POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingeneria

A NEXT-BEST-SMELL APPROACH FOR REMOTE GAS DETECTION WITH A MOBILE ROBOT

Relatore: Prof. Francesco Amigoni
Correlatore: Dr. Erik Schaffernicht

Tesi di Laurea Magistrale di:
Riccardo Polvara, matricola 817572
Marco Trabattoni, matricola 823346

Anno Accademico 2014-2015

To my Dad, who has always dreamt for me to become an engineer.

Contents

List of Figures

List of Tables

List of Listings

Abstract

The problem of gas detection is relevant to many real-world applications, such as leak detection in industrial settings and landfill monitoring. In this thesis we address the problem of gas detection in large areas with a mobile robotic platform equipped with a remote gas sensor. We propose an online algorithm based on the concept of Next-Best View for solving the coverage problem to which the gas detection problem can be reduced. To demonstrate the applicability of our method to real-world environments, we performed a large number of experiments, both in simulation and in a real environment. Our approach proves to be highly efficient in terms of computational requirements and to achieve good performance.

Sommario

Il problema legato al rilevamento del gas è di fondamentale importanza in molte applicazioni, quali l'identificazione delle perdite in ambienti industriali e il monitoraggio delle discariche. In questa tesi affrontiamo il problema del rilevamento del gas in ampi spazi interni con una piattaforma robotica mobile, equipaggiata con un sensore gas remoto. Proponiamo un algoritmo online basato sul concetto di Next-Best-View per risolvere il problema di copertura a cui il problema del rilevamento del gas può essere ridotto. Per dimostrare l'applicabilità del nostro metodo abbiamo svolto un elevato numero di esperimenti, sia simulati che in un ambiente reale. Il nostro approccio ha dato prova di essere altamente efficiente in termini di risorse computazionali, e di raggiungere buone prestazioni.

Aknowledgment

Firstly, I would like to express my sincere gratitude to my advisors Prof. Francesco Amigoni and Dr. Erik Schaffernicht, for support while I was working on my Master thesis and the associated article, for their patience, motivation, and immense knowledge. Their guidance helped me during the time of research and writing of this thesis.

My sincere thanks also goes to Prof. Achim Lilienthal, who provided me an opportunity to join his team, and who gave access to the laboratory and research facilities. Without his precious support it would not be possible to complete this work.

I thank my fellow labmates Tomek, Malcolm, Ravi, Chit and Han for the stimulating discussions, for the sleepless nights we were working together before my departure, and for all the fun we have had in the five months spent in Örebro.

Last but not the least, I would like to thank my family: my parents and my sister, for supporting me spiritually throughout writing this thesis and my life in general.

Chapter 1

Introduction

Recent advances in mobile robotics showed that the employment of autonomous mobile robots can be an effective technique to deal with tasks that are difficult or dangerous for humans. Examples include map exploration, map coverage, search and rescue, and surveillance. Fundamental issues involved in the development of autonomous robots span through locomotion, sensing, localization, and navigation. One of the most challenging problems is the definition of navigation strategies. A navigation strategy can be generally defined as the set of techniques that allow a robot to autonomously decide where to move in the environment in order to accomplish a given task.

In the last years robot olfaction, a branch of robotics that combines gas sensors with the freedom of maneuver of mobile robots, has attracted more and more interest [30]. This interested has further increased with the birth of remote sensors which, unlike in-situ ones that must come directly into contact with the gas to be able to perceive it, allow to perform scans of the environment from considerable distances [7, 11], up to a couple of tens of meters, as in the case of TDLAS sensor, based on the spectroscopy analysis of the wavelengths of a laser [22, 14]. Because of their large size and cost it is virtually impossible to adopt a distributed solution with multiple sensors of this type in order to cover an environment, and therefore it has been tried to integrate this technology with mobile robots.

In this thesis we focus on the problem of gas detection in large environments, like an office or a corridor with multiple rooms, calculating a path to cover all the area and to gain a complete mapping of the detected gas. Our approach is online, namely it incrementally perceives the environment and makes decisions on the basis of the knowledge obtained so far rather than a priori like happens with an offline strategy (Figure 1.1).

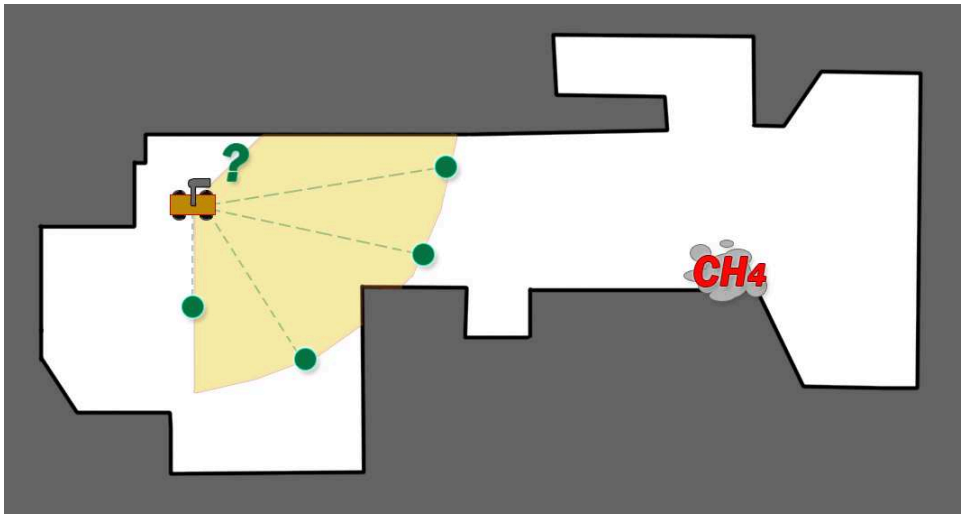

Figure 1.1: The Next-Best-Smell algorithm identifies, on the frontier separating explored and unexplored area, the next pose the robot has to reach combining in a single utility function different criteria.

In more details, we propose a Next-Best-Smell algorithm that exploits *Multi-Criteria-Decision-Making* (MCDM) [3] to combine different criteria in order to decide the next position the robot should reach to cover the environment with its remote gas sensor. After having reached the last position chosen, the robot acquires a new partial map, integrates it with the global map saved in his memory, decides the next location and reaches it, and repeat the steps above until it reachs an acceptable level of coverage. Candidate locations are identified among multiple points on the frontier dividing scanned and unscanned space. In MCDM a robot evaluates the candidate locations in a partially explored environment according to an utility function that combines different criteria (for example, the distance of the candidate location from the robot and the expected amount of new information acquirable from there). Criteria are combined in a way that accounts for their synergy and redundancy.

The proposed approach seems to be promising: results reported in the following sections demonstrate that, because of the greedy nature of the Next-Best-Smell algorithm it is possible to cover most part of the environment (a percentage around 80-90%) in relatevely few rounds, usually half of the ones required for a full coverage, but sometimes also less. For this reason it can represent an interesting solution in the field of gas detection where gas doesn't remain near the source but moves around in the environment.
These results are confirmed also by a simulated comparison made with an

offline approach, in which our algorithm obtained good performance, really close to the other's ones in some circumstances.

The structure of this thesis is as follows: first of all (Chapter 2) we illustrate the current state of the art for what concerns gas detection and online navigation, then we will define the problem we want to solve and our solution proposal (Chapter 3). After having defined the architecture of our system (Chapter 4), we will report the results of our experiments (Chapter 5), performed both in simulation and with a physical platform, and the comparison with an offline approach developed at MRO Lab at Örebro University. In the end (Chapter 6) we summarize our approach, highlighting the main concepts, its advantages, and future works.

Chapter 2

State of the art

The idea of adopting online exploration strategies represents a novel approach in the field of mobile robotic olfaction. Due to the lack of literature we split the current chapter in two parts, the first one aiming to illustrate the main techniques used until now for gas detection and the second one dedicated to online navigation.

2.1 Robot olfaction

Stationary networks of gas sensors represented until some years ago the most spread technique for gas detection, especially in situation of pollution monitoring [31]. The main particularity of this choice was the utilization of sensors defined *in situ* [23] because they should be in direct contact with the gas to perceive it. This represents a strong limitation because it poses the problem of their initial positioning and how they should be moved after some changes in the environment.

Recently, interest started growing towards mobile robot olfaction, a novel approach for gas detection using a mobile platform, with all the advantages that it can offer [21, 20].

The first step was combining mobile robots with *in situ* sensors (Figure 2.1), and this solution was used for mapping gas distribution [9, 8] and leak detection [25, 11]. The problem about this approach is that the robot still moved between predefined positions, along a path that should be recomputed after changes in the environment, as for the static networks.

One of the biggest contributions in this field was represented by the adoption of Tunable Diode Laser Absorption Spectroscopy (TDLAS) sensors, able to perform remote measurements up to a limited range ($\sim$ 30, 50

Figure 2.1: Because the flexibility they can offer, mobile robots (left) represent an important innovation in a dangerous task like the gas detection one. In the first years they were combined with in-situ gas sensors (right), whose limitation is the fact they have to be in direct contact with the gas to perceive it.

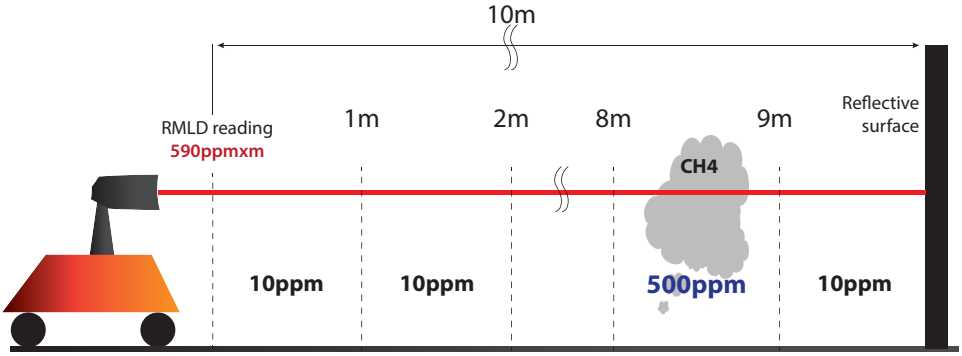

Figure 2.2: The Remote Methane Leak Detector is a TDLAS sensor which can report the integral concentration of methan along its laser beam (parts per million x meter)

meters) [8], as shown in Figure 2.2. Because performing a scanning operation is an extremely time consuming operation and the robot's battery is limited, it became really important to find a path minimizing the number of times in which the robot stopped to sense for gas.

2.2 Exploration strategies

In general, almost all navigation strategies take as input a state enclosing information about the environment and provide as output the next location reachable by the robot. The way in which this process is performed allows to distinguish two class: offline and online strategies. In the first case, the set of possible destinations is computed for every possible input state before the beginning of the exploration task. With an online strategy, instead, the decision is computed during the task execution for the different situations the robot encounters.

Defining an exploration strategy can involve a large number of different

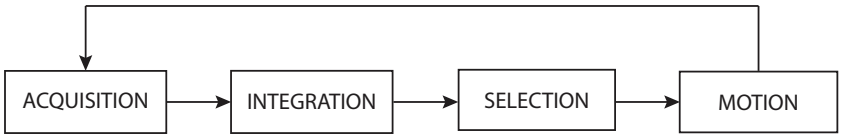

Figure 2.3: Next-Best-View system: acquire a partial map of the surrounding environment, integrate it with the global map, identify the new position to reach and then reach it. These steps are usually reiterated until full coverage is achieved.

criteria, each one with its own importance that can vary according to our scope. For example, one could search for the strategy minimizing the time spent in exploration while another for the one minimizing the total distance travelled by the robot. But sometimes a combination of these criteria can also be desirable.

Following the idea proposed by the offline strategy, some approaches were suggested in the mobile olfaction field but the most promising seems to be the one proposed by Arain *et al.* in [6], as extension of the work of Tamioka *et al* [28], based on the combination of the *Art Gallery Problem* and the *Travelling Salesman Problem*: first of all they identify all the positions from which the robot is able to see all the environment and then they calculate the minimum path connecting all these positions. In this way Arain *et al.* are able to find a closely optimal solution to cover all the environment and therefore detect any possible gas leak.

Starting from the assumption that gases move in the environment, our contribution in this thesis is a new approach that takes inspiration from the online strategy for map building, where the environment is usually unknown at the beginning and the robot has to incrementally discover it.
This kind of strategies usually follows the repetition of simple steps, reported in Figure 2.3: sensing the surrounding environment to build a partial map, integrating it with the current global map, selecting the next observation location, and reaching it.

An important feature of these systems, called Next-Best-View (NBV) systems, is how to choose the next observation location among a set of candidate locations, evaluating them according to some criteria. Usually, in NBV systems, candidate locations are chosen in such a way that they are on the frontier between the already explored free space and the unknown one [32], and they should be reachable from the current position of the robot.

During the evaluation phase of a candidate position, different criteria can be used and combined in diffent ways. A simple one is the *travelling cost* [33], according to which the best observaton location is the nearest one. Other works combine the travelling cost with *expected information gain*,

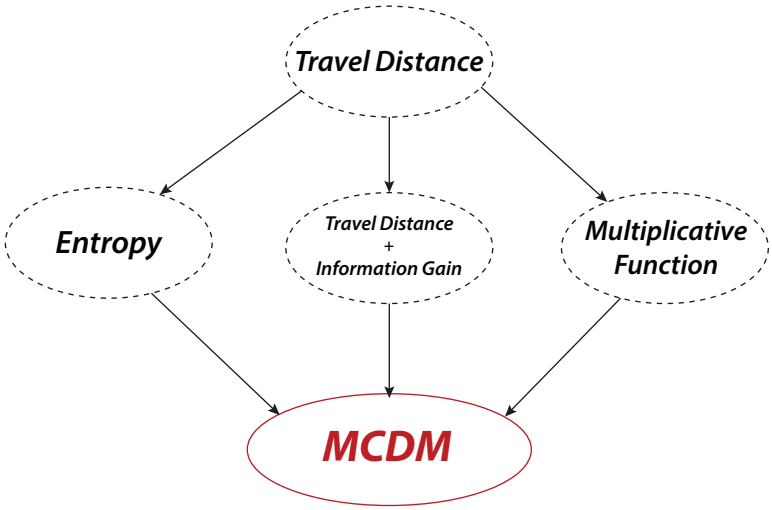

Figure 2.4: Different approaches were suggested in the field of map coverage but most of them were not theoretically-grounded. For this reason Multi Criteria Decision Making was introduced with promising results.

that is the expected amount of new information about the environment the robot can acquire performing a sensing action from the candidate location p. Given a candidate location p and called $c(p)$ and $A(p)$ the travelling cost and the expected information gain, respectively, Gonzáles-Baños and Latombe [15] combine these two criteria with an ad hoc function in order to compute an overall utility (λ weighs the travelling cost and the information gain):

$$u(p) = A(p)e^{-\lambda c(p)} \tag{2.1}$$

Similar criteria were considered also by Stachniss and Burgard [27], where the cost of reaching a candidate locations p, measured as the distance $d(p)$ from the actual position of the robot, is linearly combined with the information gain $A(p)$ acquirable from p:

$$u(p) = A(p) - \beta d(p) \tag{2.2}$$

where β balances the relative weight of the two criteria.
Other examples include the work of Amigoni *et al.* [4], in which a technique based on relative entropy is used, and of Tovar *et al.* [29], where several criteria are employed in a multiplicative function to obtain a global utility value.

The problem with the previous strategies is that they define an aggregation method too much dependent on the criteria considered. For this reason, Amigoni and Gallo [5] proposed a more theoretically-grounded approach based on multi-objective optimization, in which the best candidate

location is selected on the Pareto frontier (see Figure 2.4).

According to the goals of this thesis, in the following chapters we propose the adoption of a decision theoretical framework called *Multi-Criteria Decision Making* (MCDM) for the addressing and solving the gas detection problem. MCDM represents a flexible way to combine criteria that should be contrasted with ad hoc compositions (like weighted mean of [27], the multiplicative function of [29], and the other works listed above). This technique deals with problems in which a decision maker has to choose among a set of alternatives and its preferences depend on different, and sometimes conflicting, criteria. It is employed in several applicative domains such as Economy, Ecology, and Computer Science [16, 26]. The Choquet fuzzy integral [18] is used in MCDM to combine different criteria in a global utility function whose main advantage is the possibility to account for the relations between criteria [16, 17].

