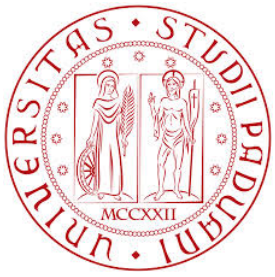


Università degli studi di Padova
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria dell'Automazione

Strategies for coverage and focus on event for robotic swarms with limited sensing capabilities

Relatore
Prof. Angelo Cenedese

Laureando
Marco Fabris
matr. 1096648

Anno Accademico 2015/2016

To my parents for their infinite patience.
To Alberto and Matteo for their precious help.

Contents

List of Figures

List of Tables

Abstract

We consider the problem of coverage in Robotic Networks: developing an efficient algorithm which is able to perform a deployment in static-obstacle-structured environments is our main idea. We are interested on the trade-off between local communication and optimal coverage, therefore we are going to present an algorithm based on article: "Sensor Coverage Robot Swarms Using Local Sensing without Metric Information" (2015), using its same scenario types. The particular aim and tasks we chose are 1. the maximization of local communication: we reformulated the algorithm in order to remove the assumption of a centralized server which collects local information from individual robots and performing the distributed approach as far as possible; 2. the definition of an expansion policy when we have multiple source of robot swarms; 3. the insertion of the possibility to detect small non-convex features in the environment which usually result "blind-spots" for some obstacles, assuming a space structure model for robots (e.g. giving them a round shape) in order to define the limitation of smallest non-convex coverable features; 4. the possibility of event detection and focusing by sending a group of robots.

Part I

Introduction

1

Preliminaries

Sensor coverage of indoor environments using teams of mobile robots is a well-studied problem in robotics. In this framework, we will focus on unknown complex indoor environments that are typically hard to cover because there are no a priori information about the space structure and there is not an optimal strategy to reach the complete coverage.
We would like to perform the task using the lowest number of agents and cheap mobile robots. In particular the aim of the thesis is to develop an efficient algorithm to deploy mobile a sensor network, which consists of a collection of sensing devices that can coordinate their actions through wireless communication and aim at performing tasks such as event detection, reconnaissance, surveillance, target tracking, monitoring over a specific region or exploration of environments that are hazardous to human operators.
The advantages of this type of applications are the possibility of visiting and monitoring every point of the given space in a automatic and distributed way, the robustness to failures of single robots, the ability of the agents to adapt their disposition and dynamics considering the obstacles in the environment, the reliability of a connected sensor network and the maximization of the sensing information utilizing the lowest amount of available local resources.

1.1 Problem description

We consider the problem of efficiently exploring an **unknown** indoor environment with a rapidly expanding **swarm of robots** with **limited and noisy local sensing** with no global localization or sensing capabilities (see Fig. 1.1). In particular, the only sensory capabilities that we assume on each robot are those of an omni-directional camera with a limited radial range of vision and a touch sensor to detect contact/collision with obstacles and other robots. We call the disk around a robot, representing a sensing radius of the omni-directional camera, the robot's disk of visibility, within which the bearing to the neighbouring robots and their identities can be detected. However, the camera does not provide a range measurement due to projection of the 3D world on to the camera plane. Thus, obstacles are detected through touch sensors near the base of the robots, and cannot be detected using the camera. The robots can also communicate with each other and can be driven in arbitrary direction.

The particular aim and tasks we chose are

1. the maximization of local communication: we reformulated the algorithm in [2] in order to remove the assumption of a centralized server that collects local infor-

mation from individual robots and performing the distributed approach as far as possible;

2. the definition of an expansion policy when we have multiple source of robot swarms;

3. the insertion of the possibility to detect small non-convex features in the environment which usually result "blind-spots" for some obstacles, assuming a space structure model for robots (e.g. giving them a round shape) in order to define the limitation of smallest non-convex coverable features;

4. the possibility of event detection and monitoring by sending a group of robots, as shown in the example in Fig. 1.2.

