary set, and so computational time ought to decrease. This is a trade-off between solution quality and computational time. Here, we discuss about how solution quality and computational time change, on the basis of constraints defined in this framework.

Cluster size

First of all, we discuss about how performances change if we limit the selection of boundary cells, by applying the constraint on the minimum size of a cluster. Figure 5.18 shows time spent to compute the exploration solution and Figure 5.19 illustrates the related cost of the exploration path, as number of steps, at different initial poses, in the case the constraint is activated or not.

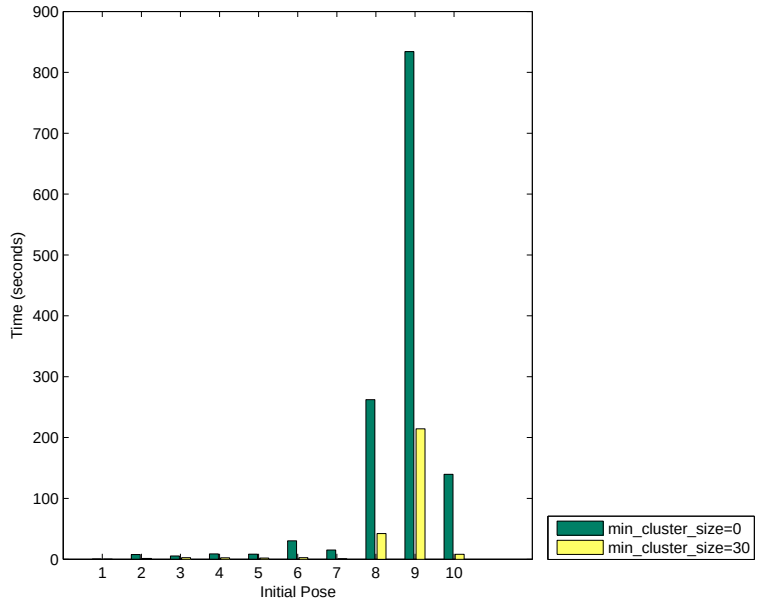


Figure 5.18: Graph bar that shows how computational time changes when minimum cluster of size constraint is activated. Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

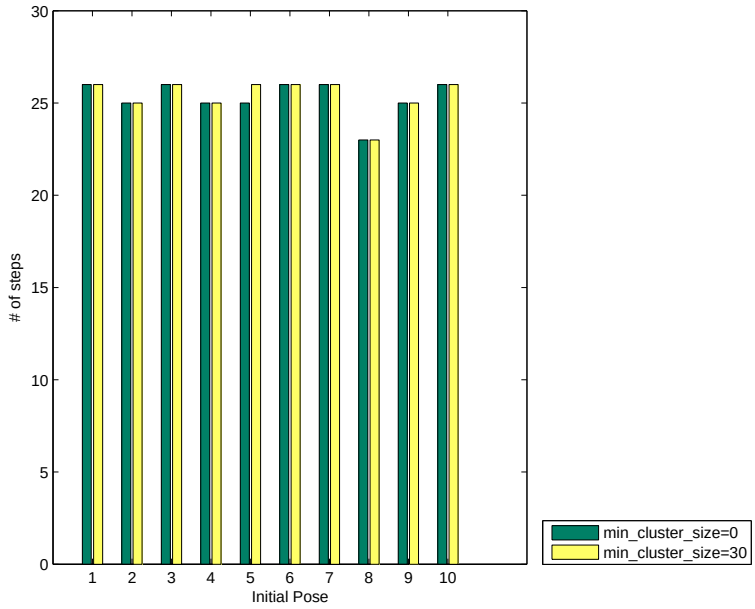


Figure 5.19: Graph bar that shows how quality of solutions changes when minimum cluster of size constraint is activated. Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

As we can see, the number of steps remains the same or worsens a little bit, compared to our optimal solution reference, but the computational time drastically decreases. This is due to the fact that some small areas are not considered anymore, leading to a cut of the branch factor of the search tree.

Figure 5.18 and Figure 5.19 show how the constraint on the size of clusters influences time spent to compute the exploration solution and the related cost of the exploration path, considering traveled distance.

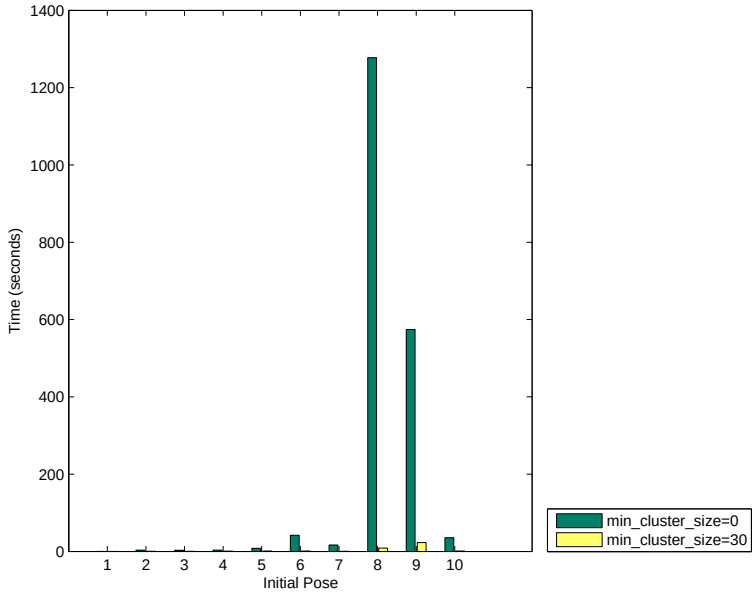


Figure 5.20: Graph bar that shows how computational time changes when minimum cluster of size constraint is activated. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

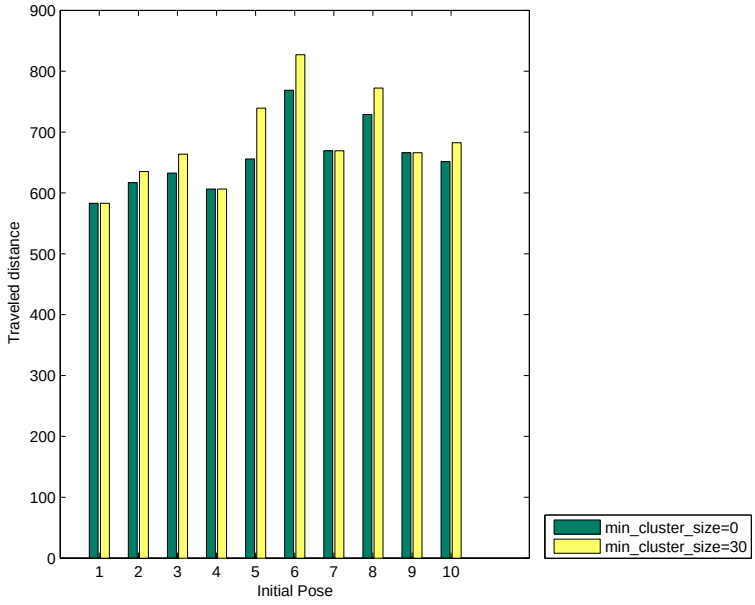


Figure 5.21: Graph bar that shows how quality of solutions changes when minimum cluster of size constraint is activated. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

Also in this case, we can observe that the time drastically decreases and

the solution remains almost the same. The fact that the solution is barely equal to the optimal solution reference is due to the nature of this constraint, namely with high probability, the constraint discards boundary cells from where the robot can perceive less, because those cells are near to a small edge of the unexplored area. The issue is how to tune this parameter, which is heavily subject to the environment and the sensor range of the robot. If it is set too high, then there are high chances that the search algorithm cannot find any solution, whereas if it is set too low, the gain from this constraint is lessened.

Safeness

Now, we analyze how performance and computational time change, if constraints on selection of single boundary cells are applied.
About safeness, i.e., selected boundary cells ought to have a distance l from obstacles, we have the same trend as the constraint on the size of clusters, as Figure 5.22, Figure 5.23, Figure 5.24, and Figure 5.25 show.