Chapter 3

Problem Definition and Solution Proposal

In this chapter we provide a short description of the problem of gas detection using a mobile robot equipped with a remote gas sensor in a known environment. Therefore we explain the approach used: how the environment and the robot is represented, how the problem has been modeled as one of optimizing a multiobjective function and how this is solved with the concept of MCDM using the Choquet Fuzzy Integral. We call our approach Next-Best-Smell.

3.1 Environment Representation

We assume the task of gas detection to be performed in an environment with no major changes in the gas distribution due to the presence of wind or other means and that gas sources do not occur on top of obstacles. These hypotheses hold in many real world scenarios, but they need to be re-evaluated if the task is one of localizing the gas source or mapping the gas distribution of the environment.

Gas sensing is carried out using a TDLAS remote gas sensor mounted on a mobile robot. TDLAS sensor reports the integral concentration measurements along a beam; in order to scan a portion of the environment a pan-tilt unit is used to aim the sensor at various orientation, performing a sweep and thus scanning a circular sector of range r and scan angle ϕ. The scan angle phi is defined as the difference $\phi_{min} - \phi_{max}$, where ϕ_{min} and ϕ_{max} are the angles defining the sweep that the gas sensor has to perform. The

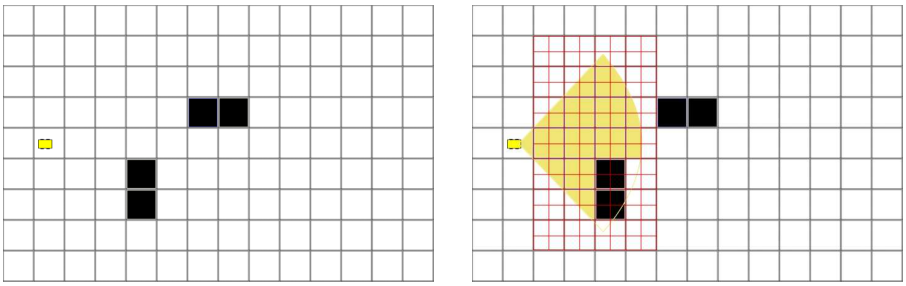

Figure 3.1: Grids representing the environment. We can see how when performing a sensing operation (figure on the right) a grid with a higher resolution is used

values of r and ϕ are restricted by the limit value r_{max} and by the maximum opening angle, respectively, due to sensor's physical constraints.

The environment in which the mobile robot acts is assumed to be known in advance. Given a map representing the environment to explore, we can divide it into a grid A of n identical sized cells: $A = \{a_1, a_2, ..., a_n\}$. The cells of A can be partitioned into two subsets: the subset O, composed of the cells containing some kind of obstacle and thus are not traversable by the robot and able to stop the beam of the gas sensor, and the subset F of free cells, not containing any obstacle and thus traversable by the robot. Two grids working in this way are used to represent the environment, one for navigation and one for gas detection, of possibly different resolutions.

We can specify the state of the robot in the grid with two values: its position and its orientation. The position is represented by the free cell a in which the robot is currently located, and we assume the robot to always be in the center of the cell. Given a set Θ of possible orientations, with Θ equally spaced in $[0, 2\pi)$, we can define the possible robot poses as the couples (a, θ) with $a \in F$ and $\theta \in \Theta$.

When not moving, the robot can perform a sensing operation to analyze the presence of gas in a portion of the environment. A sensing operation is thus defined by a robot pose $p = (c, \theta)$, the radius r, and the scan angle ϕ, that together define the *Field of Vision* (FoV) of the robot, as the set of cells that are perceived from p.

We can now introduce the concept of visibility: a free cell $a \in F$ is *visible* from a robot pose $p = (c, \theta)$ if the line segment spanning from the center of c to the center of a does not intersect any occupied cell and if the center of a is inside the circular sector centered in p and defined by r and ϕ. This corresponds to the assumption that all obstacles fully occupy grid cells and that they are high enough to obstruct the line of sight of the remote gas sensor. During a sensing operation, all the cells which are visible from p are

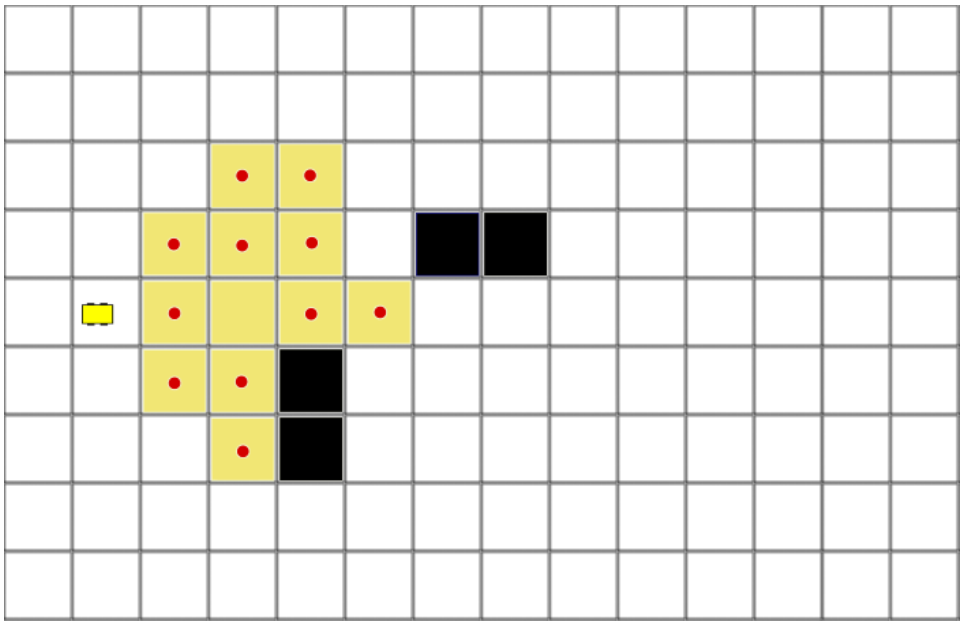

Figure 3.2: Candidate positions (marked with red dots) are the cells on the boundary between the scanned and unscanned portion of the environment.

scanned.

The problem of planning a path for gas detection in a given environment is that of finding the optimal sequence of sensing operations $\langle((c_1, \theta_1), r_1, \phi_1), ((c_2, \theta_2), r_2, \phi_2), \ . \ . \ ., ((c_n, \theta_n), r_n, \phi_n)\rangle$ to be performed in order to scan (cover) all the free cells of the environment, with pose (c_1, p_1) as the starting pose of the robot in the environment.

The solution we propose is the *Next-Best-Smell* approach, an on-line greedy algorithm following the concept of *Next-Best-View* (NBV). At each step of the algorithm the robot performs a sensing operation from its current pose and then selects the next pose from a set of candidate ones, evaluating each of them and choosing the one with the best evaluation.

3.2 Candidate Evaluation

We define *candidate positions* as the cells on the boundary between the portion of environment that has already been scanned and the one which has yet to be perceived.

For each of these candidate position we can obtain multiple candidate robot poses, one for each orientation θ belonging to the set Θ defined above.

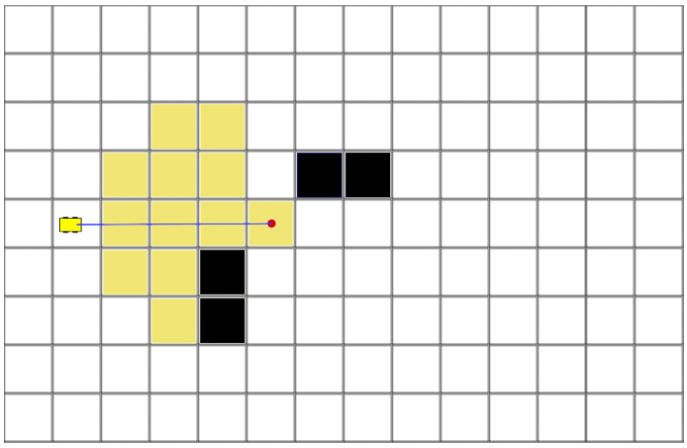

Figure 3.3: Travel distance criterion is computed as the distance between the current robot position and the candidate pose.

In order to choose the best pose among the candidate ones, we identified three *criteria* useful for the evaluation:

- *Travel distance*, computed as the distance between the current robot position and the candidate pose.
- *Information gain*, computed as the number of free unscanned cells that the robot will be able to perceive from the candidate pose.
- *Sensing time*, computed as the scan angle required to sense all the free unscanned cells visible from the candidate pose in the sensing operation (ϕ_{max} - ϕ_{min}).

For each of these criteria, a utility value indicating how much a candidate pose satisfies the criteria is obtained; the value is normalized in order to obtain a number between 0 and 1, the higher the value the better the pose is with respect to the criterion considered.

3.3 Using MCDM to find the Next-Best-Smell

In order to select the best candidate pose, a global utility function combining these utility values is necessary. We can define this function using the *Multi-Criteria Decision Making* (MCDM) method. An important aspect when evaluating on multiple criteria is the dependency among them, and a simple weighted average is unable to model this. For example, two criteria

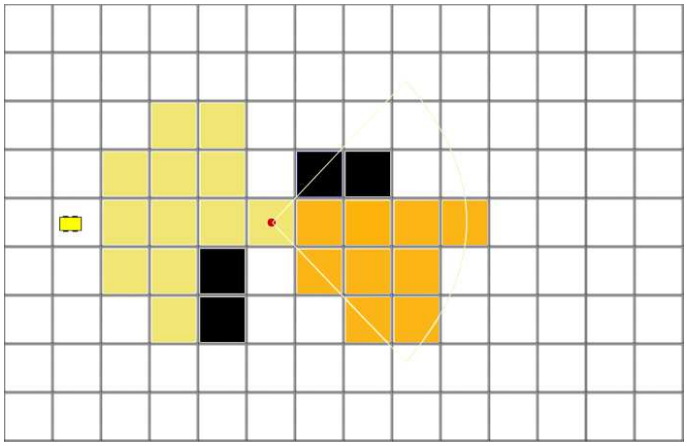

Figure 3.4: Information gain criterion is computed as the number of free unscanned cells that the robot will be able to perceive from the candidate pose.

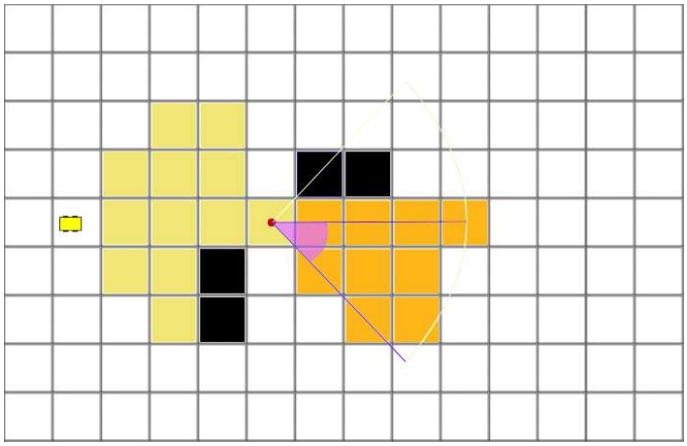

Figure 3.5: Sensing time criterion is computed as the scan angle required to sense all the free unscanned cells visible from the candidate pose in the sensing operation (ϕ_{max} - ϕ_{min}).

might estimate similar features using two different methods. In this case a relation of *redundancy* holds among them, and their overall contribution to the global utility should be less than the sum of their individual ones.

On the other hand, two criteria might estimate two very different features, meaning that in general a candidate optimizing both of them is hard to find. In this case a relation of *synergy* holds among the criteria, and their overall contribution to the global utility should be larger than the sum of the individual ones.

In order to account for the relations of redundancy and synergy when combining the utilities of criteria, MCDM provides an aggregation method that can deal with this aspect: the *Choquet Fuzzy Integral*. In order to present it, we first need to introduce a function μ : P(N) $\rightarrow$ [0, 1], where P(N) is the power set of N, with the following properties:

- $\mu(\{\emptyset\}) = 0$,
- $\mu(\mathrm{N}) = 1$,
- if A $\subseteq$ B $\subseteq$ N , then $\mu(A) \leq \mu(B)$.

This means that μ is a *fuzzy measure* on the set N of criteria and it will be used to specify weights for each subset of criteria. The weights specified by μ describe the above mentioned relations among criteria: if two criteria are redundant, then $\mu(c1, c2) < \mu(c1) + \mu(c2)$, while if they are synergic $\mu(c1, c2) > \mu(c1) + \mu(c2)$; in case $\mu(c1, c2) = \mu(c1) + \mu(c2)$ we say that the criteria are *independent*.

The global utility function of a candidate pose p can then be computed as the discrete Choquet integral with respect to the fuzzy measure μ using the utilities of p on the criteria:

$$f(u_p) = \sum_{j=1}^{n} (u_{(j)}(p) - u_{(j-1)}(p))\mu(A_{(j)},$$

where j indicates the $j-th$ criterion after criteria have been permutated in order to have, for a candidate pose p: $u_{(1)}(p) \leq ... \leq u_{(n)}(p) \leq 1$. We assume $u_{(0)}(p) = 0$. The set A is defined as $A_{(j)} = i \in N | u_{(j)}(p) \leq u_{(i)}(p) \leq u_{(n)}(p)$. It is easy to see how the weighted average is a specific case of the Choquet Integral, in which all the criteria are considered as independent.
Let's consider a simple example, using the following utilities for criteria and coalitions:

candidate	Travel Distance	Information Gain	Sensing Time	Weighted Average	Choquet Integral
p1	0.95	0.1	0.9	0.76	0.47
p2	0.7	0.6	0.7	0.68	0.64
p3	0.05	0.8	0.1	0.22	0.47

Table 3.1: Example of candidate evaluation with MCDM.

$$\mu(TravelDistance) = 0.2$$
$$\mu(InformationGain) = 0.6$$
$$\mu(SensingTime) = 0.2$$
$$\mu(TravelDistance, InformationGain) = 0.9$$
$$\mu(TravelDistance, SensingTime) = 0.4$$
$$\mu(InformationGain, SensingTime) = 0.9$$

Results on three candidate positions are shown in Table 3.1. If we use the weighted average as aggregation function, the next position we should reach is $p1$ but it does not seem the best solution because it is largely unsatisfactory from the travel distance cost's point of view. For this reason, using the Choquet Fuzzy Integral we can select as solution $p2$ because is has criteria in a balanced way.

Chapter 4

Architecture of the System

In this chapter we illustrate how the Next-Best-Smell algorithm is structured. First of all we introduce, with the help of a flowchart, the main blocks that compose our work; then we describe more in depth the techniques adopted and implemented to calculate the three criteria considered and to model the online coverage problem.

4.1 Algorithm Overview

As shown in Chapter 2, the Next-Best-View approach consists in the repetition of different steps until a chosen condition is satisfied, usually the full coverage of an environment's map. Often these steps are the following: acquire and integrate a scan from the current position, identify the next best position to reach, reach it, acquire and integrate the new scan in the global map representing the phenomenon that the robot is perceiving. Following an approach like this, it could happen that the robot goes into a corridor and then it is not able to move anymore since it does not see any further interesting position in front of it. For this reason, in the online exploration field the adoption of backtracking techniques allowing the robot to go back to previous pose until it can find a new unexplored path to follow is largely diffused. These basic concepts are collected and modeled in Figure 4.1, that summarizes how the Next-Best-Smell algorithm works.

Looking at the flowchart and starting from the top, the first operation we do at every iteration of the algorithm is to check if we have reached the coverage target we chose before. There are no limits in it; in our tests we were able to cover the full map but it is also possible to adopt a coverage value less than 100%. If this condition is satisfied, the algorithm ends with success otherwise a control loop starts.