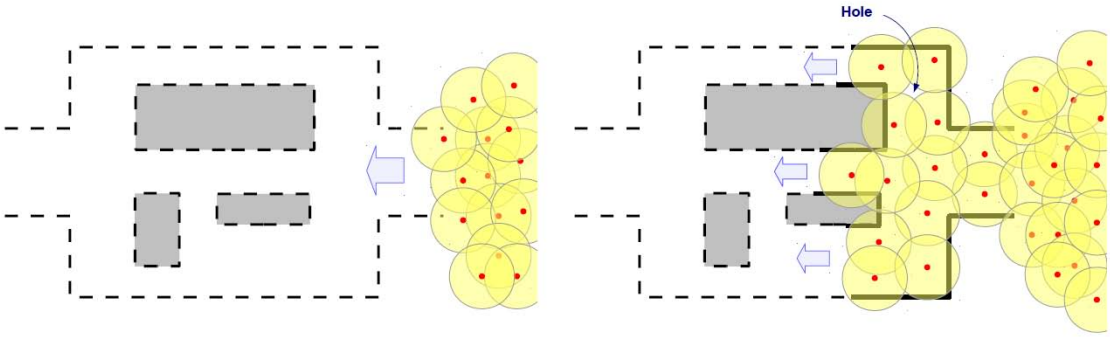


Figure 1.1.: Illustration of a swarm of robots entering an environment and attaining coverage. The hole shown on the right figure is something we would like to avoid.

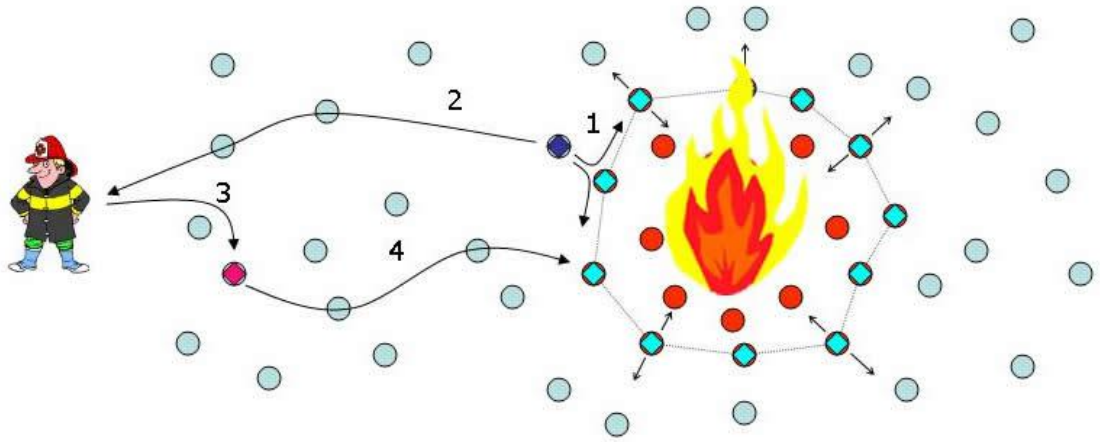


Figure 1.2.: Illustration of monitoring during a blaze (a possible example of event): the closest robots start to surround the fire in order to delimit its perimeter. Then, a fireman, helped by a previously attained coverage, can choose the **quickest** and **safest** way to extinguish the flames.

2

General framework

In this chapter, we continue the introduction providing an overview that touches upon different topics concerning engineering control systems, the current state of art for the problem we have already described, our new contributions and, finally, the outline of the reminder of the thesis.

2.1 Overview

In the recent decades researchers focused their attentions in engineering systems composed by a large number of devices that can communicate and cooperate to achieve a common goal. Although complex large-scale monitoring and control systems are not new, as for example air traffic control or smart grids applications, a new architectural model is emerging, mainly thanks to the adoption of smart agents i. e. devices that are capable of cooperating and of taking autonomous decisions without any supervisory system. In fact, traditional large-scale systems have a centralized or at best a hierarchical architecture, as in [3], which has the advantage to be relatively easy to be designed and has safety guarantees. However, these systems require reliable sensors and actuators and in generally are very expensive. Another relevant limitation related with centralized systems is that they do not scale well, due to communication and computation limitations. The recent trend, in order to avoid these problems, is to substitute costly sensors, actuators and communication systems with a larger number of devices that can autonomously compensate potential failures and computation limitations through communication and cooperation. Moreover, it is preferable to use a decentralized approach in order to offer more robustness and efficiency in terms of energy cunsumption.