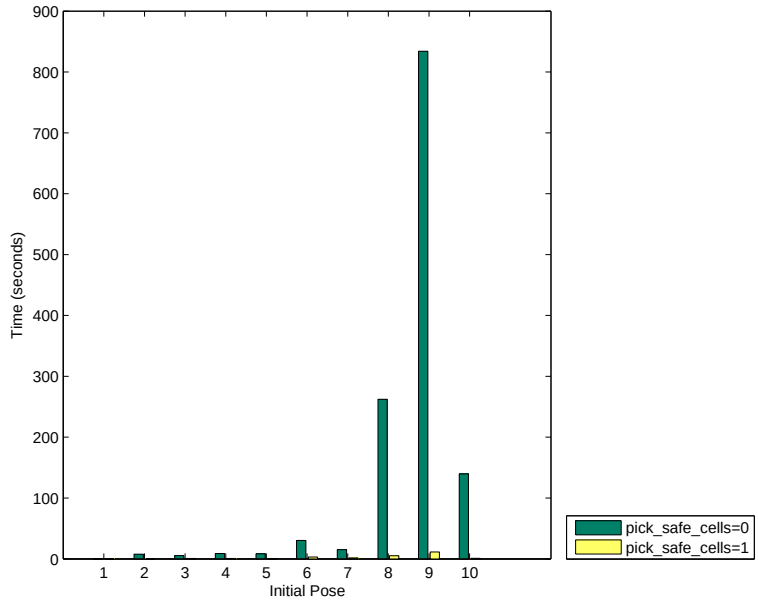


Figure 5.22: Graph bar that shows how computational time changes when safeness constraint is activated. Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $l = 16$.

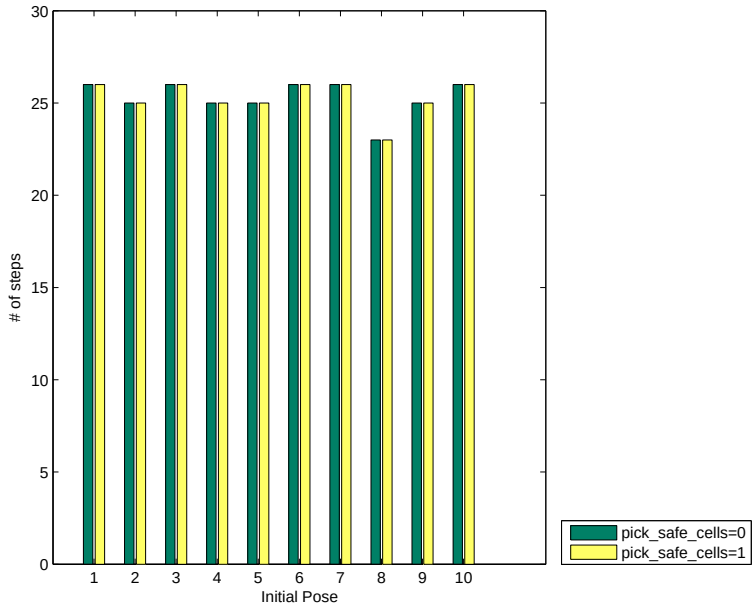


Figure 5.23: Graph bar that shows how quality of solutions changes when safeness constraint is activated. Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

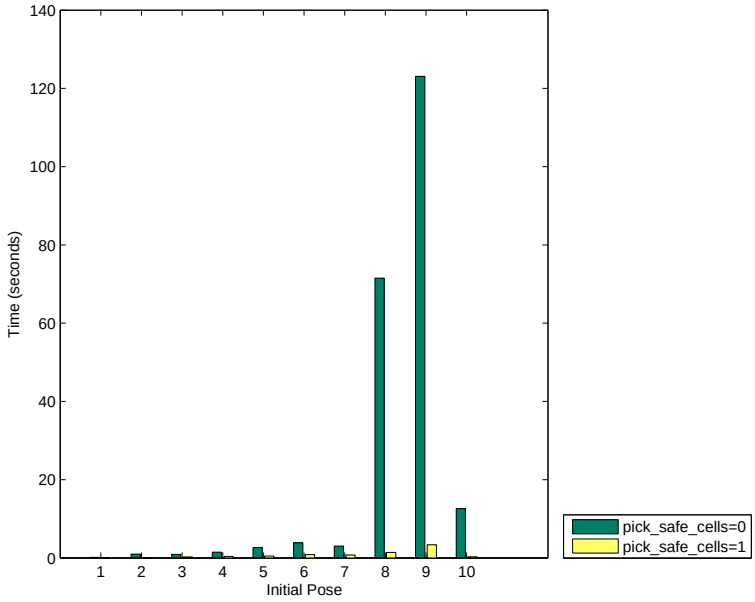


Figure 5.24: Graph bar that shows how computational time changes when safeness constraint is activated. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $l = 16$.

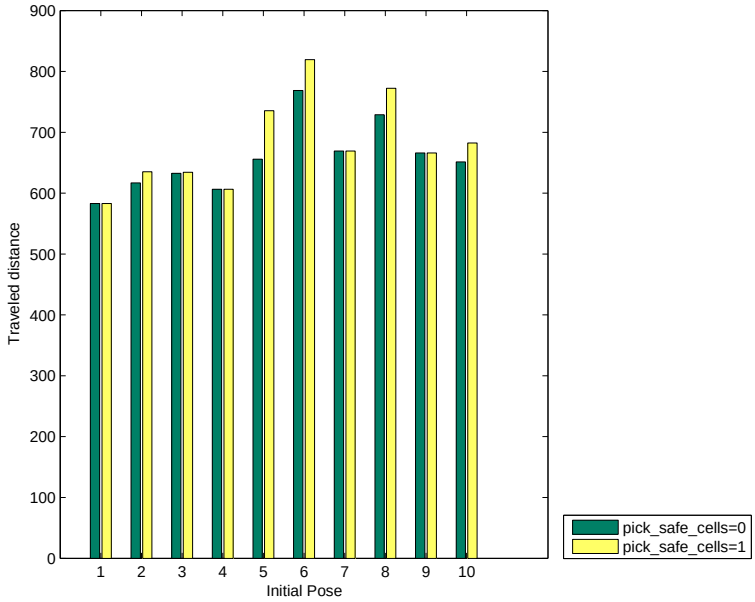


Figure 5.25: Graph bar that shows how quality of solutions changes when safeness constraint is activated. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

The solution is more or less equal to the optimal solution reference, because of the selection of the centroid activated, which usually calculates a middle point in a cluster that is adjacent to an unexplored area. So, boundary cells are most likely distant from obstacles.

Nearness

About nearness, i.e., select boundary cells not too far from the current robot pose, Figure 5.26 and Figure 5.27 show computational time for determining the solution and number of steps of the solution path, when nearness constraint is activated.

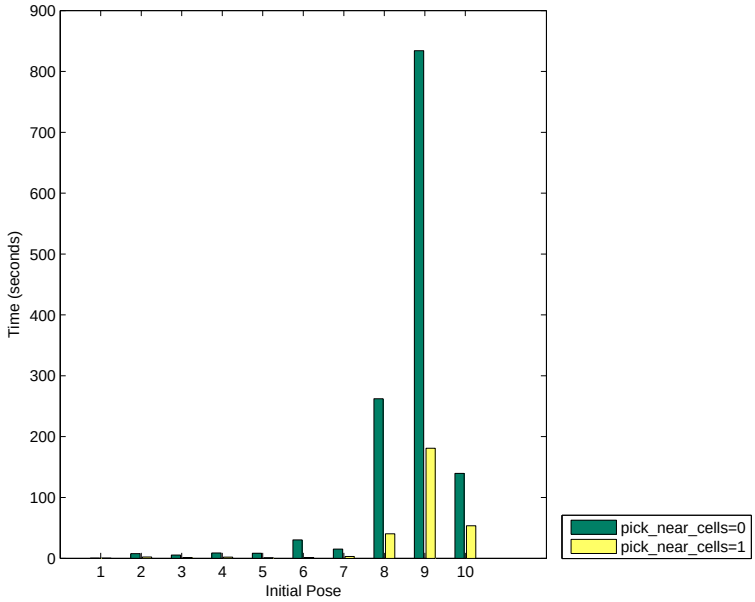


Figure 5.26: Graph bar that shows how computational time changes when nearness constraint is activated. Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $m = 16$.