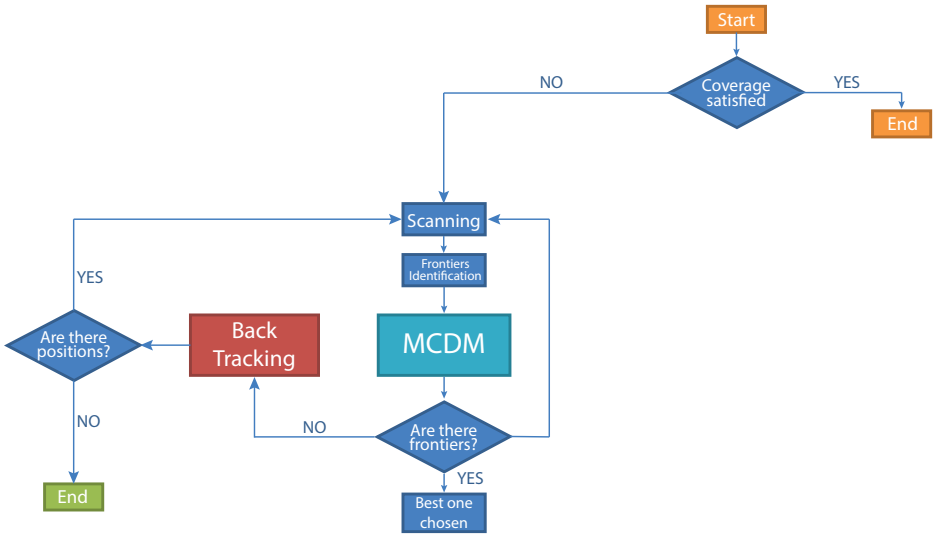

Figure 4.1: After checking the termination condition, the robot scans the surrounding environment, identifies new candidate positions and, after having evaluated them, selects the best one. These steps are iterated untill the full coverage is reached.

This loop represents the core of our online approach and it is repeated until the coverage constraint is fullfilled or the robot comes back to the initial position without other candidate poses.

The first operation performed by the robot is a scanning of the surrounding environment, with the limitation introduced by the Field of Vision parameter adopted. The scanning phase consist in marking as seen, assigning a value of 2, those cells visible from the robot's current position. If it can find, with the method that will be shown in the following section, positions candidate to be reached on the edges separating the known and the unknown area, it can start the evaluation phase.

Evaluating a position consists in applying the Choquet Fuzzy Integral described in Chapter 3, to get a score belonging to $[0, 1]$. All those positions from which the robot can't acquire a new portion of the map are discarded; the others are collected in a record with their evaluation. If there is at least one frontier in this record, the best one (the one with the highest score) is chosen as the next goal position. Therefore the robot will reach it and it will redo all the previous steps, starting from the evaluation of the coverage condition, until the algorithm's end.

If there are no interesting frontiers, it means that the robot is stuck at the end of a corridor or it has already explored all the cells around it. At this point it performs a backtracking operation: the robot returns to its

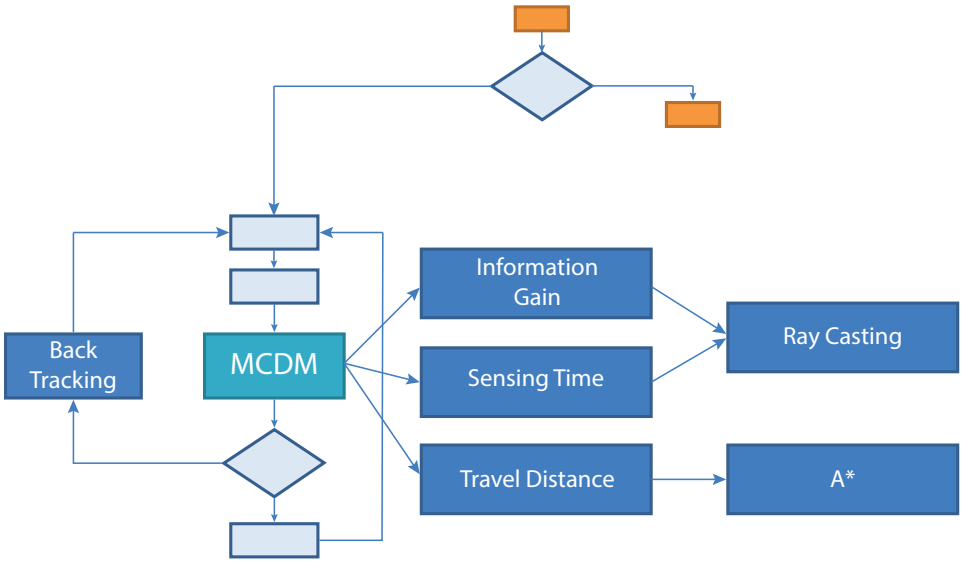

Figure 4.2: The three criteria are calculated with different techniques: ray casting is used to estimate the value of information gain and sensing time, while A is implemented for the travel distance.*

previous pose or, if it can't find any further candidate position from there, to the last pose that allows it to find a new path to follow. While performing backtracking, if the robot comes back to the initial position it means that there are no more candidate positions in the map and therefore the exploration ends, in most of the case successfully (as it will be shown in Chapter 5, in some circumstances the robot could not complete the full map coverage, even if the number of ignored cells is almost irrelevant compared to the total one).

4.2 Techniques used for Implementation

In this section we describe the techniques adopted to calculate the three criteria used in the evaluation phase: *information gain, sensing time* and *travel distance*. In the end it will be also explained how the exploration phase is implemented in the algorithm.

Figure 5.1 represents the flowchart of our algorithm with the evaluation block expanded. As it is possible to see, the evaluation phase is computed using the *Multi Decision Criteria Making* approach that merges the three criteria's contribution in a single utility function, as described in Chapter 2.

Listing 4.1: Ray caster pseudocode

```
robotX, robotY := coordinates of the robot in the map
x, y := coordinates of cell to scan
curX, curY := robotX, robotY    //current cell of the ray
m := 0  //variable to move along the ray
hit := 0        //set to 1 if an obstacle is hit by the ray
slope := atan2(y - robotY, x - robotX)  //slope between the cell to
                                        //scan and the robot
if(distance from robot to cell to scan <= range) //ray is cast only
                                                 //if the cell is in
                                                     range
    while(hit == 0)     //move along the ray until obstacle is found
        curX = robotX - m*sin(slope)    //get coordinates of current
                                        //cell of the ray
        curY = robotY + m*cos(slope)
        if (obstacle in curX, curY) hit = 1  //if cell contains an
                                             //obstacle, stop the ray
        if (curX == x && curY == y)  //if the ray reaches the cell
                cell is visible      //to scan, then it is visible
                stop ray             //stop the ray when cell is found
        else m = m + 0.2        //move along the ray until obstacle
                                //is found or cell to scan is reached
```

4.2.1 Ray Casting

In order to implement the criteria of information gain and sensing time, we implemented a ray caster.

Ray casting is a rendering technique first introduced by Arthur Appel in 1968, and has been used for a variety of tasks in the computer graphics field.

The basic idea behind ray casting is to find what is to trace various rays from the eye, one for each pixel, and then find the closes object that blocks the path of that ray.

Our ray caster works in the following way: given a cell of interest, which in the case of calculating the value of information gain for a candidate position would be a free cell which has not yet been scanned, the slope of the line connecting the centers of the cell of interest with the one of the current position is calculated.

If the line lies inside of the FoV of the robot, and the distance between the center of the considered cell and the current robot position is not greater than the range of the TDLAS sensor, a ray is shot along the line.

By moving along the ray in small increments, each cell traversed by the ray is analyzed: if an obstacle is found, the ray is stopped, otherwise if the ray reaches the cell of interest it means that it is *visible* from the current

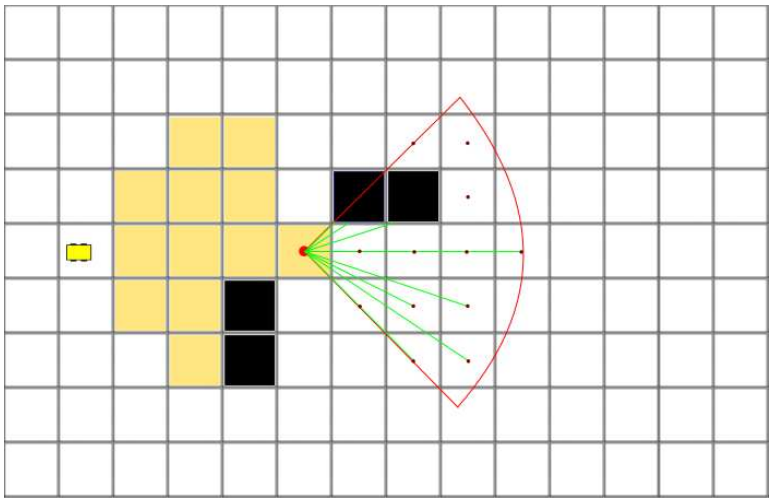

Figure 4.3: Ray casting used to calculate the value of information gain of a candidate pose. When a ray reaches the center of a free cell, the information gain value is increased.

position. An example of a ray being casted in pseudocode is shown in 4.1.

In Figure 4.3 we can see rays being casted to calculate the information gain of a candidate pose. The centers of visible cells are reached by rays, while rays pointed at non visible cells are blocked by obstacles.

The value of *information gain* is calculated as the number of free cells which have not yet been scanned and are visible from the candidate pose (Figure 4.4).

The value of *sensing time* is calculated as the angle between the first and the last rays to be shot in order to scan all the free unscanned cells visible from the candidate pose, and the values of the two slopes are stored to define the angle used in the following sensing operation (Figure 4.5).

Other than for these two criteria, we also used ray casting to identify candidate positions at each step of the algorithm, by finding visible free cells which are on the boundary between scanned and unscanned area.

4.2.2 A*

The last criteria we have to address is the travel distance, computed as the distance between the robot's current position and the target one. To have an estimate as precise as possible of such distance, we implemeted the A*

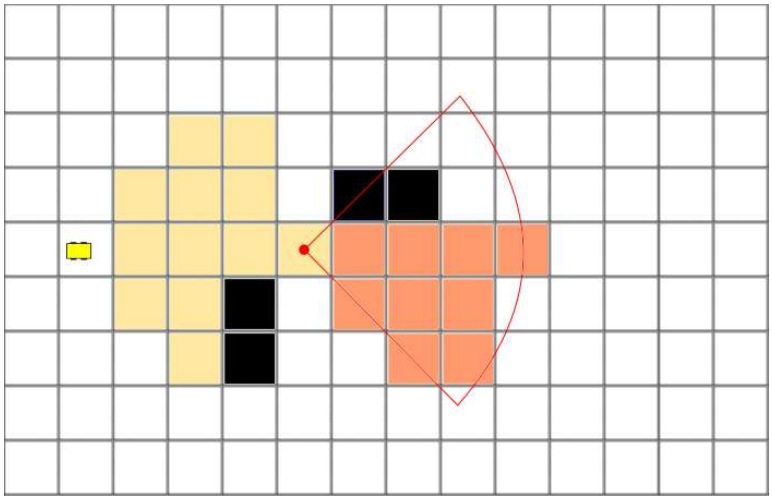

Figure 4.4: The information gain of a candidate pose is the number of visible free unscanned cells.

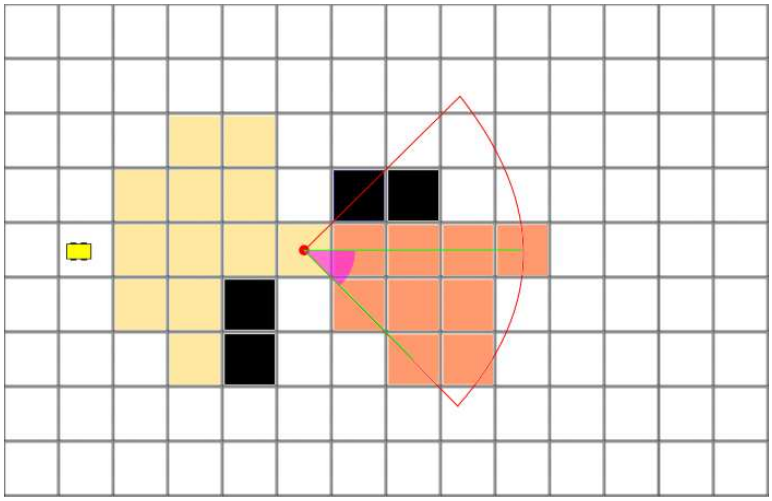

Figure 4.5: The sensing time of a candidate pose is the angle between the first and last rays shot to scan all the free unscanned cells visible from that pose.

algorithm, developed by Peter Hart *et al.* [19] as extension of Dijkstra work [13].

A* uses a best-first search to find the minimum cost path from an initial node to one goal node. Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule. Traversing the graph, A* builds up a tree of partial paths; the leaves of this tree (called the open set or fringe) are stored in a queue ordered by a cost function combining a heuristic estimate of the cost to reach a goal and the distance traveled from the initial node. Specifically, the cost function is

$$f(n) = g(n) + h(n). \tag{4.1}$$

where $g(n)$ is the cost of getting from the initial node to n and $h(n)$ is a heuristic estimate of the cost to get from n to any goal node.

To find the shortest path, the heuristic function must be *admissible*, meaning that it never overestimates the cost to get to the closest goal node from the actual one. In our experiments we use both the *Euclidean* distance and the *Manhattan* one.

If the heuristic h satisfies the additional condition $h(x) \leq d(x, y) + h(y)$ for every edge (x, y) of the graph (where d denotes the length of that edge), then h is called *consistent*. In this case, $A*$ can be implemented more efficiently, no node needs to be processed more than once. A pseudocode example of how $A*$ works is provided in the following Listing 4.2.

We provide a little example to clarify the operation perfomed by this algorithm largely used for plath planning purpose. In Figure 4.6 a robot and the target cell to reach are represented in a grid map. As first step, A* assigns a different value to each possible cell the robot can reach from its current one: in the example, a value of 10 is assigned to orthogonal cells, while a value of 14 to those diagonals. These value are reported in black in the image, and they represent the cost $g(n)$ to move from the actual position to that cell.
The red number represents instead the estimated distance between the considered cell and the goal, according to the heuristic chosen $h(n)$, in this case the Manhattan distance. This is calculated as the length of the path between two points in a grid based on a only horizontal and/or vertical movements (that is, along the grid lines).
The second step computed by A* is to sum $g(n)$ and $h(n)$ for all the cells and select the one with the smallest value. Starting from this point, the first step is repeated, expanding in the search tree the node corresponding to the

Listing 4.2: A pseudocode*

```
1     function A*(start,goal)
2     closedset := the empty set    // The set of nodes already
          evaluated.
3     openset := {start}    // The set of tentative nodes to be
          evaluated,
4     initially containing the start node
5     came_from := the empty map    // The map of navigated nodes.
6
7     g_score := map with default value of Infinity
8     g_score[start] := 0    // Cost from start along best known path.
9     // Estimated total cost from start to goal through y.
10    f_score = map with default value of Infinity
11    f_score[start] := g_score[start] + heuristic_cost_estimate(start,
          goal)
12
13    while openset is not empty
14        current := the node in openset having the lowest f_score[]
              value
15        if current = goal
16            return reconstruct_path(came_from, goal)
17
18        remove current from openset
19        add current to closedset
20        for each neighbor in neighbor_nodes(current)
21            if neighbor in closedset
22                continue
23
24            tentative_g_score := g_score[current] + dist_between(
                  current,neighbor)
25
26            if neighbor not in openset or tentative_g_score < g_score[
                  neighbor]
27                came_from[neighbor] := current
28                g_score[neighbor] := tentative_g_score
29                f_score[neighbor] := g_score[neighbor] +
                      heuristic_cost_estimate(neighbor, goal)
30                if neighbor not in openset
31                    add neighbor to openset
32
33    return failure
34
35 function reconstruct_path(came_from,current)
36    total_path := [current]
37    while current in came_from:
38        current := came_from[current]
39        total_path.append(current)
40    return total_path
```

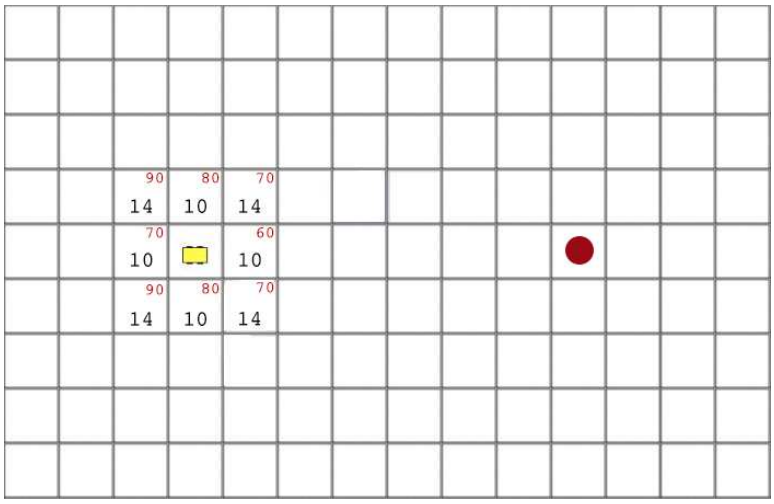

Figure 4.6: The initial pose of the robot and the target cells. Different weights are assigned to orthogonal and diagonal cells.

cell having the smallest $f(n)$, as defined in Equation 4.1, until reaching the goal. The final situation is represented in Figure 4.7.