2.1.1 Coverage Control

Deploying multiple agents to perform tasks is advantageous compared to the single agent case: it provides robustness to agent failure and allows to handle more complex tasks. The single, heavily equipped vehicle may require considerable power to operate its sensor payload, it lacks robustness to vehicle failure and it cannot adapt its configuration to environmental changes. A cooperative network of sensors and vehicles equipped with sensor, has the potential to perform efficiently and reliably tasks in a more flexible and scalable way than single better-equipped agents. Therefore, distributed control can be employed by groups of robots to carry out tasks such as environmental monitoring, automatic surveillance of rooms, buildings or towns, search and rescue as shown in the applications presented by [4].
The performance of multi-robot and sensor network in distributed area exploration is

sensitive to the location of agents in the mission space. In particular, sensors must be deployed so as to maximize the information extracted from the mission space. The goal is therefore to drive the sensors/ agents to the position such that a given region is optimally covered by sensors. This causes the network to spread out over the environment while aggregating in areas of high sensory interest, see Fig. 2.1. Furthermore, robots do not know beforehand where areas of major interest are located: the network is required to learn this information online from sensors measurements.
The problem of optimizing sensor locations in fixed sensor networks has been extensively studied in the past, and are still open. In such problems, the solution is a Voronoi partition, shown in [5], where the optimal sensor domain is a Voronoi cell in the partition and the optimal sensor location is a centroid of a Voronoi cell in the partition.
In this work, we consider a mobile sensing network composed of vehicles which are dynamic agents, equipped with sensors to sample the environment; the problem of deploying agents is referred to as the Coverage control problem. Moreover, the robots we use are not provided by a localization systems, indeed they exploit only bearing angles measurements.

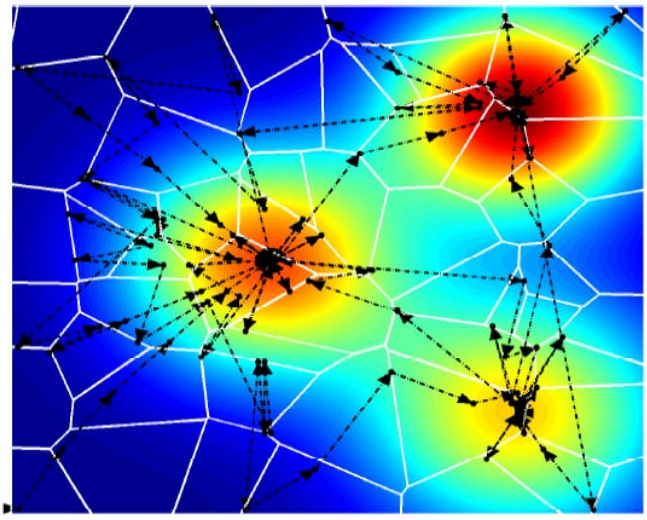


Figure 2.1.: Coverage Control: the network spread out over the environment, while aggregating in areas of high sensory interest.

2.1.2 Event detection

According to [6], event detection in wireless sensor networks is a sophisticated method for processing sampled data directly on the sensor nodes, thereby reducing the need for multihop communication with the base station of the network. In contrast to application-agnostic compression or aggregation techniques, event detection pushes application-level knowledge into the network. In-network event detection – especially the distributed form involving multiple sensor nodes – has thus an exceptional potential to increase energy efficiency, thus prolonging the lifetime of the network.
The most basic approach is local event detection. In this approach, each node gathers data from its local sensors and employs local algorithms to decide whether a specific event has occurred. Data of neighbouring nodes is not taken into account, and all signal

processing is performed on the local node itself. If an event is detected, each node or a cluster of nodes signals the results directly to the base station of the sensor network as shown in Fig. 2.2.

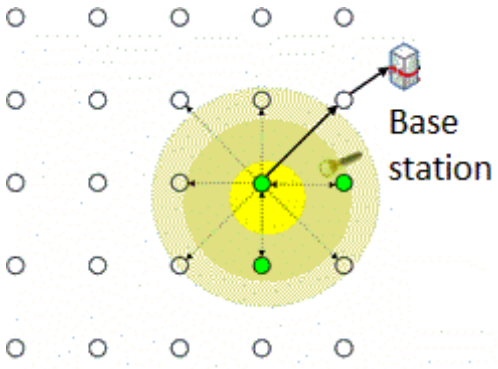


Figure 2.2.: The three green nodes form a cluster that signals the presence of a detected event (yellow area); furthermore, the cluster communicates with some of its neighbouring nodes to exchange information.