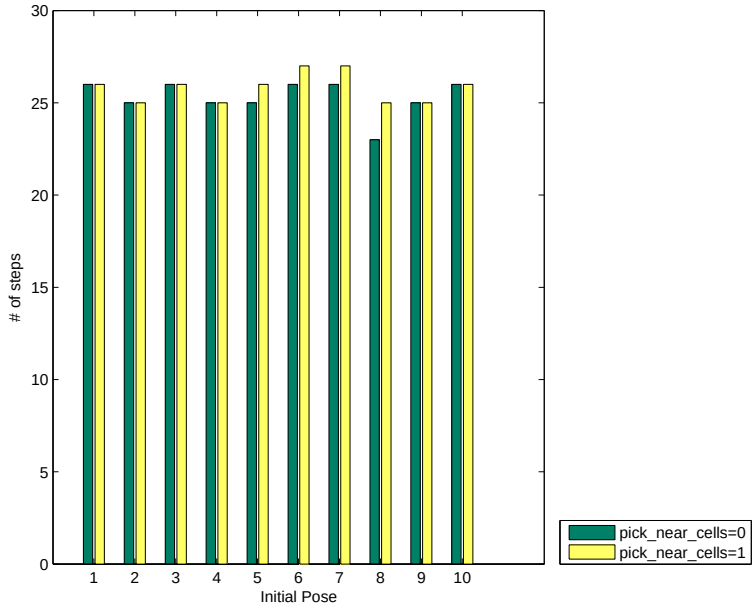


Figure 5.27: Graph bar that shows how quality of solutions changes when nearness constraint is activated. Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $m = 16$.

We can see that computational time remarkably diminishes. However,

with this constraint the solution worsens a little bit more compared to other constraints analyzed above. This is due to the fact that, considering number of steps and this specific environment, it is sometimes better to go further and then to go back to cover some small unexplored area. Instead, this constraint forces the robot to go to boundary cells close to its pose, fostering, perhaps, some small unexplored areas. But, this could be not necessary, because the goal G is set to 85%.

Figure 5.28 and Figure 5.29 show computational time for finding the solution and the traveled distance of the solution path.

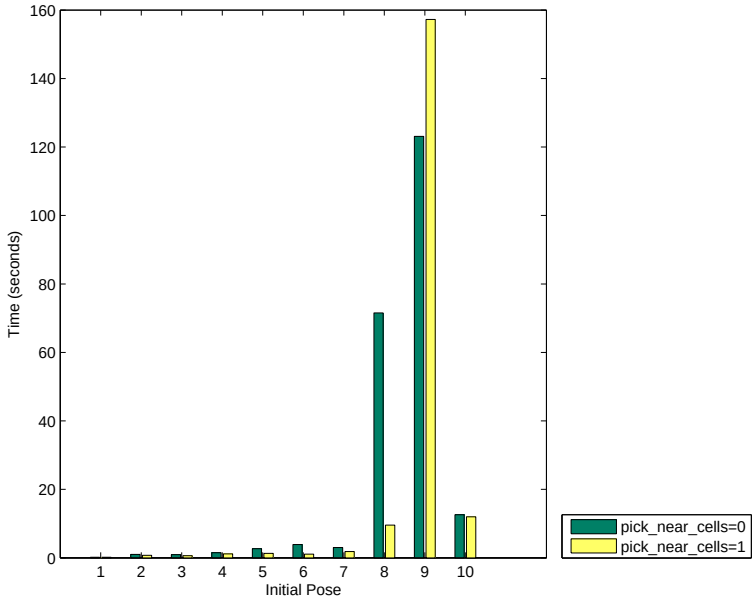


Figure 5.28: Graph bar that shows how computational time changes when nearness constraint is activated. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $m = 16$.

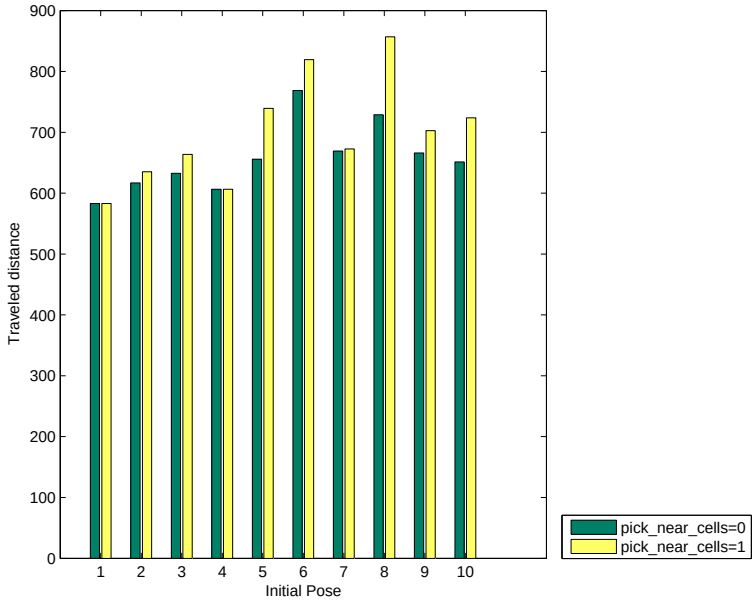


Figure 5.29: Graph bar that shows how quality of solutions changes when nearness constraint is activated. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $m = 16$.

In this case, we can see that in position 9 the time for finding a solution increases if the constraint is activated. The reason is that the search algorithm, since some paths are pruned by the constraint, has to explore other paths, where there are several branches with similar heuristic values, and so the search algorithm must expand more nodes. As expected, the quality of the solution either remains the same or worsens. We can observe that the difference of the solutions in the case of path cost as number of steps is less accentuated with respect to the case of path cost as distance traveled, because, in the former case, a change in the path does not influence very much the cost, since only number of steps is important, but not the actual path followed, whereas, in the latter, a change in the path really affects the cost of the solution.

5.3.3 Further experiments

Additionally, we performed some further experiments, to test some other initial conditions and parameters and further verify the behavior of the proposed framework.

Search algorithms

We compared the two search algorithms implemented, namely A* and branch and bound search algorithms. Unfortunately, we do not have results of branch and bound algorithm applied to the case of cost function as number of steps, since it required too much memory and time. This is due to the fact that the first initial solution found by branch and bound algorithm and the heuristic h_1 is not good enough to cut the branches of the tree. So, branch and bound search algorithm degenerates to a breadth-first search algorithm. Instead, for the case of cost function as traveled distance, we can see that it performs very well (see Figure 5.30).

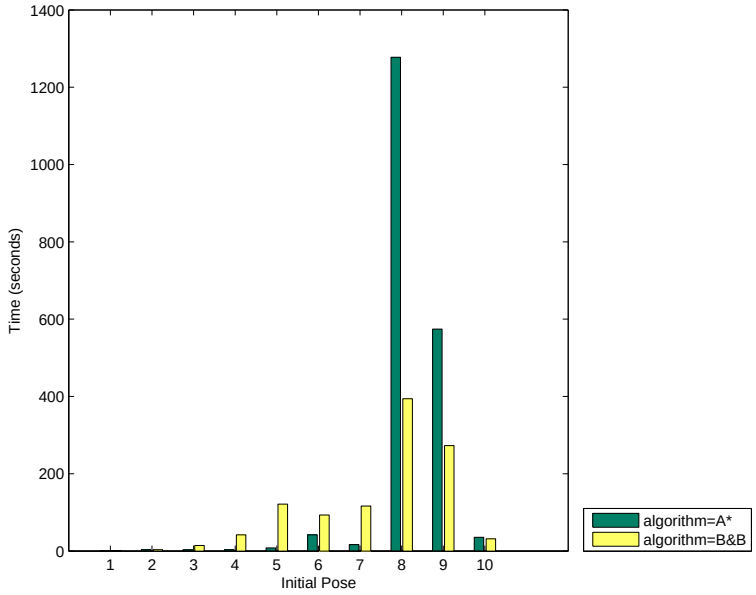


Figure 5.30: Graph bar that shows computational time spent by A* and branch and bound search algorithm. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $l = 16$.