At this point, proceeding backward from the goal to the initial pose of the robot, the path connecting these two points is built as the one passing each time in the cell minimizing Equation 4.1.

In this way, as shown in Figure 4.8, it is always possible to find the shortest path between the robot and the candidate position considered, obtaining a really good estimation of the real distance between these two points.

4.2.3 Depth-First Search

After having illustrated how we calculate information gain, sensing time and travel distance, it is also important to explain how we model and realize the online navigation. As explained in the previous section, the main problem that can arise when the robot does not follow precomputed paths is that it can be stuck at the end of a corridor or in a portion of map in which it cannot identify any further interesting position to reach. For this reason, implementing a backtracking technique that allows the robot to come back to previous positions is a good idea.

As it can be seen in Figure 4.9, we address this problem modeling the exploration as a search problem solved using the Depth-First Search (DFS) algorithm with backtracking, investigated for the first time by Charles Pierre Trémaux in the 19th century.

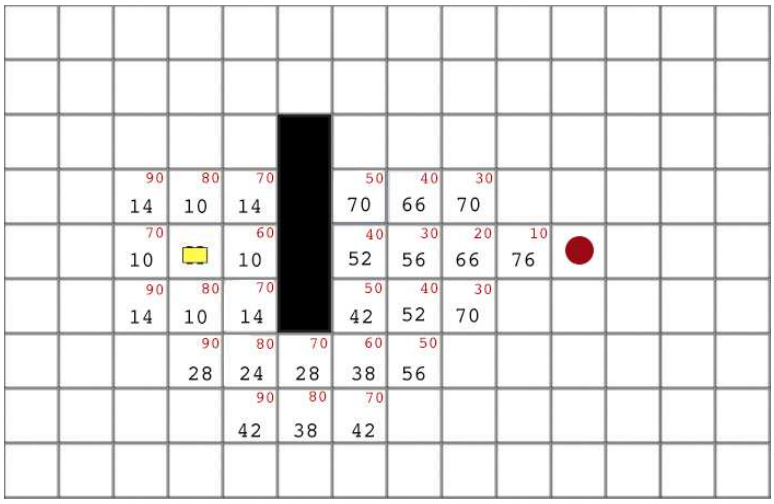

Figure 4.7: The cumulative costs are spread from the starting point to the goal one, expainding always in the tree the node with the smallest value.

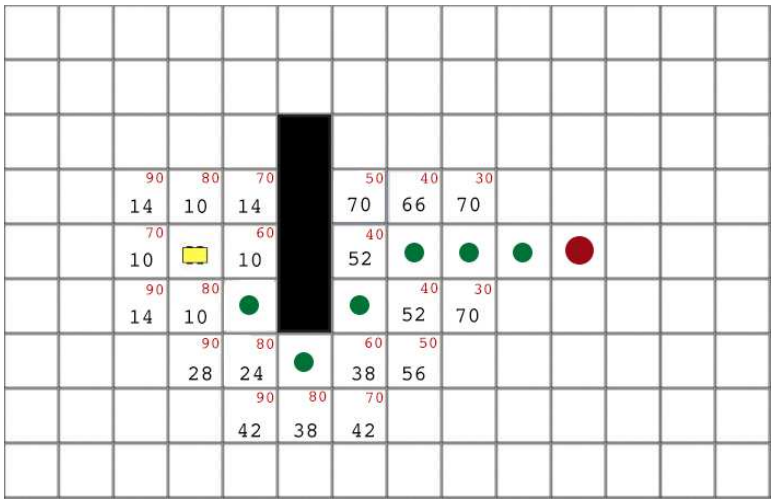

Figure 4.8: The shortest path is found proceeding backwards from the goal and selecting each time the cell with the minimum $f(n)$ value.

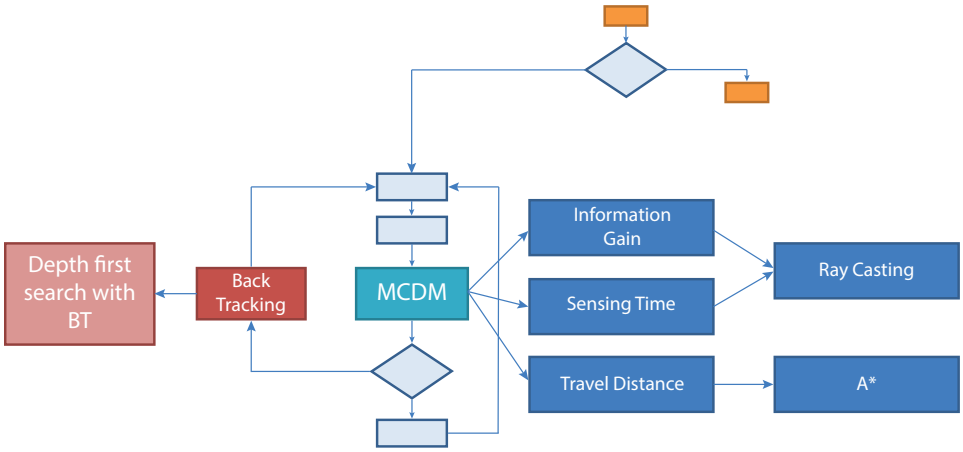

Figure 4.9: The backtracking operations is performed modeling a search problem with a depth-first search algorithm.

Differently from most cases that use DFS to explore a graph, we have not a preassigned graph containing the poses of the robot; every time the robot reaches a new one, we add it in the tree structure adopted as a leaf of the parent node representing the previous pose of the robot. Moreover, the new position is added to a closed list to prevent the robot to reach it another time in the future. In correspondence of each pose we also save all the other frontiers identified by the robot with it. The structure we obtain is represented in Figure 4.10.

The exploration phase takes place in this way: as said, we build a tree in depth adding each time the new pose as a child of the previous one; to each node is associated a list called *nearCandidate* of frontiers visible from that pose. This list, initially empty, is filled every time the robot reaches a new position with those frontiers that provide new information about the map. When the robot can not find further position in front of it, it evaluates all those frontiers inside the nearCandidate record. After having possibly reached all of them and followed new paths, it performs a backtracking operation to the upper level of the tree. These steps, reported in Figure 4.11, are repeated until the coverage is obtained or the robot reaches the initial pose without completing the task.

If we image the environment map as a graph in which each free cell is a node, the only possibility to not explore entirely this graph is that it's not *connected*. In graph theory, an undirected graph is called connected when there is a path between every pair of vertices. In a connected graph, there are no unreachable vertices.
Given this definition it is obvious that it is impossible to fully explore a map

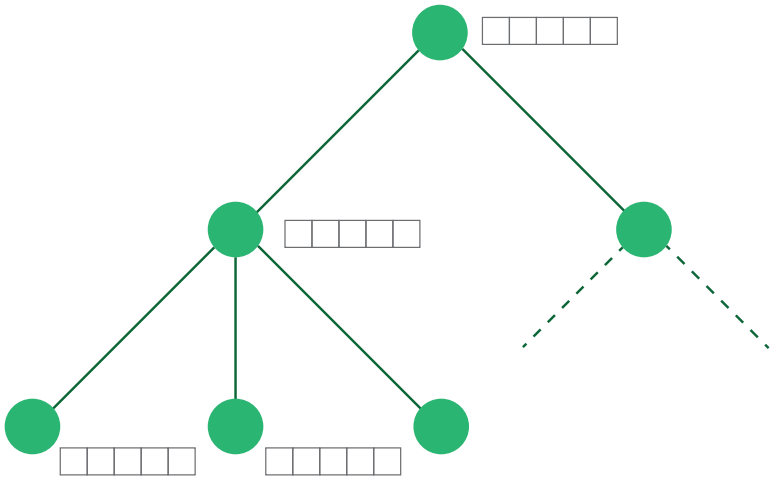

Figure 4.10: Structure of the tree built during the navigation. Every time the robot reach a new position, this is add in the free with a list of all frontiers visible from there

only if there are some free and unreachable cells, maybe because the access to them is prevented by the ostacles. This is true both in simulation and in the real scenario.

4.3 ROS Implementation

We implemented the Next-Best-Smell algorithm on the *Robot Operating System* (ROS) platform [2] to run it on the robot platform used during our tests. In this section we provide only a brief overview of the system, please refer to Appendix A for full details.

The peculiarity of ROS is that every communication between two nodes is realized with an asynchronous system based on the $publish - subscribe$ paradigm. However, if needed it, there is also the possibility to use synchronous calls (through the *service* oriented system).

Our algorithm, implemented in the *mcdm_framework* node, subscribes to *move_base/global_costmap/costmap* topic as shown in Listing 4.3, provided by the *map_base* node, to get the *costmap_2d* of the environment in which the robot is located. A costmap is a map in which the obstacles are inflated by a user defined radius. For our navigation purposes, this fact prevents to choose as goal a pose too close to walls or obstacles, in a way the robot can't reach it due to its kinematics or physical constraints.

After the Next-Best-Smell algorithm identifies a new pose, a *MoveBaseGoal* message is created and sent to *move_base* node (Listing 4.4). This node provides an implementation of an action (*actionlib*) that, given a goal

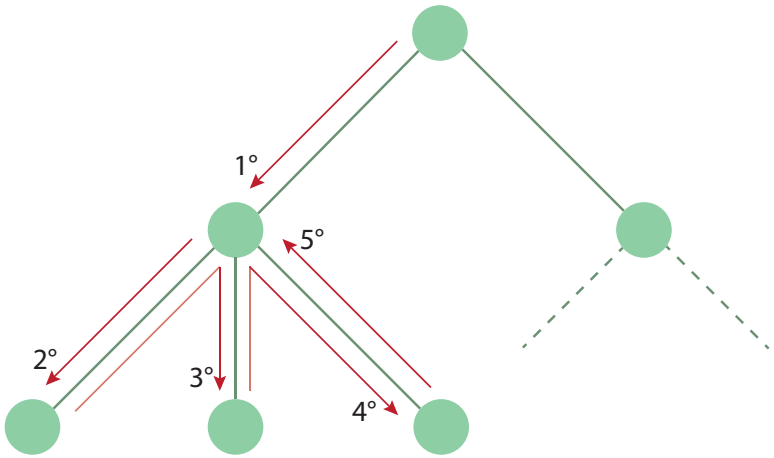

Figure 4.11: Every time the robot can not find any further frontiers in front of it, it comes back to the previous pose and try to expand the corresponding node towards a new direction

Listing 4.3: Subscription ot map_server node to the the global_costmap of the environment

```
if (map_service_client_.call(srv_map)){

        costmap_sub = nh.subscribe<nav_msgs::OccupancyGrid>("move_base
            /global_costmap/costmap", 100, grid_callback);
        costmap_update_sub = nh.subscribe<map_msgs::
            OccupancyGridUpdate>("move_base/global_costmap/
            costmap_updates", 10, update_callback);

        if(costmapReceived == 0) {
            ROS_INFO_STREAM( "waiting for costmap" << std::endl);
        }

        if(costmapReceived == 1){
                // all the code is inside this block
        }

    sleep(1);
    ros::spinOnce();
    r.sleep();

}
```

Listing 4.4: Sensing a MoveBaseGoal message to move the robot to the next pose

```
move_base_msgs::MoveBaseGoal goal;
double orientZ = (double)(target.getOrientation()* PI/(2*180));
double orientW =  (double)(target.getOrientation()* PI/(2 * 180));
move(p.point.x ,p.point.y, sin(orientZ), cos(orientW));

//-----------------------------------------

void move(int x, int y, double orZ, double orW){
    move_base_msgs::MoveBaseGoal goal;

    MoveBaseClient ac ("move_base", true);
    goal.target_pose.header.frame_id = "map";
    goal.target_pose.header.stamp = ros::Time::now();

    goal.target_pose.pose.position.x = x;
    goal.target_pose.pose.position.y = y;
    goal.target_pose.pose.orientation.z = orZ;
    goal.target_pose.pose.orientation.w = orW;

    ROS_INFO("Sending␣goal");
    ac.sendGoal(goal);

    ac.waitForResult();  //the execution flow is stopped until when
        the robot reaches the new pose

    if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
        ROS_INFO("I'm␣moving...");
    else
        ROS_INFO("The␣base␣failed␣to␣move");
}
```

in the world, will attempt to reach it with a mobile base. The *move_base* node links together a global and local planner to accomplish its global navigation task.

When the robot reaches a new pose in the map, the next step is performing a scanning operation. Listings 4.5 and 4.6 show a parallel blocking thread that subscribes to the */ptu_control/state* topic to get the actual position of the TDLAS gas sensor and then invokes the */ptu_control/sweep* service to perform the sweep.

Listing 4.5: Retrieving the exact pose of the pan-tilt unit before performing a scanning operation

```
void scanning(){
    ros::NodeHandle nh("~");
    ros::Subscriber ptu_sub;
    ptu_sub = nh.subscribe<std_msgs::Int16>("/ptu_control/state",100,
        stateCallback);
    ros::AsyncSpinner spinner(0);
    spinner.start();
    auto start = chrono::high_resolution_clock::now();
    gasDetection();     // call the proper method to perform a
        scanning operation
    while(ros::ok()){

        while(statusPTU!=3){    // check if the scanning operation has
             started
            sleep(1);
            //ROS_INFO("PTU status is...%d",statusPTU);
        }
        ROS_INFO("Scanning started!");
        ros::WallDuration(5).sleep();
        while(statusPTU!=0){    // check if the scanning operation is
            still running
            sleep(1);
            //ROS_INFO("PTU status is...%d",statusPTU);
        }

        ROS_INFO("Gas detection COMPLETED!");
        auto end = chrono::high_resolution_clock::now();
        double tmpScanning = chrono::duration<double,milli>(end -start
            ).count();
        timeOfScanning = timeOfScanning + tmpScanning;
        spinner.stop();
        break;
    }

}
```

Listing 4.6: Sending the right parameter to ptu_control node to perform a scanning operation

```
void gasDetection(){

        ros::NodeHandle n;
        ros::ServiceClient client1 = n.serviceClient<ptu_control::
            commandSweep>("/ptu_control/sweep");
        ptu_control::commandSweep srvSweep;

        if(min_pan_angle > max_pan_angle){
            double tmp = min_pan_angle;
            min_pan_angle = max_pan_angle;
            max_pan_angle = tmp;
        }

        srvSweep.request.min_pan        = min_pan_angle;
        srvSweep.request.max_pan        = max_pan_angle;
        srvSweep.request.min_tilt       = tilt_angle;
        srvSweep.request.max_tilt       = tilt_angle;
        srvSweep.request.n_pan          = num_pan_sweeps;
        srvSweep.request.n_tilt         = num_tilt_sweeps;
        srvSweep.request.samp_delay     = sample_delay;

        if (client1.call(srvSweep)){
            ROS_INFO("Gas␣detection␣in␣progress␣...␣<%.2f~%.2f,%.2f>",
                min_pan_angle,max_pan_angle,tilt_angle);
        }else{
            ROS_ERROR("Failed␣to␣initialize␣gas␣scanning.");
        }

}
```

Chapter 5

Experimental evaluation

In this chapter we describe the results we obtained in simulation and in the real world, testing the Nexy-Best-Smell approach on the Husky A200 platform. In the end we will make a comparison with the work developed by Asif Arain *et al.* [6]. To do so, we will introduce first of all the parameters required by our algorithm and the evaluation metrics that measure the total execution time. Moreover we will investigate and explain how the final outcome may vary according to different values of the parameters introduced.