2.1.3 Dispatch of robotic agents

Given a set of events and a set of robots, the dispatch problem is to allocate one robot for each event to visit it. In a single round, each robot may be allowed to visit only one event (matching dispatch), or several events in a sequence (sequence dispatch problem). The distributed version of the problem is embedded into a sensor network field, where each event is discovered by a sensor and reported to a robot. Existing centralized and distributed solution based on stable marriage problem suffers from the presence of long robot paths. The other existing solution, based on k-means clustering and traveling salesman tour, suffers from the design goal of allocating the same number of events to each robot. Anyway, one of the most use approach is shown in Fig. 2.3: a previous coverage of the are is followed by the dispatch operations, i.e. sending robots.

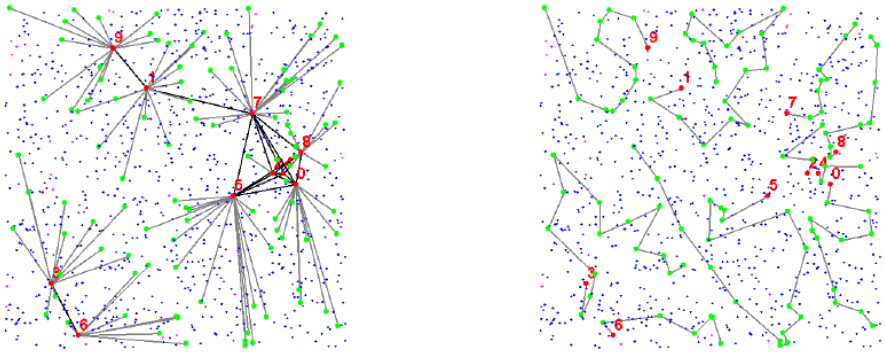


Figure 2.3.: Approached used in [1] is very common nowadays: before a network of agents is deployed (left picture); then, dispatch algorithms start to run (right picture) in order to design trajectories for sending robots.

In particular, as shown in the approach of [7], robot models follow dynamics of physical vehicles (e.g. front-wheeled cars) that need a path planning algorithm to be sent.

2.2 State of art

Old coverage applications are based on centralized localization for each robot (e.g. GPS) in order to accomplish Voronoi partition of the environment and the minimization of a coverage functional as shown in [5]. However, such approaches need a priori information about the environment where the robots would have deployed. Lately complete sensor coverage of indoor environments using swarms of robots have been studied modelling the known sensor network as a graph, robots are assumed to have global localization and can navigate independently from one location to another in a global coordinate frame. In all of these lines of research, global localization of the robots, a priori knowledge of the environment (obstacle configuration), availability of metric information and ability to control the robots from one point in the environment to another have been assumed. This approach and its results can be found in [8].
First distributed versions of coverage algorithms for robot swarms come with very limited theoretical guarantee because coverage purely based on graph theory does not ensure sensor coverage nor optimality; furthermore, these kind of approaches inherently assume availability of some metric information.
In recent years coverage by sensor network is studied more formally using simplicial complex theory described in [9], [10] and [11] and homological tools from algebraic topology. Such approaches are completely topological and require little to no metric information, as solution proposed in [12]. In general, homology computation is extremely useful in detecting holes in sensor network coverage.
Closely related to our work is the literature of exploration of unknown environment using simplicial complex without global localization. Those similar studies about the maximization of coverage area with a limited number of robots equipped with restricted local sensors are discussed in [13] where the authors assume some metric information and the coverage task boils down to a Maximum Area Triangulation Problem (MATP), that is the maximal sets of non-overlapping triangles with vertices in the given points whose union is the convex hull of the point set.
The latest research from which our work draws inspiration is [2]: this article offers new prospectives and improves the previous studies using a different approach based on three aspects. Firstly their method is robust to robot failure and can start from an arbitrary configuration of the robots, since the computations at every iteration are purely based on the current state only. Secondly their approach of pushing the robots through the graph in order to expand the frontier to the unknown regions, instead of navigate a robot from the source to the frontier, has a lower execution time at each cycle, and does not require any restriction on the minimum distance between two neighbouring robots. Lastly, their proposed algorithm does not require the workspace to be simply-connected as they demonstrate through experimental results in complex indoor environment.
The authors use simplicial complex representation (Vietoris-Rips complex, see [14]) to solve the problem of deploying robots in an unknown environment without global localization. Since their approach is fundamentally topological, the proposed method is highly robust to sensor noise. Assuming sufficient numbers of available robots they attain complete sensor coverage of the entire finite environment.

2.3 Contributions

Starting from the good aspects described in [2] the main focus of the thesis is on the following topics.