In most cases, branch and bound algorithm performs slightly worse than A* algorithm, but there is not very much difference. Nevertheless, there are 2 cases where branch and bound performs really better than A* (i.e., in initial poses 8 and 9). This is due to the fact that branch and bound manages to find a solution close to the actual optimal solution at the beginning and, thus, it can prune the search tree, whereas A* has to check more paths because of heuristic h_2, which is conservative.
Figure 5.31 shows that in all trials branch and bound algorithm processes more nodes than A* algorithm. Still, it could happen that the computational time required by branch and bound to find a solution might be lower,

because the complexity of different operations on node, that is expanding nodes, checking if a node is duplicated or ordering nodes in the frontier, is different. As predictable, Figure 5.32 illustrates that solutions found by the two algorithms cost the same in terms of traveled distance.

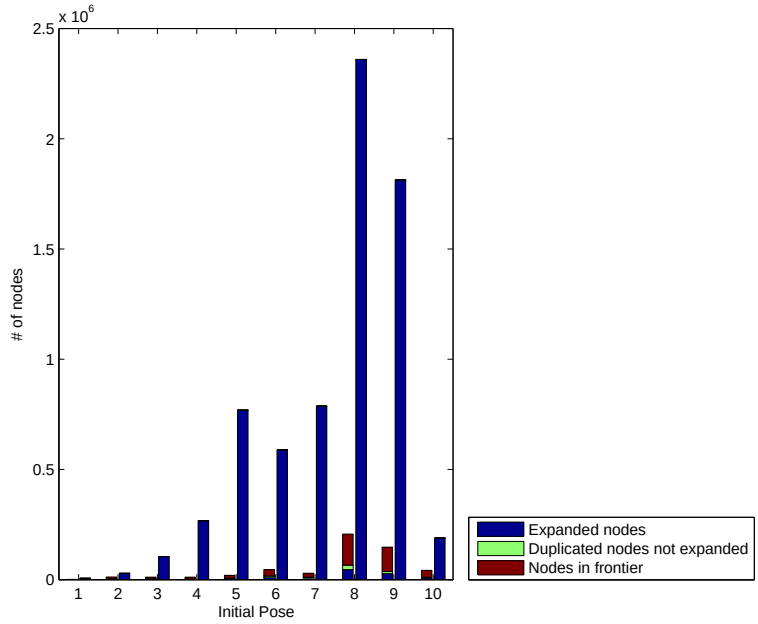


Figure 5.31: Graph bar that shows number of nodes processed by A* and branch and bound search algorithm. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $l = 16$.

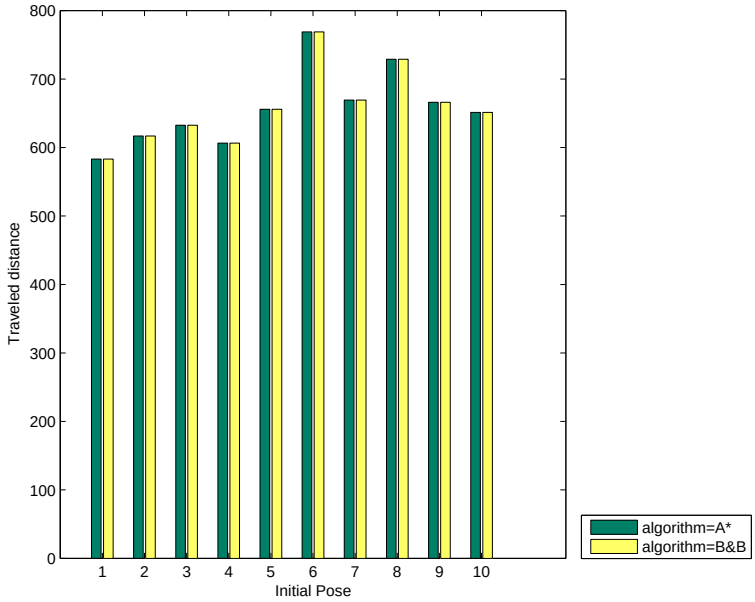


Figure 5.32: Graph bar that compares solutions between A* and branch and bound. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

Perception models

About the perception model, we performed some experiments with realistic laser sensor model and compared them with results obtained by using footprint sensor model. Figure 5.33 displays that computational time is very large, in the case of laser sensor and number of steps as cost function, in initial pose 3, compared to the case of footprint sensor. As a matter of fact, the search algorithm usually takes more time to find a solution in the case of laser sensor as perception model, because it has to process more nodes due to small unexplored areasderiving from shadows cones of obstacles, that generates several boundary cells. Also the steps to achieve a perception action is more complicated. The converse happens in initial poses 8 and 9, where computational time in the case of laser sensor is slightly lower than the other case. The reason is that, in these initial poses, in the case of footprint sensor, nodes are more or less equivalent in terms of f, while in the other case, there are several nodes with a very high f, because these nodes refer to states where the robot cannot perceive very much about the environment (small unexplored areas, near the wall). On the basis of the quality in terms of heuristics of these nodes generated by those boundary cells, the search algorithm can be faster or slower.

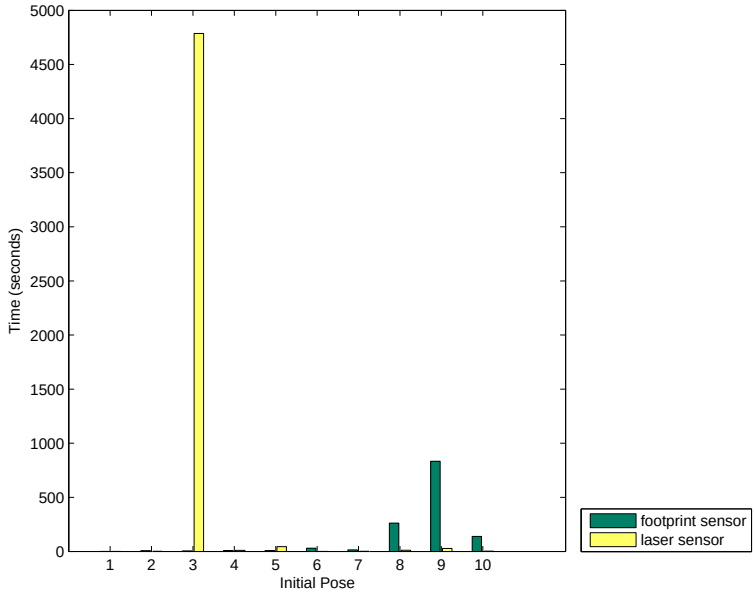


Figure 5.33: Graph bar that shows how computational time changes if footprint or laser sensor is used. Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $l = 16$.

Furthermore, we can see in Figure 5.34 that also the trend of the solution over different initial poses has not a pre-defined pattern.

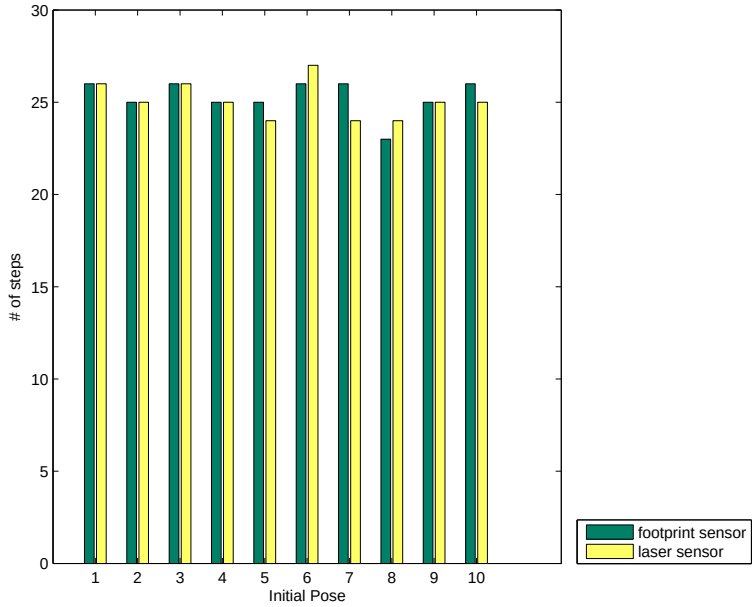


Figure 5.34: Graph bar that compares solutions found used footprint sensor and laser sensor. Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

This non-regular trend is determined by the computation of the centroid and the number of boundary cells generated. In fact, if laser sensor is employed, there could be more boundary cells, because laser sensor can leave some holes in the perceived area, due to its realism, while footprint sensor does not. So the search algorithm has a superset of the boundary to consider.
Instead, in the case of distance as cost function, we can observe that, when laser sensor is the perception model, computational time is always greater than computational time in the case of footprint sensor model (see Figure 5.35).