5.1 Parameters and Evaluation Metrics

In this section we will explain which parameters we consider during the modeling of exploration's problem.

As introduced in Chapter 3, we represent the environment as an occupancy grid after having discretized the map with different resolutions. Each cell in this representation can assume three values according to its status: 0 if the cell is free and unscanned, 1 if it is occupied by an obstacle or 2 if it is free and already scanned by the robot. During the exploration, the robot uses two different grids: one, more coarse, is used for navigation purposes, to avoid situations in which the robot could crash against a wall or an obstacle; another one, finer, is used for keeping track of gas (mark cells as scanned). For this reason the first parameter is the resolution of the navigation map: in our tests we decided to use square cells with side of one meter, but it's possible to use also the same resolution of the scanning map, that usually is higher (20 or 50 centimeters per side).

The second parameter we consider in our work is the range r of the gas sensor, used for scanning purpose. The sensor is able to scan visible cells whose center's distance from the robot is not larger than this value. In our

experiments, depending on the map used, r assumed a value of 10 or 15 meters.

Another important parameter related to the platform used is the maximum scan angle *phi*, already introduced in Chapter 3. As expressed before, the scan angle *phi* defines the sweep that the gas sensor has to perform. Due to the physical limitations introduced by the pan-tilt unit mounted on, the maximum value of this parameter is 180 degrees on the platform used for tests in the real world.

On the environment represented as a grid, the robot can assume a finite number of orientations, either 4 or 8 in our tests, depending if we consider only the main ones (North, East, South and West) or also the derived one (North-East, North-West, South-West and South-East). Usually having more orientations in which a scanning operation can be performed is an advantage for the robot but experiments in simulation demonstrated that this is not always true in the case of greedy algorithms. As it will be explained in subsection 5.4, introducing more orentations means allowing the presence of two or more candidate positions with the same evaluation score. Therefore, from the robot's point of view they can lead to similar path even if they can evolve in complete different ways due to the presence of obstacles in the map. Because our algorithm is based on the Next-Best-View concept, we cannot establish which is the best position (among the ones with the same rank) minimizing the exploration time, and therefore it is impossible to solve this conflict without negating the hypothesis of the greedy approach.

The last parameter the user can set is the threshold value used during the pruning operation. As mentioned in Chapter 4, we model the entire exploration as a search problem, building a tree in which each node is a pose of the robot and whose children are the possible poses reachable from the parent node. In this scenario, a threshold helps the robot to speed up the navigation towards the more interesting poses, whose evaluation score is greater than the pruning value, and discarding the others.

The main feature we analyzed is the total exploration time, that is estimated as the sum of the travel time and the sensing time. While in simulation it is only an estimate of the real one, for experiments in the real world the total exploration time is computed starting from the instant in which the ROS node of the algorithm is launched and ending when the exploration is completed, either successfully or not. Since the exploration time is affected by many factors it was our interest to discover which ones they are and track how they influence it.

The first and most important evaluation metric is represented by the total number of sensing positions, or how many times the robot stops to

perform a sensing operation in order to detect the presence of gas. Directly connected to this, the travel distance, computed with A*, is an estimation of the real path followed by the robot and connecting all the sensing positions starting from its initial pose to the last one. Given that and assuming the robot is moving with costant speed of 0.5 m/s it is easy to compute an approximation of the time spent traveling.

As said before, every time time the robot stops it performs a sensing operation and thanks to ray casting, as described previously in Chapter 4, we are able to compute the exact angle required to observe the unscanned area. The sum of all these angles represents another evaluation metric, used to estimate the total scanning time with a polynomial function.

5.1.1 Criteria Weights

As described in Chapter 3, we modeled the decision related to the next position to reach as a multi-objective optimization problem, in which we considered three criteria: *information gain*, *travel distance* and *sensing time*. The advantage of using the Choquet Fuzzy Integral instead of a different aggregation function is the possibility to model relationships among coalition of the criteria considered. Taking inspiration from Game Theory, we can define a coalition as any subset of criteria. Therefore it is clear that, trying to optimize three criteria and three coalitions among them means solving a 6-dimensional problem.

The goal of our simulated experiments was to identify a set of weights that could guarantee good results in terms of total exploration time. To make it easier we reduced our problem to three dimensions, focusing our attention only to the three single criteria and modeling a slighty synergic relationship among them. In this way we were able to represent this problem in a graphical way: due to the following constrains

$$x_1 + x_2 + x_3 = 1 \tag{5.1}$$

and considering $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$ we can draw a simplex as space of the solutions.
Trying to solve this problem didn't represent a good idea since it is only a rough approximation, without considering the coalitions of more than one criterion, of the real 6-dimensional problem. Therefore we ran a large number of tests, under different conditions and using different maps, and we analized the results obtained.

First of all we identified some points of interest on the simplex surface and we tested their performance. These points are the following ones: the

three vertices, the ortocenter, the middlepoint of each side and six simmetric points beloning to the three bisectors, and all of them are reported in Table 5.1, where the coalitions are built in the following way:

$$Coalition1 = \{informationgain, traveldistance\}$$
$$Coalition2 = \{informationgain, sensingtime\}$$
$$Coalition3 = \{traveldistance, sensingtime\}$$
$$Coalition4 = \{informationgain, traveldistance, sensingtime\}$$

The meaning behind the use of the simplex is the following: if we want to maximize one of the three criteria considered we have to choose an higher value for its weight, and then share what is left between the other two, paying attention to satisfy eq. 5.1.

According to this, if we are strongly interested in scanning as much as possible the available free area despite the distance between the actual position of the robot and the target one, we could assign a weight greater than 0.5 to *information gain*, and then split the rest between *travel distance* and *sensing time*; in the same way we can reason about the other criteria. Of course it is also possible to find a balance among multiple criteria: for example, Configuration F tries to optimize at the same time either the *information gain* and the *travel distance*, while Configuration D assigns equal important to all the criteria.

The tests we completed, as it will be shown in the following sections, identified the upper part of the simplex as the one providing the weights able to guarantee a better performance: giving an high importance to *information gain* seems to be the best choice to speed up the exploration, while assigning also a small weight to the other criteria could improve the convergence speed according to the nature of the environment.

5.2 Simulation

Different and multiple tests were run in simulation to find the best combination of criteria for solving the coverage problem.

The first used the map of Freiburg University [1], discretized in cell of one square meter of resolution, obtaing 6113 free cells.

The results, contained in the Table 5.2, were computed starting always from the same initial pose. As explained in the previous subsection, a better exploration time was obtained in those situations in which *information gain* was the only considered criterion (configuration A) or most part of the power

Configuration	informationGain	travelDistance	sensingTime	Coal.1	Coal.2	Coal.3	Coal.4
A	1	0	0	1	1	0	1
B	0	1	0	1	0	1	1
C	0	0	1	0	1	1	1
D	0.333	0.333	0.333	0.766	0.766	0.766	1
E	0.6	0.2	0.2	0.9	0.9	0.5	1
F	0.428	0.428	0.144	0.956	0.672	0.672	1
G	0.2	0.6	0.2	0.9	0.5	0.9	1
H	0.144	0.428	0.428	0.672	0.672	0.956	1
I	0.2	0.2	0.6	0.5	0.9	0.9	1
L	0.428	0.144	0.428	0.672	0.956	0.672	1
M	0.5	0.5	0	1	0.6	0.6	1
N	0	0.5	0.5	0.6	0.6	1	1
L	0.5	0	0.5	0.6	1	0.6	1

Table 5.1: Criteria configuration - Weights assigned to each criterion and coalitions among them. A coalition is defined as a subset of the criteria involved in the evaluation.

Configuration	Coverage satisfied	Cells visisted	Distance travelled	Travel Time(s)	Number of turning	Sum of scanned angles	Scanning time(s)	Total time(m)
A	yes	107	1417.57	2835.14	121	213.52	6018.28	147.55
B	no	279	2724.81	5449.62	213	202.59	8200	227.49
C	yes	223	2948.04	5896.08	256	157.27	6740.67	210.61
D	yes	162	1701.89	3403.79	148	175.16	6045.3	157.48
E	yes	153	1690.18	3380.36	172	166.20	5853.26	153.89
F	yes	155	1445.13	2890.27	154	166.29	5899.98	146.50
G	yes	169	1612.56	3225.13	162	184.14	6071.27	154.94
H	yes	192	2247.89	4495.79	169	171.53	6505.33	183.35
I	yes	186	1913.23	3826.46	163	169.14	6419.14	170.76
J	yes	169	1832.24	3664.48	162	158.31	6005.91	161.17
K	yes	139	1116.57	2233.14	122	215.32	6566.87	146.66
L	no	207	2398.65	4797.31	190	184.24	7175.16	199.54
M	yes	166	1623.5	3247	173	147.47	5809.77	150.94

Table 5.2: Results for Freiburg University map for each considered configuration. In only two cases among thirteen the exploration was not totally completed.

was splitted between *information gain* e *travel distance* (configurations F and M).
Configuration K represented one of our best results: it estimanted 146,66 minutes as total exploration time, of which 37,21 minutes were used by the robot to move and the others for scanning purpose. Comparing this configuration with Configuration A it's easy to see that the robot stopped more times but the travel time is shorter. This is perfectly fine according to the Next-Best-Smell approach, since in Configuration K we are considering (and therefore we are trying to optimize) also the *travel distance*: in this way the robot, between two candidate positions with the same value as *information gain* it selected the closer one, minimizing the distance between its current position and the goal.

Figure 5.1 shows all the sensing position reached by the robot in Configuration K, which are all the positions in which the robot stopped and performed scanning operations. Differently, in Figure 5.2 we can observe that the robot, in the same configuaration, explored 5000 cells, corresponding to 81,6 % of the total number (6113), in 74 round (a round is one iteration of the algorithm), a bit more than half of the final number (139). This behaviour is expected since the greedy approach tends to maximize

Figure 5.1: Sensing positions identified during the exploration of the Freiburg University's map using Configuration K, 15 meters for the gas sensor range and 180 degrees for the scan angle ϕ.

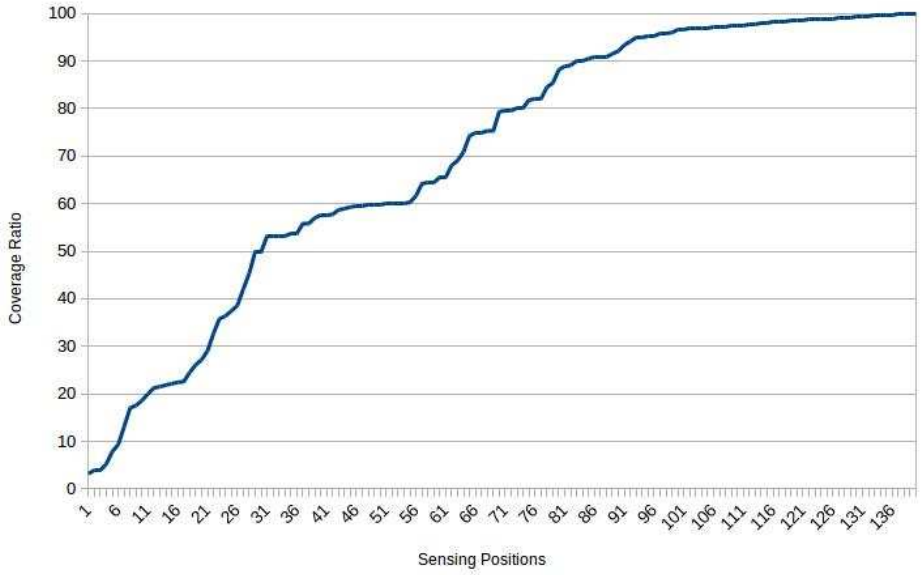

Figure 5.2: Coverage ratio during the exploration of the Freiburg University's map using Configuration K, 15 meters for the gas sensor range and 180 degrees for the scan angle ϕ.

always towards the local optimum, discarding those positions with a small *information gain* that in most of the cases correspond to corners. For this reason the robot scanned most of the environment from relatively few positions but in the end it took the same time to discover those unkown cells that it had been skipped before.

Other tests were run on the Teknikhuset's corridor's map, the corridor immediately outside the Mobile Olfaction Lab. The results are reported in Table 5.3. After having built it with the physical robot, the map was purged from unreachable cells behind glass surfaces and it was discretized in a grid whose cells were half square meter. The result, shown in Figure 5.3, counted 916 free cells.

The first thing that we discovered analizing these results is the fact that now the best configurations are not only the ones that maximize the *information gain* but also those that take care about the *sensing time*. This is because this map has tigher spaces therefore using a smaller angle to perform a sensing operation is more important than in an open environment as in the Freiburg example. Considering the best result obtained with configuration M, Figure 5.4 shows, as before with the Freiburg map, that our algorithm allowed the robot to scan 746 cells out of 916, corresponding to 81,44 %, in just nine positions, actually 37,5 % of sensing positions final number.

Configuration	Coverage satisfied	Cells visisted	Distance travelled	Travel Time(s)	Number of turning	Sum of scanned angles	Scanning time(s)	Total time(m)
A	yes	20	276.96	553.93	23	44.42	1192.44	29.10
B	yes	68	251.88	503.76	22	42.73	1596.41	35.00
C	yes	42	331.24	662.49	28	26.63	1211.39	31.23
D	yes	42	356.93	713.87	27	32.31	1324.77	33.97
E	yes	27	215.22	430.44	23	34.35	1119.92	25.83
F	yes	38	267.29	534.59	29	32.96	1288.89	30.39
G	yes	39	633.24	1266.49	39	31.68	1258.5	42.08
H	yes	40	472.31	944.63	34	30.04	1227.58	36.20
I	yes	36	275.01	550.02	24	26.51	1067.79	26.96
J	yes	37	242.75	485.51	29	25.66	1135.95	27.02
K	yes	28	240.21	480.42	20	43.43	1309.37	29.82
L	yes	40	447.99	895.99	34	29.50	1220.71	35.27
M	yes	24	223.08	446.16	21	34.47	1073.27	25.32

Table 5.3: Results for Teknikhuset corridor map for each considered configuration. The full coverage was completed in all the tests.

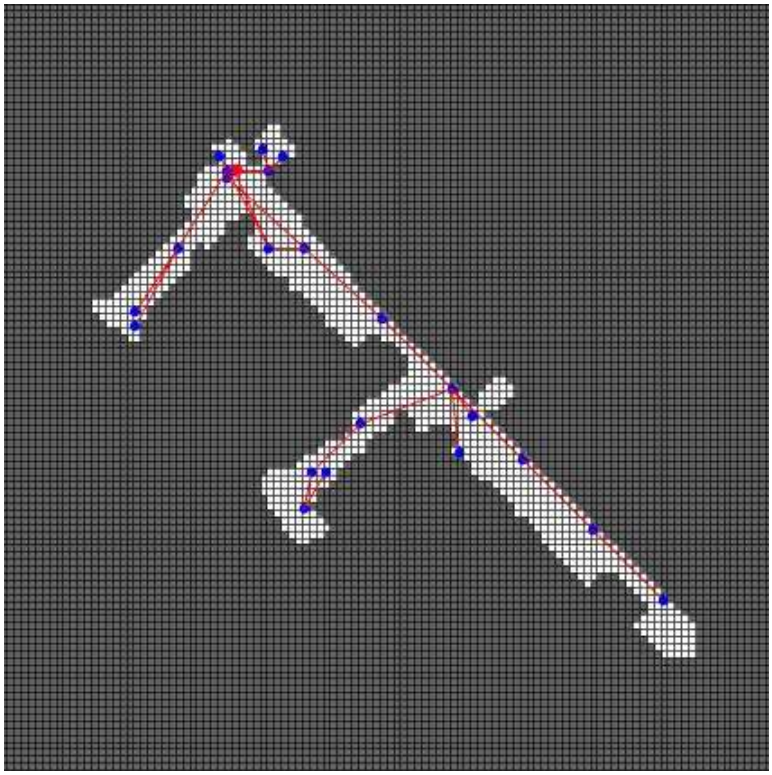

Figure 5.3: Sensing positions identified during the exploration of the Teknikhuset corridor's map using Configuration M, 15 meters for the gas sensor range and 180 degrees for the scan angle ϕ.