- A completely **distributed approach** that requires local communication only: each robot needs to share with its neighbours just the information about measured bearing angles and neighbourhoods. In particular when we search useless robots instead of computing cycles in the whole graph we just compute angles between neighbours. This makes more efficient and completely distributed the computation of redundant robots which implies a faster distributed execution.
- A **realistic physical model** for the robots that have been assumed as circular homogeneous shapes. This fact leads us to take into account collisions between robots and between a robot and environment; hence we developed a model for robot-environment interactions.
- Since robots have been provided by a model, we even adopt an **overlapping avoidance protocol** between each agent which permits to well attain deployments without position overlapping.
- We define an expansion protocol when robots invading the environment are generated from **different sources**. In particular we analyse the difference between the situation in which the sources can or cannot share information (*i.e.* there is or there is not a graph connection between them) and the opposite circumstance.
- A robot that is not involved in the coverage expansion, since it is located far from the extending frontier, does not need to be taken into account during the deployment process of new agents. Thanks to this observation, we introduced a **forgetting factor methodology**. For this kind of robots, sharing information is not necessary: this allows to save energy in a real scenario and make more efficient our algorithm.
- When coverage is finally attained, we guarantee the possibility of **focusing on an event**. This consists in: detect the presence of the event, localize the robots around it and, in conclusion, perform a dispatch, i.e. to send agents close to the event with a feedback control law.

2.4 Outline of the work

The remainder of the thesis is organized as follows.

- Chapter 3 presents the Graph Theory explained in [15] we need to use for describing connections between robots: graphs are the basis for representing robotic networks because they allow to schematize agents as nodes and links as edges;
- Chapter 4 shows two important topological tools to perform Coverage: the classical Voronoi's tessellations attained using the well-known Lloyd's algorithm (see [5]) and the latest approach of Vietoris-Rips Complex Theory (see [14] and [16]);

- Chapter 5 deals with theoretical coverage bounds we found for rectangular scenarios, base stations, providing some useful estimations for understanding the best ways to deploy robots in this framework;
- Chapter 6 copes with some concepts taken from [17] about the controllability for nonlinear systems we need to examine in order to model agents as front-wheeled cars, in particular the Manifold Theory and the Lie Brackets will be exploited;
- in Chapter 7 all models employed in this work are presented: scenarios, robots, robotic networks, environment interactions, the front-wheeled car model and the events;
- Chapter 8 describes how the coverage and event detection algorithm works and shows several ad hoc simulations in order to explain to the reader what are the basic goals we reached;
- Chapter 9 shows different kind of simulations in complex structure scenarios;
- Chapter 10 illustrates simulations showing the part regarding event detection and dispatch;
- Chapter 11 concludes the thesis and describes directions for future works.

Part II

Theoretical results

3

Graph theory

A graph is a way of representing relationships that exist between pairs of objects (see [15]). That is, a graph is a set of objects, called vertices, together with a collection of pairwise connections between them. Graphs have applications in a host of different domains, including mapping, transportation, electrical engineering and robotic networks. The latter application will be the main topic studied in the next chapter.

3.1 Definitions

Viewed abstractly, a **graph** $\mathcal{G}$ is simply a set $\mathcal{V}$ of **vertices** and a collection $\mathcal{E}$ of pairs of vertices from $\mathcal{V}$, called **edges**. Thus, a graph is a way of representing connections or relationships between pairs of objects from some set $\mathcal{V}$. Incidentally, some books use different terminology for graphs and refer to what we call vertices as nodes and what we call edges as arcs. We use the terms "vertices" and "edges".
The set $\mathcal{N}_u = \{v_i | (u, v_i) \in \mathcal{E}, i = 1, 2, 3, ...\}$ is called **neighborhood** of u. This gathers all vertices which are connected to u by a single edge.
A **weighted graph** is a graph that has a numeric (for example, integer) label $w(e)$ associated with each edge e, called the **weight** of the edge e.
Edges in a graph are either directed or undirected. An edge (u, v) is said to be **directed** from u to v if the pair (u, v) is ordered, with u preceding v. An edge (u, v) is said to be **undirected** if the pair (u, v) is not ordered. Undirected edges are sometimes denoted with set notation, as u, v but for simplicity we use the pair notation (u, v), noting that in the undirected case (u, v) is the same as (u, v).
If all edges in a graph are undirected, then we say the graph is an **undirected graph**. Likewise, a **directed graph**, also called **digraph**, is a graph whose edges are all directed. A graph that has both directed and undirected edges is often called a **mixed graph**. Note that an undirected or mixed graph can be converted into a directed graph by replacing every undirected edge (u, v) by the pair of directed edges (e, v) and (v, u). It is often useful, however, to keep undirected and mixed graphs represented as they are, for such graphs have several applications.
The two vertices joined by an edge are called the **end vertices** (or **endpoints**) of the edge. If an edge is directed, its first endpoint is its **origin** and the other is the **destination** of the edge. Two vertices u and v are said to be **adjacent** if there is an edge whose end vertices are u and v. An edge is said to be **incident** on a vertex if the vertex is one of the edge's endpoints. The **outgoing edges** of a vertex are the directed edges whose origin is that vertex. The **incoming edges** of a vertex are the directed edges whose destination is that vertex. The **degree** of a vertex v, denoted $deg(v)$, is the number of incident edges of v. The **in-degree** and **out-degree** of a vertex v are