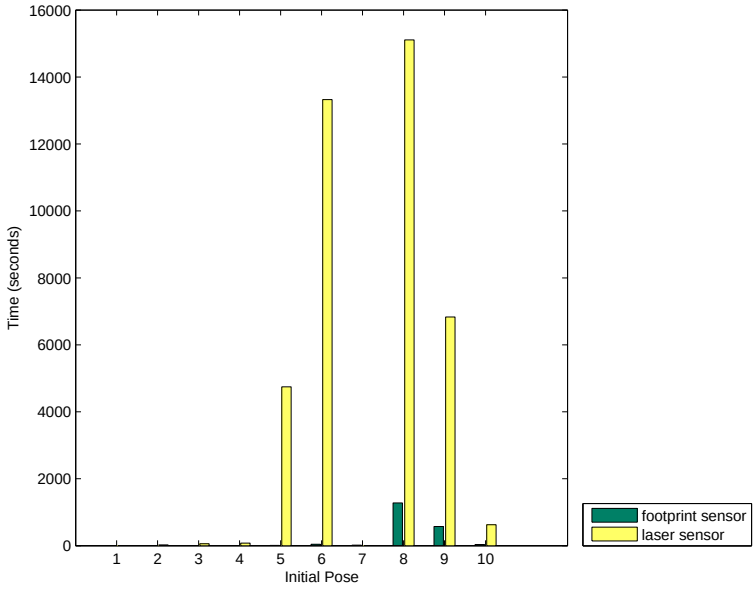


Figure 5.35: Graph bar that shows how computational time changes if footprint or laser sensor is used. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true, $l = 16$.

The reason is that perception with laser sensor generates several boundary cells that are very close, but not adjacent, to each other. So, when traveled distance is considered, the search algorithm has to check several nodes, even if there is not any gain in terms of perceived area, because the function f that evaluates a node just considers the traveled distance. In the former case, it is very relevant whether the robot has perceived more, because of its definition of heuristic.
The quality of solutions has not a regular trend (see Figure 5.36), though, because of the same reasonings made when number of steps was considered.

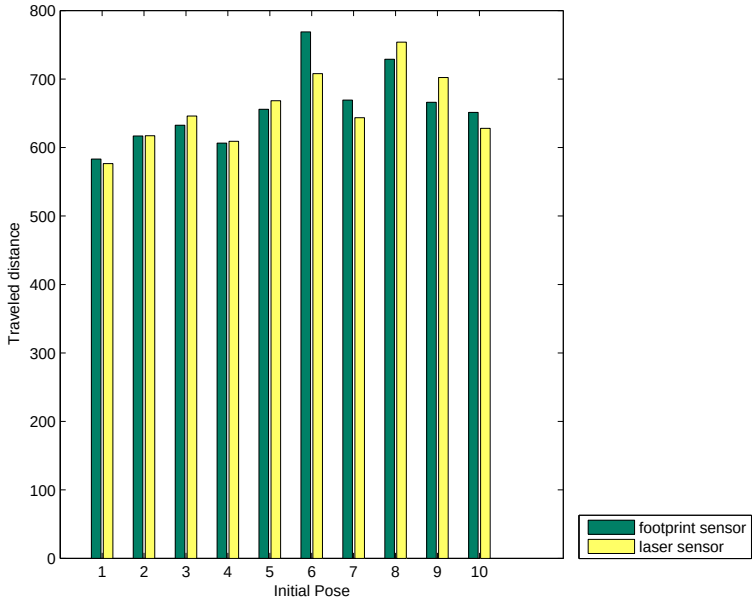


Figure 5.36: Graph bar that compares solutions found used footprint sensor and laser sensor. Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=20, pick_centroid=true.

5.4 Other experiments

Here, we show some preliminary results obtained by performing experiments on the other two environments, namely the openspace and the obstacles environments.

If the cost of the solution is 0, then it means that the search algorithm did not have any successor node to expand, and so the search algorithm terminated, because the frontier was empty.

5.4.1 Openspace environment

In case of openspace environment, the experiments were carried out with the constraint on the clusters size activated, beside the selection of centroids, because, otherwise, the search algorithm could not find any solution with the interval time. This environment is more complex compared to the indoor environment, because there is more area to explore and there is no hint provided by obstacles for the selection of centroids as in the case of indoor environment. However, intuitively, the constraint on cluster size does not affect optimality in such an environment, because the faster way to explore the environment is to go to the largest unexplored area.

Figure 5.37 and Figure 5.38 show computational time and quality of solutions, when number of steps is considered as cost function.

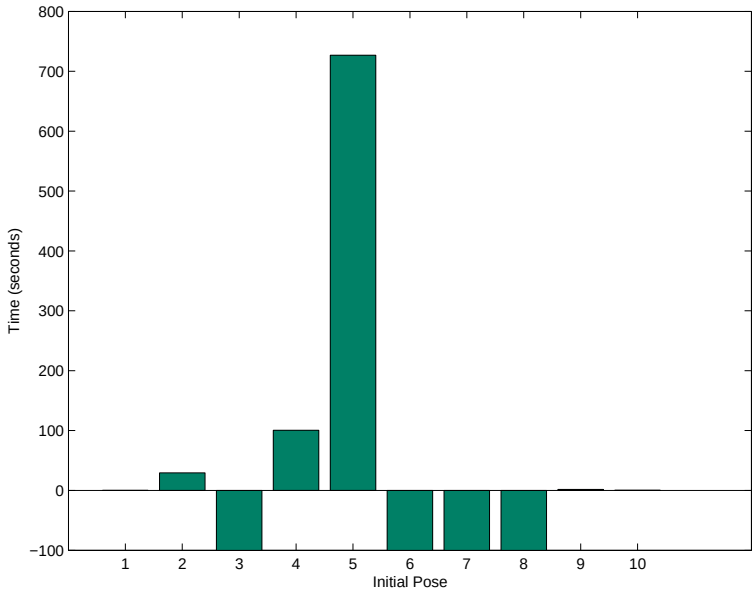


Figure 5.37: Graph bar that shows how computational time in openspace environment (if time is negative, it means that no solution has been found within the interval time set). Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=30, pick_centroid=true, min_cluster_size=100.

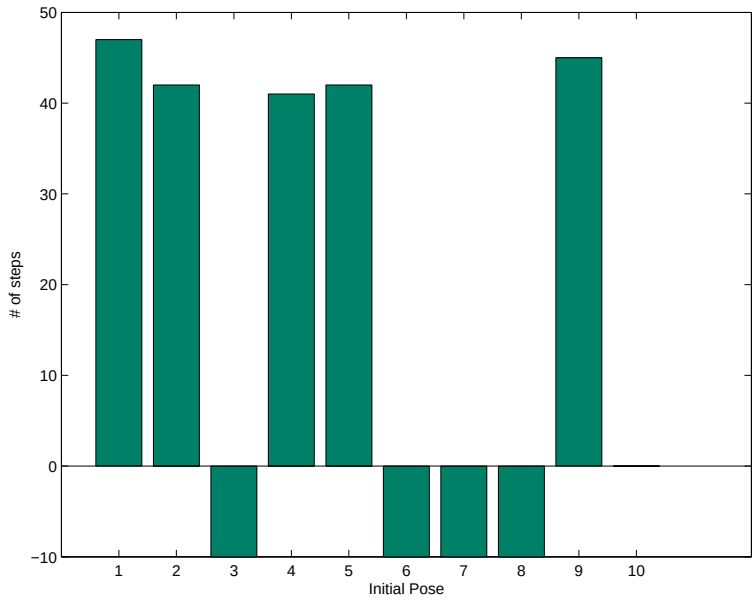


Figure 5.38: Graph bar that shows the quality of solutions in openspace environment (if value is negative, it means that no solution has been found within the interval time set; if it is 0, then no solution has been found because all nodes are not the goal). Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=30, pick_centroid=true, min_cluster_size=100.