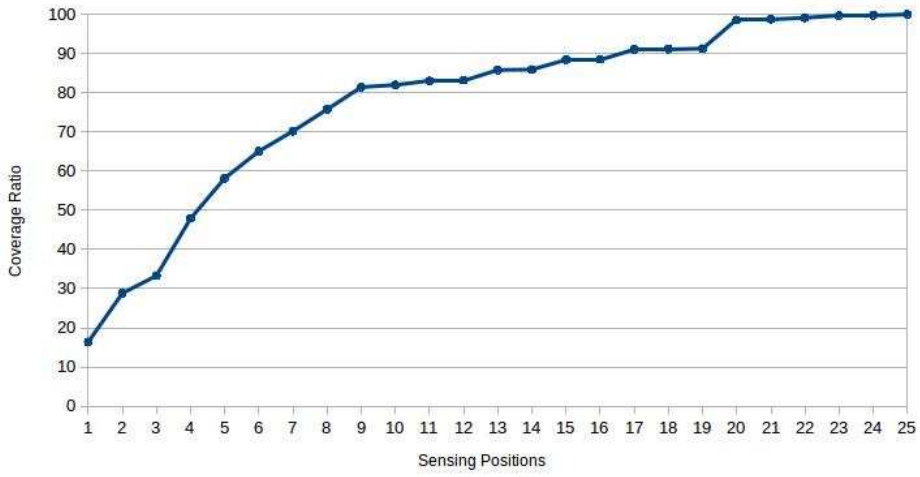

Figure 5.4: Coverage ratio during the exploration of the Teknikhuset corridor's map using Configuration M, 15 meters for the gas sensor range and 180 degrees for the scan angle ϕ.

5.3 Real World Experiments

The Next-Best-Smell algorithm was tested also on a physical platform: the Husky A200 reported in Figure 5.5, produced by Clearpath Robotics.

About the Husky, on the base platform we mounted a Light Detection and Ranging (LIDAR) sensor to determine the object's distance from the robot using a laser pulse. Then we mounted also a pan-tilt unit that provided accurate real-time positioning of the gas sensor and, above it, a Remote Methane Leak Detector (RMLD) using a tunable infrared diode for open path optical absorption specific to methane. To simulate methane leaks in the environment we filled some bottles with the gas and we put them in the corridor or behind some corners.

Differently from the simulated experiments, we had to use another map of the Teknikhuset's corridor, obtained from the one used before but cleaned from those spaces impossible to reach due to the kinematics limits of the Husky robot, as those under the stairs or behind the tables presented in the environment. Therefore the grid resulted after the discretization in cells of fifty centimeters per side has less free cells than the previous one, 896 instead of 916. The robot completed the full exploration in 27.72 minutes; the sensing positions reached are reported in Figure 5.6 as blue dot, while the yellow ones are the bottles filled with methane.

Despite our choice of using NDT-MCL localization [24] that should be more robust than other techniques, during this experiment we had some

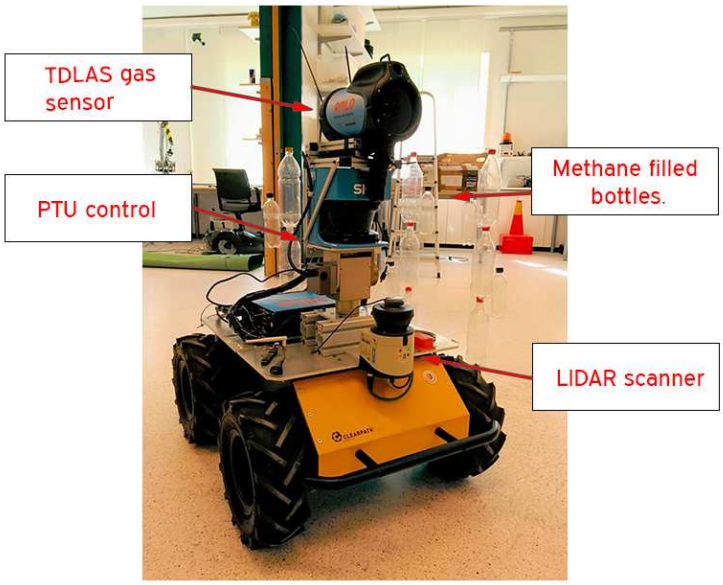

Figure 5.5: Husky A200 (Clearpath Robotics) equipped with a Lidar scanner, a pan-tilt unit and a TDLAS sensor for remote gas detection. Methane leaks are simulated in the environment with some bottles filled with the gas

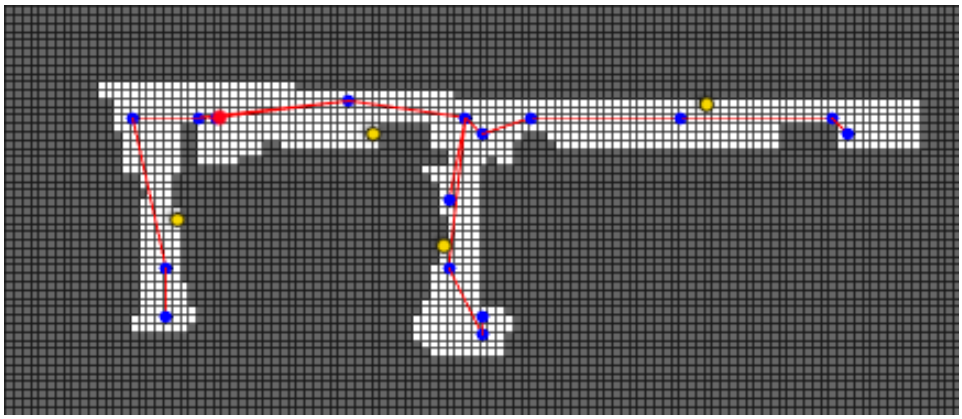

Figure 5.6: Sensing positions obtained during the experiment in the real world using 10 meters for the gas sensor range and 180 degrees for the scan angle ϕ.

	Real	Simulation
Result	894/894	896/896
Total cells visited	15	15
Total (est.) travelled distance (cells)	184.04	218.44
Total (est.) travel time	368.08	436.89
Total number of turning	12	13
Sum of scan angles (radiants)	30.64	33.11
Total (est.) scanning time	860.74	880.70
Total estimated time (min)	20.48	21.96
Total real time (min)	27.72	—

Table 5.4: Comparison between the results obtained with the experiment in the real world and the ones in simulations, using the same configuration and the same map. The different number of total cells depend on the discretization process inside ROS.

issues due to the human presence in the environment that affected the total exploration time; however the robot was able to complete the task in a reasonable time. We run a test with the same configuration also in simulation and we compared the result, which are collected in Table 5.4.

The first thing that it is important to notice is that the total number of cells in the two experiments is different regardless the map is the same: this is due to the discretization process inside ROS, that use two threshold values to establish if a cells is free or unknown. For this reason, it could happen that a pixel is considered in different way in simulation and in ROS if not completely white or black.

The second and most important thing is the total time required for the exploration of the map: as expected the real time was greater than the estimated one but the gap between the two was affected, as anticipated, by some localization problem that can often happen working with a physical platform . Moreover the driver of the pan-tilt unit we used was affected by a bug that made it perform randomly two sweeps instead of one and this represented, of course, the biggest contribution in making the real experiments slower than the simulated one.

Apart from these problems, the robot was still able to scan the four simulated gas leaks, even if the methane filled bottles were placed in quite hidden places like corners or behind obstacles.

5.4 Reflection on Experimental Results

After having run many test, we collected some interesting results that allow us to understand in a better way the convergence of the Next-Best-Smell algorithm.

The first comment is related to the number of possible orientations the robot can assume. Due to the greedy nature of this approach, it could happen that the more directions we use, the higher the probability that two or more frontiers have the same evaluation score: the robot will choose the one pushed as first in the data structure containing all the candidate positions. This fact can lead to worse performances considering the total exploration time, because two paths that seem similar at the beginning (from an evaluation score's point of view) can evolve in complete different ways due to the presence of obstacles in the map. Because we cannot look forward to establish which is the best orientation to choose to minimize the exploration time, it is impossible to solve this conflict without negating the hypothesis of the greedy approach.

In Table 5.5, the results obtained with the Freiburg University's map (discretized in cells of one meter per size, whose 6113 are free) using four and eight orientations are reported: differently from what could be expected, in nine tests out of thirteen, using four orientations resulted in a better solution, despite in two occasions the map was not completely explored. However this is not always true, as an experiment with another map, the one of Teknikhuset's corridor, shows. In this case using eight directions instead of four speeds up the convergence of our approach in nine case out of thirteen. Therefore we could say that this behaviour exists but it is strongly related to the nature of the map, for example the corridor's disposition and the amount of obstacles present in it. In situations like in the Freiburg example, where there are large and empty spaces, four orientations are sufficient to obtain good results because more directions could have similar evaluation score. Differently, in environments with stricht spaces as in the Teknikhuset's corridor, more orientations allows the robot to follow straight paths in a way parallel to the walls, minimizing the travel time and therefore the total exploration time.

A second interesting aspect is related to the pruning threshold used during the frontiers evaluation. Our tests, reported in Table 5.6 and Table 5.7 demonstrated that using a threshold can increase the convergence speed only in the case of a non-complete coverage ($<100\%$). However, for every value of coverage we want to satisfy, a threshold value that is too high can

	Freiburg University		Teknikhuset Corridor	
Configuration	Total time(m) 4 dir	Total time(m) 8 dir	Total time(m) 4 dir	Total time(m) 8 dir
A	147.55	134.53	29.10	25.25
B	227.49	227.01	35.00	44.62
C	210.61	155.02	31.23	28.07
D	157.48	169.79	33.97	29.78
E	153.89	199.40	25.83	22.27
F	146.50	146.65	30.39	31.62
G	154.94	201.38	42.08	33.19
H	183.35	207.13	36.20	27.98
I	170.76	229.26	26.96	22.15
J	161.17	178.45	27.02	25.92
K	146.66	208.48	29.82	31.99
L	199.54	157.04	35.27	24.45
M	150.94	154.42	25.32	26.11

Table 5.5: Having 8 directions instead of 4 among to choose can represent either an advantage (with tight spaces like in the Teknikhuset's corridor, on the right) either a disadvantage (with open spaces like in the Freiburg example, on the left).

Value	Coverage reached	Cells visited	Exploration time(m)
0	100	101	172.29
0.01	99.32	78	129.70
0.02	98.9	69	113.87
0.03	98.02	61	103.32
0.04	97.52	56	95.68
0.05	97.51	55	94.35
0.06	96.89	51	87.11
0.07	96.56	49	82.44
0.08	95,12	46	79.05
0.09	94.24	43	73.16
0.1	93.06	39	67.25
0.12	93.09	38	64.05
0.13	91.18	35	59.81
0.15	90.44	34	57.98
0.2	86.84	30	51.34

Table 5.6: The adoption of a pruning factor speeds up the convergence of our algorithm but decrease the total amount of area we can cover.

	Thresholds								
	0		0.01		0.025		0.04		
Coverage	Satisfied	Cells visited	Satisfied	Cells visited	Satisfied	Cells visited	Satisfied	Cells visited	Total cells
100	V	101	X	78	X	64	X	56	6113
99	V	94	V	76	X	64	X	56	6051
98	V	92	V	74	V	64	X	56	5991
97	V	85	V	72	V	61	V	56	5930
96	V	79	V	65	V	59	V	54	5869
95	V	73	V	63	V	57	V	51	5808
94	V	72	V	59	V	53	V	49	5747
93	V	71	V	58	V	49	V	46	5686
92	V	71	V	58	V	48	V	44	5624
91	V	70	V	57	V	48	V	43	5563
90	V	59	V	53	V	47	V	42	5502

Table 5.7: Assigned a coverage constraint to satisfy, an higher pruning factor can speed up the convergence speed of Next-Best-Smell, but in some cases (if a full coverage is required) it could prevent to complete the task.

prevent to reach our goal, and this is particularly true if we want to cover the entire map.

The experience related to the simulated experiments helped us to understand the behaviour of the weights configurations and to suggest which to use according to the nature of the map. From the results obtained we can conclude that the choice of the configuration is strongly related to the structure of the environment. In situations like in the Freiburg example, with large and empty spaces, configurations that privilege travel distance over sensing time (F, G, K) are preferred because it is highly probable that scanning will be carried out with maximum opening angle and rendering the sensing time criterion less meaningful. On the other hand, in environments with tight spaces as in the Teknikhuset's corridor, it is more convenient that configurations take care about the sensing time (I, J, K, M) because using only the sufficient angle required for perceiving a new portion of the map can improve significantly the performance.

In conclusion, we found *information gain* to be the most importan criterion, since maximizing it means focusing mainly on exploring new cells. For this reason, Configurations A, E and L that assign large weight to information gain are often a good choice. These experiments identify the upper part of the simplex reported in Figure 5.7 as the one representing the weights able to obtain a better performance. Giving an high importance to information gain is most important to speed up the exploration, and assigning also a non-zero but smaller weight to the other criteria could improve the convergence speed depending on the particular map of the environment.

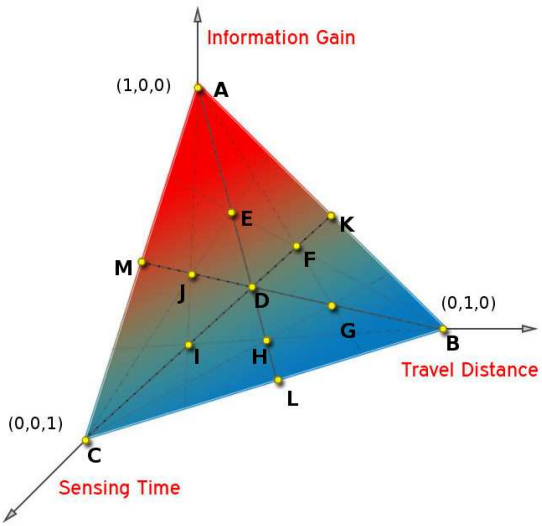

Figure 5.7: The weights assigned to the three criteria are shown on the simplex surface. Experimental activities demonstrated the the upper part of the simplex represent the locus of point offering a better overall performance.

5.5 Comparison with Offline Approach

In the end we made a comparison between the Next-Best-Smell algorithm and the one developed by Asif Arain *et al.*, explained in [6]. Briefly, they solved the exploration problem in two stages: first of all the so called *Art Gallery Problem* in a given map, to identify those positions from which the robot can see all the environment; after this, the *Travelling Salesman Problem* among these position to find the shortest path connecting all of them is solved. Since this approach consists in an optimization problem with an high computational effort, the solution was computed offline using Matlab and then the list of positions to reach was provided to the robot.

The comparison has been carried out in simulation using the Teknikhuset's corridor's map, and results are reported in Table 5.8.

In his experiments, Arain *et al.* used a configuration with 90 degrees for the scan angle ϕ and a range of 10 meters for the laser sensor, with 4 possible orientations for sensing operations, obtaining a total of 17 sensing positions and an estimated total exploration time of 18.35 minutes. The sensing position are reported in Figure 5.8.