As we can see, the search algorithm was able to find a solution in initial poses where there are some obstacles that could guide the robot (i.e., 1, 2, 3, 4, 9, 10), while it was terminated when the initial poses were in the middle of the environment (i.e., 3, 6, 7, 8). The solutions are quite close to each other, with the average of solutions found of 43.4 and standard deviation of 2.51.

Figure 5.39 and Figure 5.40 show computational time and quality of solutions, when traveled distance is considered as cost function.

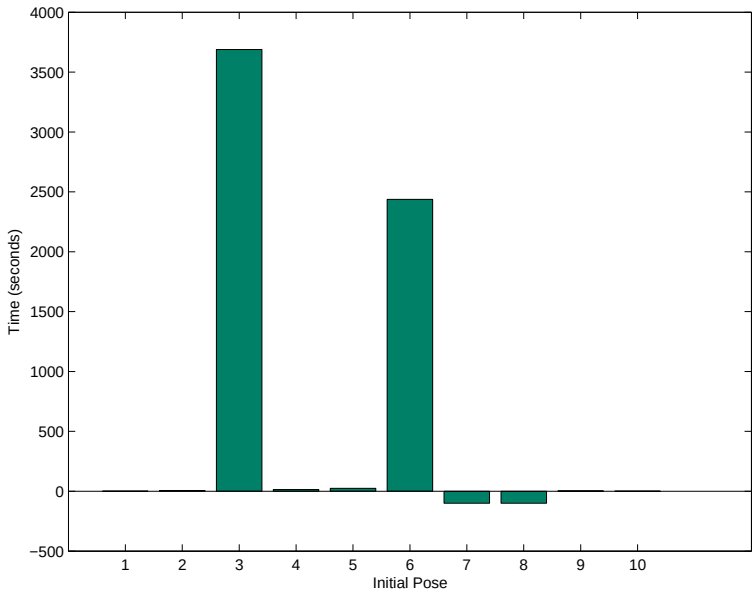


Figure 5.39: Graph bar that shows how computational time in openspace environment (if time is negative, it means that no solution has been found within the interval time set). Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=30, pick_centroid=true, min_cluster_size=100.

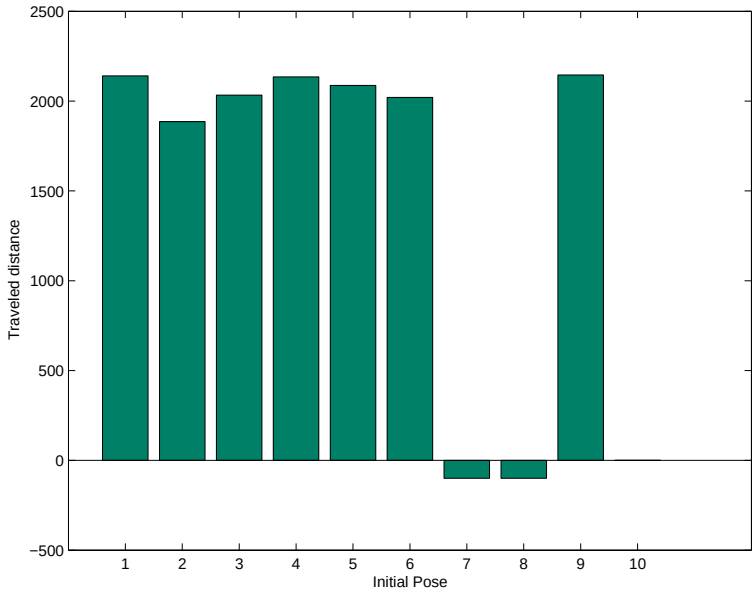


Figure 5.40: Graph bar that shows the quality of solutions in openspace environment (if value is negative, it means that no solution has been found within the interval time set; if it is 0, then no solution has been found because all nodes are not the goal). Initial parameters: cost_function=traveled distance, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=30, pick_centroid=true, min_cluster_size=100.

In this case, just in initial poses 7 and 8 the search algorithm was not able to find any solution. So, the search algorithm was able to determine optimal exploration paths in more trials with respect to the former case. Moreover, the search algorithm in initial pose 10 was not able to find a solution because there was not any successor nodes left in the frontier. The constraint on the cluster size discarded all possible successors. This is due to the fact that in the former case boundary cells in a given state do not differ very much from each other in terms of function f, while in the latter does. Also here, the cost of the solutions is close to each other, with an average of solutions found 2063.9 and a standard deviation of 43.4.

In Table 5.3, the average performances (over 10 trials) of exploration strategies in openspace environment are reported. Although these results are not comparable with the results obtained by our framework, we can assert that the solutions found are reasonable in terms of cost. As a matter of fact, there is very much difference between the average costs of our solutions and the average costs of solutions of real exploration strategies.

		# OF STEPS		DISTANCE	
	SENSOR RANGE	15	20	15	20
G = 85%	random	164.0	95.4	9241.0 (1824.9)	6503.0 (1733.2)
	greedy	163.5	91.7	3884.9 (576.0)	2900.8 (357.0)
	GB-L	185.3	110.1	4150.4 (436.9)	3061.9 (427.5)
	A-C-G	164.9	96.3	3410.3 (324.9)	2441.9 (180.3)
G = 90%	random	173.9	101.1	10700.1 (2299.4)	6672.5 (834.2)
	greedy	177.9	103.9	4315.9 (599.1)	3404.2 (522.6)
	GB-L	197.3	115.7	4332.0 (417.2)	3119.4 (208.5)
	A-C-G	182.5	104.5	3840.7 (344.2)	2164.7 (282.2)
G = 95%	random	191.3	107.8	11749.7 (1462.7)	7255.3 (1052.7)
	greedy	194.7	111.9	4841.8 (559.4)	3737.3 (418.4)
	GB-L	208.8	122.7	4767.8 (474.2)	3490.9 (327.6)
	A-C-G	197.3	111.2	4312.5 (284.4)	3044.1 (341.1)

Table 5.3: Results obtained in openspace environment by different exploration strategies [2].

5.4.2 Obstacles environment

The obstacles environment is more and more complex compared to the other two environments. The reason is that there are several obstacles that determine several boundary cells and thus several paths, even if the selection of centroids is activated. So, the search tree has plenty of branches, that the search algorithm ought to take into account. As a matter of fact, we had to impose other constraints, namely the constraint on the cluster size, and nearness, in order to have solutions in a reasonable time.

Figure 5.41 and Figure 5.42 show computational time and quality of solutions when number of steps is considered. Also here, we have some

initial poses where the search algorithm spent more effort to find a solution.

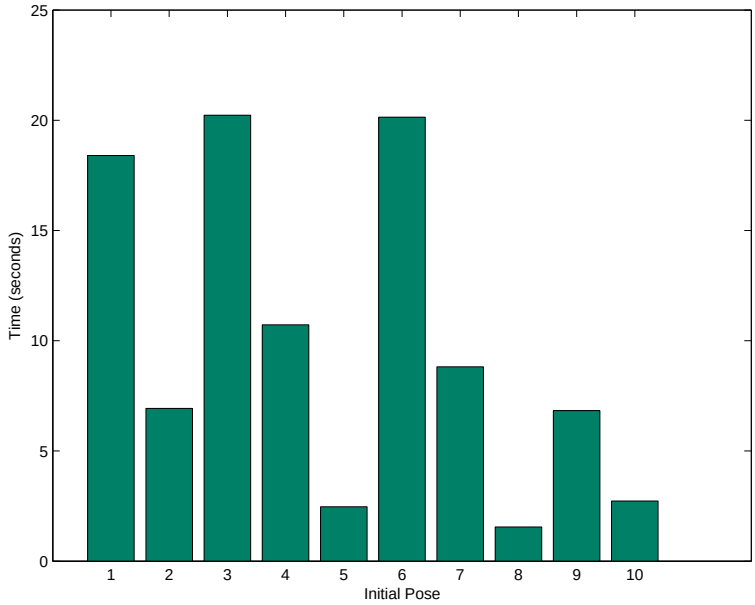


Figure 5.41: Graph bar that shows how computational time in obstacles environment (if time is negative, it means that no solution has been found within the interval time set). Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=30, pick_centroid=true, min_cluster_size=16, pick_near_cells=1, m=60.

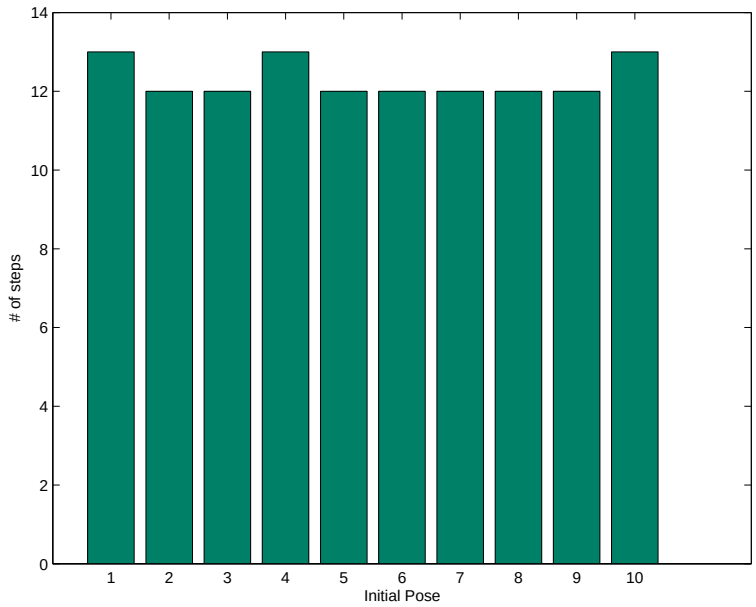


Figure 5.42: Graph bar that shows the quality of solutions in obstacles environment (if value is negative, it means that no solution has been found within the interval time set; if it is 0, then no solution has been found because all nodes were not the goal). Initial parameters: cost_function=# of steps, cell_size=4, $G = 0.85$, laser_sensor=false, sensor_range=30, pick_centroid=true, min_cluster_size=16, pick_near_cells=1, m=60.

The cost of the solutions is more or less the same in all poses, as expected. The average of solutions, when number of steps is considered, is 12.3 steps with a standard deviation of 0.5.

Figure 5.43 and Figure 5.44 show computational time and quality of the solutions when traveled distance is considered. In this case, we can observe the same behavior as before.

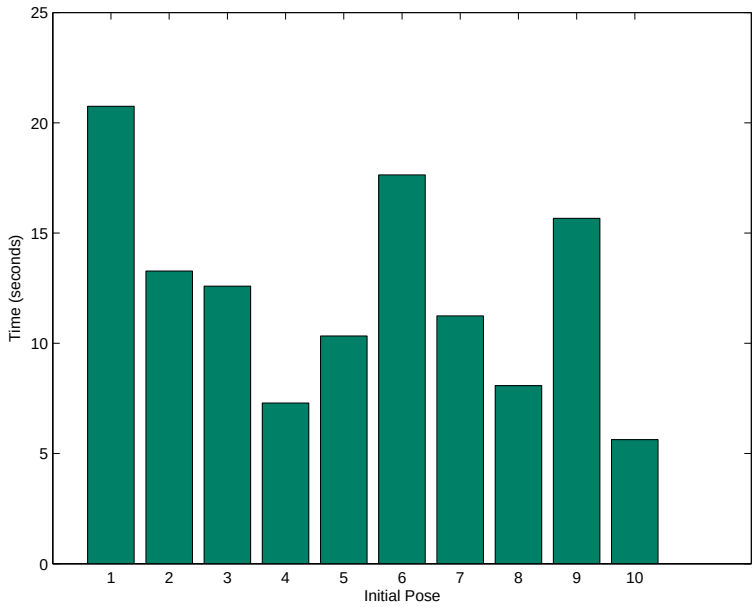


Figure 5.43: Graph bar that shows how computational time in obstacles environment (if time is negative, it means that no solution has been found within the interval time set). Initial parameters: cost_function=traveled distance, cell_size=2, $G = 0.85$, laser_sensor=false, sensor_range=30, pick_centroid=true, min_cluster_size=16, pick_near_cells=1, m=60.

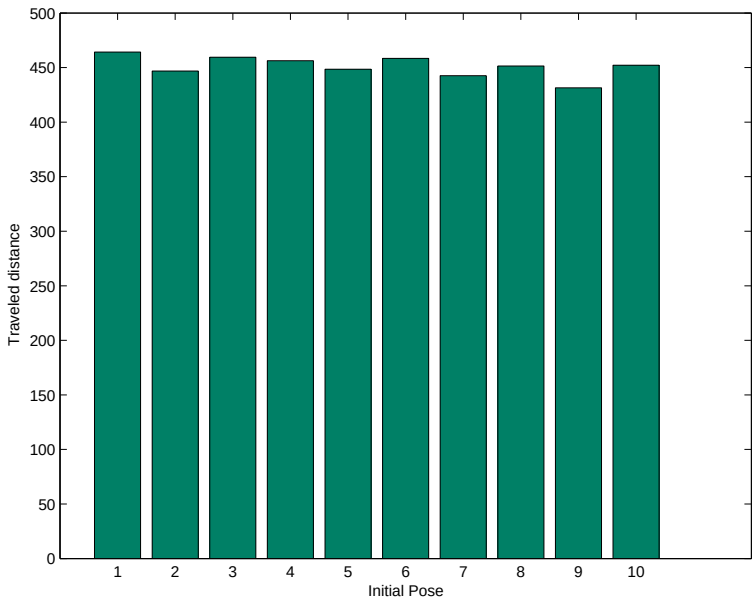


Figure 5.44: Graph bar that shows the quality of solutions in obstacles environment (if value is negative, it means that no solution has been found within the interval time set; if it is 0, then no solution has been found because all nodes were not the goal). Initial parameters: cost_function=traveled distance, cell_size=2, $G = 0.85$, laser_sensor=false, sensor_range=30, pick_centroid=true, min_cluster_size=16, pick_near_cells=1, m=60.

The average of solutions, when traveled distance is considered, is 451.1 m with a standard deviation of 9.5 m.

In Table 5.4, the average performances (over 10 trials) of exploration strategies in obstacles environment are reported. These results cannot be compared with the results obtained by our framework, but we can see that, as expected, the average costs of our framework is less than the average performances of real exploration strategies, since we used a sensor range greater than the ones used in [2].

		# of steps		distance	
	sensor range	15	20	15	20
G = 85%	random	50.2	27.6	2671.9 (1222.0)	1403.9 (674.9)
	greedy	44.6	27.2	816.7 (175.4)	820.4 (450.4)
	GB-L	46.9	26.4	688.6 (86.1)	509.2 (64.6)
	A-C-G	45.9	27.0	708.1 (81.1)	568.7 (111.0)
G = 90%	random	53.1	29.8	2854.6 (831.7)	1688.1 (603.1)
	greedy	49.2	29.3	941.9 (279.0)	934.9 (513.6)
	GB-L	49.4	28.5	730.3 (84.7)	549.4 (85.6)
	A-C-G	47.5	28.6	737.7 (102.4)	615.9 (149.0)
G = 95%	random	56.5	31.0	2880.0 (930.5)	1735.6 (791.6)
	greedy	52.8	30.8	1027.4 (408.4)	985.2 (495.5)
	GB-L	52.3	29.7	767.4 (83.0)	576.6 (72.2)
	A-C-G	51.0	30.9	777.7 (81.7)	692.3 (194.5)

Table 5.4: Results obtained in obstacle environment by different exploration strategies [2].

Chapter 6

Conclusions

Exploration strategies are a fundamental component for autonomous mobile robots whose task is to build map of an initially unknown environment.

A gap in the current research is that there is no standard evaluation method for comparing exploration strategies. They are usually relatively compared with each other but never with an absolute optimal baseline.