Using Configuration E and the same parameters, our algorithm completes in 40 sensing positions, with an estimated total exploration time of 27.78 minutes. The sensing position are reported in Figure 5.9

If we change our configuration by using a scan angle ϕ of 180 degrees

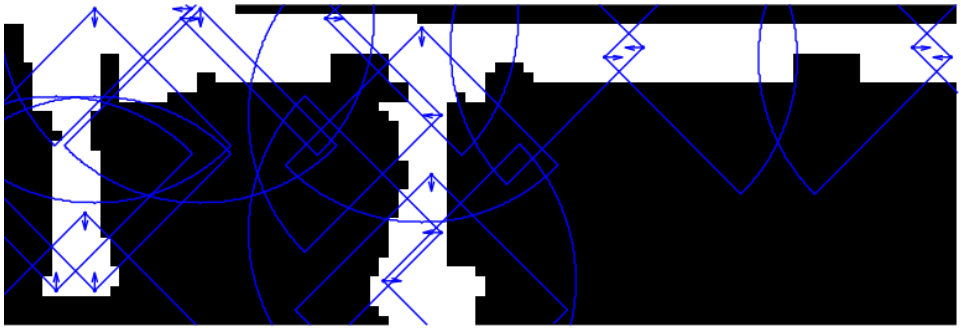

Figure 5.8: Sensing positions obtained with the offline algorithm using only 4 directions, 90 degrees for the scan angle ϕ and a gas sensor range of 10 meters.

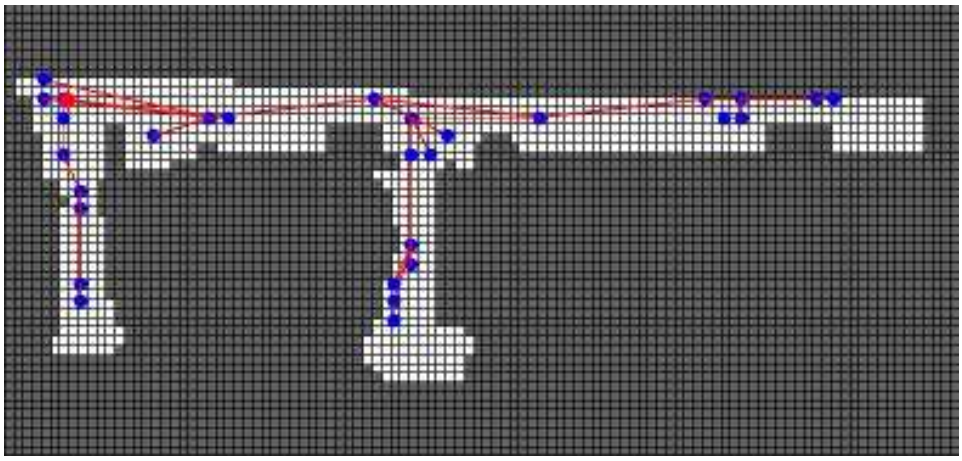

Figure 5.9: Sensing positions obtained with the online algorithm using only 4 directions, 90 degrees for the scan angle ϕ and a gas sensor range of 10 meters.

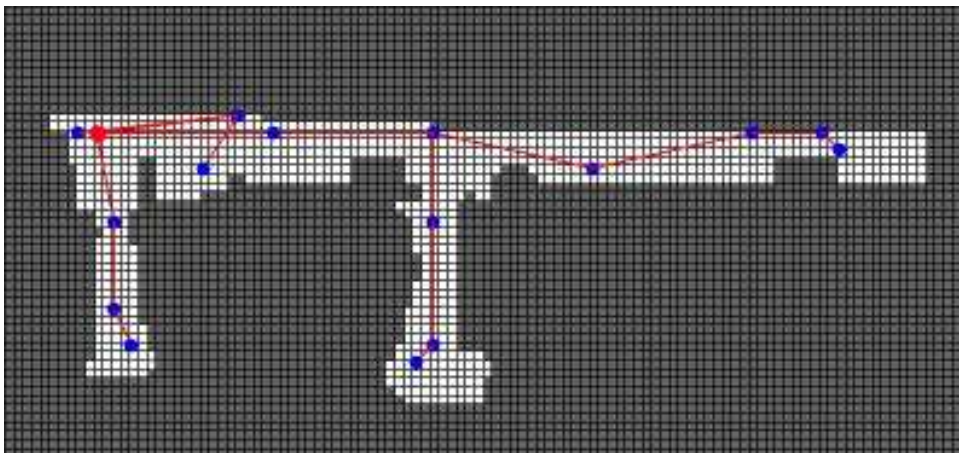

Figure 5.10: Sensing positions obtained with the online algorithm using 8 directions, 180 degrees for the scan angle ϕ and a gas sensor range of 10 meters.

	Offline	Online 1	Online 2
Sensing positions	17	40	20
Travel time (s)	472	519	424
Scanning time (s)	629	1146	816
Exploration time (m)	18.35	27.78	20.68

Table 5.8: Comparison between offline and online approach. Two configurations are reported for the online: 4 orientations with 90 degrees for the scan angle ϕ for the first and 8 orientations with 180 degrees for ϕ for the second.

and 8 possible orientations, the Next-Best-Smell algorithm performs much closer to the offline one: a total of 20 sensing positions with an estimated exploration time of 20.68 minutes, as shown in Figure 5.10.

Differently from the offline algorithm, in which adopting 8 orientations instead of 4 could cause the explosion of the state space, using a machine equipped with an Intel Core i5-3570K with a clock speed of 3.40Ghz and 8 GB of memory, running a test on the corridor's map with 4 orientations takes 0.132 seconds, while a test with 8 orientations takes 0.136 seconds.

We can obtain results really close to the offline algorithm because in [6] the angle ϕ used for sensing operations assumes the value ϕ_{max} for all the exploration, while in our solution ϕ_{max} is simply a boundary on the scanning angle, which is chosen dynamically at each operation.

For this reason, and also considering how sensing operations are the most time-consuming operation in the task of interest, while a broader scan angle ϕ might end up slowing the exploration in [6], it speeds up greatly the exploration with our approach.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

Mobile robotics is becoming an effective tool to solve tasks which are dangerous for humans, such as the one of gas detection.

Robotic olfaction has been attracting more and more interest in the last years, especially since the adoption of TDLAS sensors, which can let us move from stationary sensor networks to gas detection with mobile robots.

In this thesis we presented a Multi Criteria Decision Making (MCDM) approach to define a Next-Best-View (NBV) exploration strategy in order to detect the presence of gas in large environments by means of a TDLAS sensor.

Our solution adapts exploration techniques already used for tasks such as mapping and search and rescue to the gas detection problem, and works by choosing at each step the next best sensing pose by combining the values of three different criteria into a global utility function, and then maximizing its value.

The global utility function, generated using the Choquet fuzzy integral, takes into account the interactions among criteria through the relationships of synergy and redundancy, and by tuning the weights given to the sets of criteria it is possible to adapt the exploration to the interests of the user. In our experiments we tried to tune the parameters in order to minimize the total time required for the exploration.

Experimental activity both in simulation and in the real world has shown the potential of the proposed approach in defining an effective exploration strategy for the task of interest, especially when the complete map coverage

is not necessary.

Our Next-Best-Smell algorithm has been tested mainly in known environment, but several tests have also been run in simulation without giving the robot any a priori information about the environment, with promising results. All of our assumptions still hold in the unknown evironment case, and the exploration is succesfully completed with results being 0 to 10% worse than the one obtained in the known environment, depending on the starting position of the robots and the values of the parameters set.

6.2 Future Works

We identified several possible ways to expand our work.

First of all, some more or different criteria could be used, such as taking into account the presence of gas found in a sensing operation in order to explore the areas which have a higher probability of presenting a gas leakage first. Moreover, a dynamic set of weights for the criteria might be added, in order to change the behavior of the robot depending for example on the level of charge of the battery, or the presence of gas in the environment.

More interestingly, we have identified two main area of interests for future works: unknown environments and multi-robot systems.

As already mentioned, tests have been run in simulation without a priori knowledge about the environment with promising results. The first and most interesting task to perform would be to test the algorithm in unknown environments in the real world.

We also believe that multi-robot systems might be an interesting direction for the future, either by working with omogeneus robots with TDLAS sensors, or even with an etherogenous group of robots, mounting either TDLAS sensors or in-situ sensors, and coordinating their actions in order to obtain a faster coverage of the environment.

Bibliography

[1] https://www.openslam.org/gmapping.html.

[2] www.ros.org.

[3] F. Amigoni and N. Basilico. Exploration strategies based on multi- criteria decision making for searching environments in rescue operations. *Autonomous Robots*, 31(4):401–417, November 2011.

[4] F. Amigoni, V. Caglioti, and U. Galtarossa. A mobile robot mapping system with an information-based exploration strategy. In *Proceedings of International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 71–78, 2004.

[5] F. Amigoni and A. Gallo. A multi-objective exploration strategy for mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 3861–3866, 2005.

[6] A. Arain, M. Trincavelli, M. Cirillo, E. Schaffernicht, and A.J. Lilienthal. Global coverage measurement planning strategies for mobile robots equipped with a remote gas sensor. *Sensors*, 15(3):6845–6871, 2015.

[7] W. Baetz, A. Kroll, and G. Bonow. Mobile robots with active ir-optical sensing for remote gas detection and source localization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2773–2778, 2009.

[8] V. Hernandex Bennetts, E. Schaffernicht ans T. Stoyanov, A. Lilienthal, and M. Trincavelli. M. robot assisted gas tomography - localizing methane leaks in outdoor environments. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 6362–6367, 2014.

[9] V. Hernandez Bennetts, A.J. Lilienthal, A.A. Khaliq, V.P. SesÃ¨, and M. Trincavelli. Towards real-world gas distribution mapping and leak localization using a mobile robot with 3d and remote gas sensing capabilities. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2335–2340, 2013.

[10] P. Biber and W. Strasser. The normal distributions transform: a new approach to laser scan matching, 2003.

[11] G. Bonow and A. Kroll. Gas leak localization in industrial environments using a tdlas-based remote gas sensor and autonomous mobile robot with the tri-max method. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 987–992, 2013.

[12] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1322 – 1328, 1999.

[13] E. W. Dijkstra. A note on two problems in connection with graphs. *Numer. Math.*, 1:269–271, 1959.

[14] M.B. Frish, R.T. Wainner, B.D. Green, M.C. Laderer, and M-G. Allen. Standoff gas leak detectors based on tunable diode laser absorption spectroscopy. In *SPIE Optics East*, 2005.

[15] H. Gonzáles-Bños and J.-C. Latombe. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research*, 21(10-11):829–848, 2002.

[16] M. Grabish. The application of fuzzy integrals in multicriteria decision making. *European Journal of Operational Research*, 89(3):445–456, 1996.

[17] M. Grabish and C. Labreuche. A decade of application of the choquet and sugeno integrals in multi-criteria decision aid. *4OR A Quarterly Journal of Operations Research*, 6(1):1–44, 2008.

[18] M. Grabish, T. Murofushi, M. Sugeno, and J. Kacprzyk. *Fuzzy Measures and Integrals. Theory and Applications.* Physica Verlag, 2000.

[19] P. E. Hart and N.J. Nilssonand B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics SSC4*, volume 4, pages 100–107, 1968.

[20] H. Ishida, Y. Wada, and H. Matsukura. Chemical sensing in robotic applications: A review. *EEE Sens. J.*, 12:3163–3173, 2012.

[21] G. Kowadlo and R.A. Russell. Robot odor localization: A taxonomy and survey. *Int. J. Robot. Res.*, pages 869–894, 2008.

[22] M. Lackner. Tunable diode laser absorption spectroscopy (tdlas) in the process industries- a review. In *Rev. Chem. Eng*, volume 3, pages 65–147, 2007.

[23] Z. Liu, S. Cheng, S. Hu, D. Zhang, and H. Ning. A survey on gas sensing technology. *Sensors*, 12:9635–9665, 2012.

[24] J. Saarinen, H. Andreasson, T. Stoyanov, and A. Lilienthal. Normal distributions transform monte-carlo localization (ndr-mcl). In *EEE/RSJ International Conference on Intelligent Robots and Systems*, pages 382–389, 2013.

[25] S. Soldan, J. Welle, T. Barz, A. Kroll, and D. Schultz. Towards autonomous robotic systems for remote gas leak detection and localization in industrial environments. *Springer Tracts in Advanced Robotics*, 92:233–247, 2014.

[26] P. Sridhar, A. Madni, and M. Jamshidi. Multi-criteria decision making in sensor networks. *IEEE Instrumentation & Measurement Magazine*, 11(1):24–29, 2008.

[27] C. Stachniss and W. Burgard. Exploring unknown environments with mobile robots using coverage maps. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1127–1134, 2003.

[28] Y. Tomioka, A. Takara, and H. Kitazawa. Generation of an optimum patrol course for mobile surveillance camera. *IEEE Trans. Circuits Syst. Video Technol.*, 22:216–224, 2012.

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

[30] M. Trincavelli, M. Reggente, S. Coradeschi, A. Loutfi, H. Ishida, and A. Lilenthal. Towards environmental monitoring with mobile robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2210–2215, 2008.

[31] W. Tsujita, A. Yoshino, H. Ishida, and T. Moriizumi. Gas sensor network for air-pollution monitoring. *Sens. Actuators B: Chem*, 110:304–311, 2005.

[32] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings of IEEE International symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 146–151, 1997.

[33] B. Yamauchi, A. Schultz, W. Adams, and K. Graves. Integrating map learning, localization and planning in a mobile robot. In *Proceedings of Intelligent Control (ISIC)*, pages 331–336, 1998.

Appendix A

Ros Architecture

As described in Chapter 5, we tested our algorithm on the Husky A200 robot, developed by Clearpath Robotics. To do so, we had to implement to code in the *Robot Operating System (ROS)* framework. ROS provides standard operating system services such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor, control, state, planning, actuator and other messages.

At the lowest level, ROS offers a message passing interface that provides inter-process communication and is commonly referred to as a middleware. The ROS middleware provides these facilities:

- publish/subscribe anonymous message passing
- recording and playback of messages
- request/response remote procedure calls
- distributed parameter system

A communication system is often one of the first needs to arise when implementing a new robot application. ROS's built-in and well-tested messaging system saves you time by managing the details of communication between distributed nodes via the anonymous publish/subscribe mechanism. Another benefit of using a message passing system is that it forces you to implement clear interfaces between the nodes in your system, thereby improving encapsulation and promoting code reuse.

The asynchronous nature of publish/subscribe messaging works for many

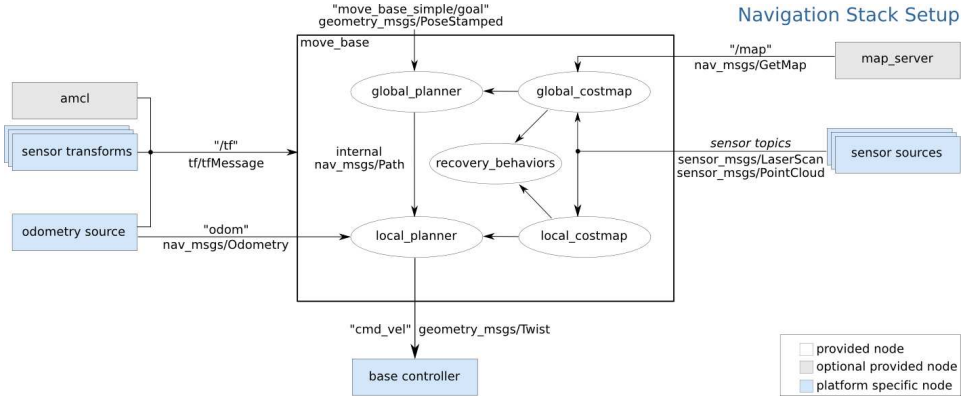

Figure A.1: Navigation stack tf tree (wiki.ros.org)

communication needs in robotics, but sometimes you want synchronous request/response interactions between processes. The ROS middleware provides this capability using services.