In this thesis, we presented the development of a framework that returns the optimal exploration path for a specific environment. The problem of determining an optimal exploration path has been faced with a novel approach, which formulates the exploration problem as a search problem and assumes that the environment is initially known. In this way, we can apply usual off-line search algorithms, like A* and branch and bound. The proposed framework contributes to the definition of a standard methodology for comparing exloration strategies in an absolute way.

Several issues emerged during this work. The most remarkable one is that, since the search problem has exponential complexity, due to the number of nodes generated, it ought to be relaxed by changing initial conditions and imposing constraints on boundary cells, so that the framework can be applicable to real environments and return a solution in a reasonable time. We pointed out where it is possible to intervene, in order to relax the problem and so to have less computational complexity, and we provided some examples of constraints.

Experimental results validate the formulation we provided for the problem of determining the optimal solution for the exploration in a specific environment. All expected trends about computational time and quality of solutions are satisfied. Specifically, we observed that by applying different constraints on boundary cells the quality (optimality) of the solution does not vary much, but the computational time is abated.

Several issues are worth for further investigation. First of all, we need a deeper theoretically analysis of the problem. For example, we can observe that several paths are duplicated. So it is necessary to understand what relations there are among different states (i.e., inclusion, equality, etc.).

Secondly, we could define better heuristics, so that search algorithms do not degenerate to breadth-first search.

In addition, more experiments need to be carried out, by testing other environments, in order to assess stronger conclusions.

Moreover, this framework could be extended to the case of multirobot setting, where multiple robots explore the map in a coordinated way. A method could be to apply coordination approaches, where robots are seen as cooperative agents.

The last but not the least issue concerns an optimization of the current implementation of the framework. For example, the ordering algorithm of the frontier can be improved by implementing a binary insertion sort algorithm.

In a broader view, it would be interesting to investigate how this framework, that determines the optimal exploration path in a specific environment, could be useful to improve real on-line exploration strategies, by exploiting hints deriving from optimal solutions found by the framework.

Bibliography

[1] Amigoni, F. Experimental evaluation of some exploration strategies for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2008), Pasadena, California, USA, May 19-23 2008* (2008), IEEE, pp. 2818–2823.

[2] Amigoni, F., and Basilico, N. On evaluating performance of exploration strategies for an autonomous mobile robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2008), September 22-26, 2008, Acropolis Convention Center, Nice, France* (2008), IEEE.

[3] Amigoni, F., and Basilico, N. Exploration strategies based on multi-criteria decision making for search and rescue autonomous robots. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), Taipei, Taiwan, May 2-6 2011* (2011), IEEE, pp. 99–106.

[4] Amigoni, F., and Gallo, A. A multi-objective exploration strategy for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2005), Barcelona, Spain, April 18-22 2005* (2005), IEEE, pp. 3850–3855.

[5] Deng, X., and Papadimitriou, C. H. Exploring an unknown graph. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS1990), St. Louis, Missouri, USA, October 22-24 1990* (1990), vol. I, IEEE, pp. 355–361.

[6] Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation 7*, 6 (December 1991), 859–865.

[7] Franchi, A., Freda, L., Oriolo, G., and Vendittelli, M. A decentralized strategy for cooperative robot exploration. In *Proceedings*

of the 1st International Conference on Robot Communication and Coordination (ROBOCOMM2007), Athens, Greece, October 15-17 2007 (2007), A. F. T. Winfield and J. Redi, Eds., vol. 318 of *ACM International Conference Proceeding Series*, IEEE, pp. 7:1–7:8.

[8] Franchi, A., Freda, L., Oriolo, G., and Venditelli, M. A randomized strategy for cooperative robot exploration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA2007), Roma, Italy, April 10-14 2007* (2007), IEEE, pp. 768–774.

[9] Gabriely, Y., and Rimon, E. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry: Theory and Applications 24*, 3 (2003), 197–224.

[10] Georgiev, A., and Allen, P. K. Localization methods for a mobile robot in urban environments. *IEEE Transactions on Robotics 20*, 5 (October 2004), 851–864.

[11] Gini, G., and Caglioti, V. *Robotica*. Zanichelli, 2003.

[12] González-Baños, H. H., and Latombe, J.-C. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research 21*, 10-11 (October 2002), 829–848.

[13] Koenig, S. Exploring unknown environments with real-time search or reinforcement learning. In *Proceedings of the Conference on Advances in Neural Information Processing Systems 11 (NIPS1998), Denver, Colorado, USA, November 30 - December 5, 1998* (1998), M. J. Kearns, S. A. Solla, and D. A. Cohn, Eds., The MIT Press, pp. 1003–1009.

[14] LaValle, S. M. *Planning algorithms*. Cambridge University Press, 2006.

[15] Lee, D., and Recce, M. Quantitative evaluation of the exploration strategies of a mobile robot. *International Journal of Robotics Research 16*, 4 (August 1997), 413–447.

[16] Leonard, J. J., and Feder, H. J. S. A computationally efficient method for large-scale concurrent mapping and localization. In *Proceedings of the Robotics Research: Ninth International Symposium (ISRR1999), Snowbird, Utah, USA, October 9-12 1999* (1999), J. Hollerbach and D. Koditschek, Eds., vol. 6 of *Springer Tracts in Advanced Robotics*, Springer-Verlag, pp. 169–176.

[17] Nagatani, K., Okada, Y., Tokunaga, N., Kiribayashi, S., Yoshida, K., Ohno, K., Takeuchi, E., Tadokoro, S., Akiyama, H., Noda, I., Yoshida, T., and Koyanagi, E. Multirobot exploration for search and rescue missions: A report on map building in RoboCupRescue 2009. *Journal Field Robotics 28*, 3 (May-June 2011), 373–387.

[18] Peniak, M., Marocco, D., and Cangelosi, A. Autonomous robot exploration of unknown terrain: a preliminary model of mars rover robot. In *Proceedings of the 10th ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA2008), Noordwijk, The Netherlands, November 11-13 2008* (2008).

[19] Rekleits, I. Exploration tutorial. In *Proceedings of the Seventh Canadian Conference on Computer and Robot Vision (CRV2010) SLAM camp, Ottawa, Ontario, May 29 2010* (2010).

[20] Russell, S. J., and Norvig, P. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[21] Se, S., Lowe, D. G., and Little, J. J. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *International Journal of Robotics Research 21*, 8 (August 2002), 735–760.

[22] Siegwart, R. Y., and Nourbakhsh, I. R. *Introduction to autonomous mobile robots*. MIT Press, 2004.

[23] Sim, R., and Dudek, G. Effective exploration strategies for the construction of visual maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2003), Las Vegas, California USA, October 27-31 2003* (2003), vol. 4, pp. 3224–3231 vol.3.

[24] Singh, S., Simmons, R. G., Smith, T., Stentz, A., Verma, V., Yahja, A., and Schwehr, K. Recent progress in local and global traversability for planetary rovers. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA2000), San Francisco, California, USA, April 24-28 2000* (2000), IEEE, pp. 1194–1200.

[25] Taylor, C. J., and Kriegman, D. J. Exploration strategies for mobile robots. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation (ICRA1993), Atlanta, Georgia, USA, May 1993* (1993), IEEE, pp. 248–253.

[26] Thrun, S. Robotic mapping: A survey. In *Exploring artificial intelligence in the new millennium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann Publishers Inc., San Francisco, California, USA, 2003, pp. 197–224.

[27] Thrun, S., Burgard, W., and Fox, D. *Probabilistic Robotics*. The MIT Press, 2005.

[28] Tovar, B., Muñoz-Gómez, L., Murrieta-Cid, R., Alencastre-Miranda, M., Monroy, R., and Hutchinson, S. Planning exploration strategies for simultaneous localization and mapping. *Robotics and Autonomous Systems 54*, 4 (2006), 314–331.

[29] Tovey, C. A., and Koenig, S. Improved analysis of greedy mapping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2003), Las Vegas, California USA, October 27-31 2003* (2003), pp. 3251–3257.

[30] Yamauchi, B. A frontier-based approach for autonomous exploration. In *Proceedings of the 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA1997), Monterey, California, USA, July 1997* (1997), IEEE, pp. 146–151.