A.1 Navigation stack

The first operation to realize with a mobile robot is implement a navigation stack that allows the robot to move and to localize itself in the surrounding environment. The diagram in Figure A.1 shows an overview of how the robot should be configured to run. The white components are required components that are already implemented, the gray components are optional components that are already implemented, and the blue components must be created for each robot platform. The pre-requisites of the navigation stack are the following:

- Transform configuration: the navigation stack requires that the robot be publishing information about the relationships between coordinate frames using tf;
- Sensor information: the navigation stack uses information from sensors to avoid obstacles in the world, it assumes that these sensors are publishing either $sensor\_msgs/LaserScan$ or $sensor\_msgs/PointCloud$ messages over ROS;
- Odometry information: the navigation stack requires that odometry information be published using tf and the $nav\_msgs/Odometry$ message;

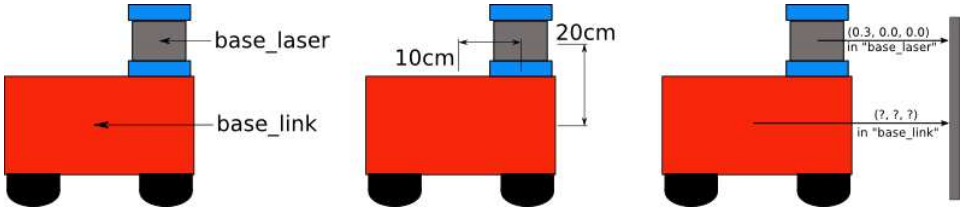

Figure A.2: Example of a tf transform (wiki.ros.org)

- Base controller: The navigation stack assumes that it can send velocity commands using a $geometry\_msgs/Twistmessage$ assumed to be in the base coordinate frame of the robot on the 'cmd_vel 'topic. This means there must be a node subscribing to the 'cmd_vel'topic that is capable of taking $(vx, vy, vtheta)$ <==> $(cmd\_vel.linear.x$, $cmd\_vel.linear.y, cmd\_vel.angular.z)$ velocities and converting them into motor commands to send to a mobile base.
- Mapping;
- Localization;

Now some of previous topic are described more in details.

A.1.1 Transform Configuration

Many ROS packages require the transform tree of a robot to be published using the tf software library. At an abstract level, a transform tree defines offsets in terms of both translation and rotation between different coordinate frames.

Let's assume that we have some data from the laser in the form of distances from the laser's center point. In other words, we have some data in the 'base_laser 'coordinate frame. Now suppose we want to take this data and use it to help the mobile base avoid obstacles in the world. To do this successfully, we need a way of transforming the laser scan we've received from the 'base_laser 'frame to the 'base_link 'frame. In essence, we need to define a relationship between the 'base_laser 'and 'base_link 'coordinate frames, as shown in Figure A.2.

To define and store this relationship using tf, we need to add them to a transform tree. Conceptually, each node in the transform tree corresponds to a coordinate frame and each edge corresponds to the transform that needs to be applied to move from the current node to its child. Tf uses a tree structure to guarantee that there is only a single traversal that links any two coordinate frames together, and assumes that all edges in the tree are

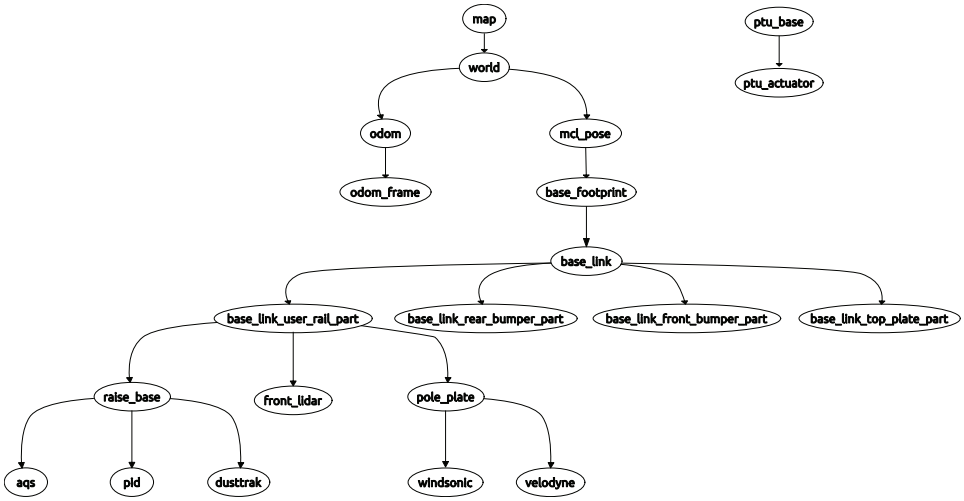

Figure A.3: Tf tree of the Husky robot used in our experiments

directed from parent to child nodes.

With this transform tree set up, converting the laser scan received in the 'base_laser'frame to the 'base_link'frame is as simple as making a call to the tf library. Our robot can use this information to reason about laser scans in the 'base_link'frame and safely plan around obstacles in its environment. In Figure A.3 is reported, for the interest of the reader, the full tf tree of the platform used.

A.1.2 Odometry Information

The navigation stack uses tf to determine the robot's location in the world and relate sensor data to a static map. However, tf does not provide any information about the velocity of the robot. Because of this, the navigation stack requires that any odometry source publish both a transform and a $nav\_msgs/Odometry$ message over ROS that contains velocity information. The $nav\_msgs/Odometry$ message stores an estimate of the position and velocity of a robot in free space: The pose in this message corresponds to the estimated position of the robot in the odometric frame along with an optional covariance for the certainty of that pose estimate. The twist in this message corresponds to the robot's velocity in the child frame, normally the coordinate frame of the mobile base, along with an optional covariance for the certainty of that velocity estimate.

In A.2 is shown the code used to get the exact position of the robot, with its orientation, in the *map* frame.

Listing A.1: An example of nav_msgs/Odometry

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Listing A.2: Code used to get from tf the exact pose of the robot

```
geometry_msgs::PoseStamped getCurrentPose()
{
    ros::Time _now_stamp_ = ros::Time(0);

    tf::StampedTransform start_pose_in_tf;
    tf::TransformListener _tf_listener;

    _tf_listener.waitForTransform("map", "base\_link", _now_stamp_,
        ros::Duration(2.0));
    try
    {
        _tf_listener.lookupTransform("map", "base\_link", _now_stamp_,
            start_pose_in_tf);
    }
    catch(tf::TransformException& ex)
    {
        ROS_INFO("TRANSFORMS␣ARE␣COCKED-UP␣PAL!␣Why␣is␣that␣:=>␣%s",
            ex.what());
    }

    tf::Vector3 start_position = start_pose_in_tf.getOrigin();
    tf::Quaternion start_orientation = start_pose_in_tf.getRotation();

    geometry_msgs::PoseStamped start_pose;
    start_pose.header.stamp = start_pose_in_tf.stamp_;
    start_pose.header.frame_id = start_pose_in_tf.frame_id_;

    tf::pointTFToMsg(start_position, start_pose.pose.position);
    tf::quaternionTFToMsg(start_orientation, start_pose.pose.
        orientation);

    return start_pose;
}
```

Listing A.3: An example of YAML file

```
image: testmap.png
resolution: 0.1
origin: [0.0, 0.0, 0.0]
occupied_thresh: 0.65
free_thresh: 0.196
negate: 0
```

A.1.3 Mapping

The map on the environment in which the robot is located is provided by a package named 'map_server '. Usually maps are stored as a couple of file: the YAML file describes the map meta-data, and names the image file. The image file encodes the occupancy data.

The image describes the occupancy state of each cell of the world in the color of the corresponding pixel. Whiter pixels are free, blacker pixels are occupied, and pixels in between are unknown. Thresholds in the YAML file are used to divide the three categories; thresholding is done inside the 'map_server '. When communicated via ROS messages, occupancy is represented as an integer in the range [0,100], with 0 meaning completely free and 100 meaning completely occupied, and the special value -1 for completely unknown. The listing A.3 presents an example of a YAML file. The fields required are the following: Required fields:

- image : Path to the image file containing the occupancy data; can be absolute, or relative to the location of the YAML file
- resolution : Resolution of the map, meters / pixel
- origin : The 2-D pose of the lower-left pixel in the map, as (x, y, yaw), with yaw as counterclockwise rotation (yaw=0 means no rotation). Many parts of the system currently ignore yaw.
- occupied_thresh : Pixels with occupancy probability greater than this threshold are considered completely occupied.
- free_thresh : Pixels with occupancy probability less than this threshold are considered completely free.
- negate : Whether the white/black free/occupied semantics should be reversed (interpretation of thresholds is unaffected)

A.1.4 Localization

Robust robot localization is critical in mobile robotics to associate measurements with spatial locations. Bieber and Straßer in [10] proposed a different environmente representation, called Normal Distribution Transform (NDT), based on a sparse Gaussian mixture model where robot localization can be achieved using e.g. Monte Carlo Localization (MCL) [12]. In NDT-MCL [24], the standard MCL probabilistic framework uses NDT as a representation for both, the map and the observations to relax hard discretization assumptions imposed by e.g. grid-map models. This, combined with a piece-wise continuous NDT environment representation, allows to achieve the accuracy and repeatability needed for industrial applications in large environments.

A.2 Other Ros Packages

In Figure A.4 is reported to *nodegraph* of out system. The active nodes are represented in the circular shapes, while the topics to which they publish or subscribe are in small rectangles. Nodes are contained in packages.

In the previous section we have already discussed, directly or not, about the following nodes: NDT-MCL (for localization), move_base (for sending motor commands to the robot), lidar (for providing sensor information) and odo_static (for odometry). Despite of the *husky* package, containing all the description files related to the platform and dipendent on it, we have to present the nodes fundamentals for achieving the gas detection task.

The first one is the *rmld_node* that jas the goal of record in a log all the values read by the TDLAS, but the most important one for our experience is, for sure, the one named *ptu_control*. As described in Chapter 4, with the *ray casting* we are able to estimate the minimum and maximum angle required to scan an unknown area. Passing these values to this node, we can move the pan-tilt unit which the TDLAS sensor is mounted on only for the desided angular sector, saving precious time. In A.4 is reported the code written to call the service and to send the message containing the required parameters.

Listing A.4: Code used for sending messages to the pan-tilt unit

```
1
2  void gasDetection(){
3
4          ros::NodeHandle n;
5          ros::ServiceClient client1 = n.serviceClient<ptu_control::
               commandSweep>("/ptu_control/sweep");
6          ros::ServiceClient client2 = n.serviceClient<amtec::GetStatus
               >("/amtec/get_status");
7
8          ptu_control::commandSweep srvSweep;
9
10         if(min_pan_angle > max_pan_angle){
11             double tmp = min_pan_angle;
12             min_pan_angle = max_pan_angle;
13             max_pan_angle = tmp;
14         }
15
16         srvSweep.request.min_pan        = min_pan_angle;
17         srvSweep.request.max_pan        = max_pan_angle;
18         srvSweep.request.min_tilt       = tilt_angle;
19         srvSweep.request.max_tilt       = tilt_angle;
20         srvSweep.request.n_pan          = num_pan_sweeps;
21         srvSweep.request.n_tilt         = num_tilt_sweeps;
22         srvSweep.request.samp_delay     = sample_delay;
23
24         if (client1.call(srvSweep)){
25             ROS_INFO("Gas␣detection␣in␣progress␣...␣<%.2f~%.2f,%.2f>",
                   min_pan_angle,max_pan_angle,tilt_angle);
26         }else{
27             ROS_ERROR("Failed␣to␣initialize␣gas␣scanning.");
28         }
29
30
31 }
```

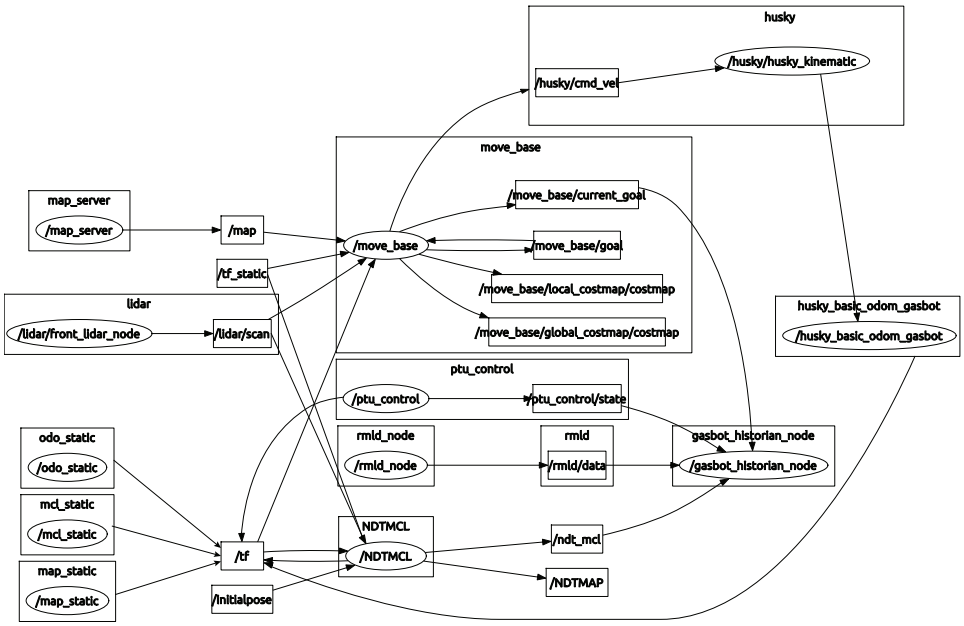

Figure A.4: Nodegraph representing all the active nodes

Appendix B

Practical Issues

We had to face many issues while implementing our algorithm, both during the design and coding part and during experiments in the real world.

B.1 Map representation

In order to represent the environment, we generate a matrix composed by 0 (free cells) and 1 (obstacle cells) from a map image file. To obtain cells of a chosen size, for example $1m^2$, the cells obtained while reading the map might have to be clustered together, depending on the resolution of the image file.

This often generated *unreachable cells*, cells which could be seen but become unreachable for the robot because of obstacle expansion while clustering.

Because of the required precision when using maps, we decided to let the robot move in cells of $1m^2$ in the real life experiments and in most of simulation experiments, in order to take into account the space required by the robot to move without hitting walls and obstacles, also due to possible non-perfect localization.

Since we did not want to lose precision in the gas detection, we decided to introduce a second grid representing the map, used for sensing operation, with the possibility to choose the resolution of both grids independently. This however might introduce more clustering, depending on the resolution of the grids and of the original map file.

Maps in some cases had to be manually fixed, by removing free cells which where unreachable because it would be physically impossible for the robot to reach them, or by manually expanding some obstacles, such as

stairs, which were not generated in the correct size in the Teknikhuset's corridor map because the lidar used to generate the 2d map is set close to the ground, or walls in case of windows, which are not sensed by the lidar scan.

Moreover, maps used in our algorithm and maps used in ROS were read in a different way, resulting mirrored and shifted, requiring some static transforms in order to run the algorithm successfully.

B.2 Ray casting

We used ray casting for a significative number of tasks in our implementation: finding candidate positions, performing sensing operations, calculating the criteria of information gain and sensing time.

Two main implementation were used. In both of them, the angles and the FoV used are considered in a cartesian system fixed to the environment, and not to the robot.

In our first implementation, given the robot pose, a huge number of rays were cast depending on the range and FoV of the robot, for example with a range of 20 meters and a FoV of 180 degrees 1024 rays had to be casted in order to sense all the visible cells in a sensing operation. In this implementation, given the initial and final angle ϕ_{start} and ϕ_{end} defining the FoV of the robot, a fixed number n of ray was shot, with slopes going from ϕ_{start} to ϕ_{end} with an increase of $\frac{\phi_{end}-\phi_{start}}{n}$.

In order to obtain a more precise solution, which didn't require to choose a different number of rays depending on robot's parameters, we made a second implementation, in which a single ray is shot for each cell to check if it is visible from the current robot pose. Details on this implementation are reported in Chapter 4.

B.3 Issues during experiments

We faced many issues while doing experiments in the real world. Apart from problems due to stairs not recognized in the map and unreachable cells that compromised some of our runs, some issues prevented us from being able to run tests on the same map as Arain [6], due to the NDT map used for localization not overlapping with the occupancy grid used to send the next pose to the robot, generating an imprecise localization.

Moreover, an issue in the ptu unit of the robot made the sensing operation randomly perform two sweeps instead of one, slowing down the explo-

ration by a significative amount of time.