the number of the incoming and outgoing edges of v, and are denoted $indeg(v)$ and $outdeg(v)$, respectively.
The definition of a graph refers to the group of edges as a **collection**, not a **set**, thus allowing for two undirected edges to have the same end vertices, and for two directed edges to have the same origin and the same destination. Such edges are called **parallel edges** or **multiple edges**. Parallel edges can be in a flight network, in which case multiple edges between the same pair of vertices could indicate different flights operating on the same route at different times of the day. Another special type of edge is one that connects a vertex to itself. Namely, we say that an edge (undirected or directed) is a **self-loop** if its two endpoints coincide. A self-loop may occur in a graph associated with a city map, where it would correspond to a "circle" (a curving street that returns to its starting point).
A **path** is a sequence of alternating vertices and edges that starts at a vertex and ends at a vertex such that each edge is incident to its predecessor and successor vertex. A **cycle** is a path with at least one edge that has its start and end vertices the same. We say that a path is **simple** if each vertex in the path is distinct, and we say that a cycle is **simple** if each vertex in the cycle is distinct, except for the first and last one. A **directed path** is a path such that all edges are directed and are traversed along their direction. A **directed cycle** is similarly defined.
A graph is said **connected** if for each $(u, v) \in \mathcal{E}$ there exists a path which connects u to v. A **forest** is a graph without cycles. A **tree** is a connected forest. A **spanning tree** is a tree which covers all the vertices of a graph.

3.2 Shortest path problem

Let $\mathcal{G}$ be a weighted graph. The length (or weight) of a path P is the sum of the weights of the edges of P. That is, if $P = ((v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k))$, then the length of P, denoted $w(P)$, is defined as

$$w(P) = \sum_{i=0}^{k-1} w((v_i, v_{i+1})) \tag{3.1}$$

The **distance** from a vertex v to a vertex u in $\mathcal{G}$, denoted $d(u, v)$, is the length of a minimum length path (also called shortest path) from v to u, if such a path exists.
People often use the convention that $d(v, u) = +\infty$ if there is no path at all from v to u in $\mathcal{G}$. Even if there is a path from v to u in $\mathcal{G}$, the distance from v to u may not be defined, however, if there is a cycle in $\mathcal{G}$ whose total weight is negative.
There is an interesting approach for solving this **single-source** problem based on the **greedy method** design pattern. Recall that in this pattern we solve the problem at hand by repeatedly selecting the best choice from among those available in each iteration. This paradigm can often be used in situations where we are trying to optimize some cost function over a collection of objects. We can add objects to our collection, one at a time, always picking the next one that optimizes the function from among those yet to be chosen.

3.2.1 Dijkstra's algorithm

The main idea in applying the greedy method pattern to the single-source shortest-path problem is to perform a weighted breadth-first search starting at v. In particular, we can use the greedy method to develop an algorithm that iteratively grows a "cloud" of vertices out of v, with the vertices entering the cloud in order of their distances from v. Thus, in each iteration, the next vertex chosen is the vertex outside the cloud that is closest to v. The algorithm terminates when no more vertices are outside the cloud, at which point we have a shortest path from v to every other vertex of $\mathcal{G}$. This approach is a simple, but nevertheless powerful, example of the greedy method design pattern.
Applying the greedy method to the single-source, shortest-path problem, results in an algorithm known as **Dijkstra's algorithm**. When applied to other graph problems, however, the greedy method may not necessarily find the best solution. Nevertheless, there are a number of situations in which the greedy method allows us to compute the best solution. In the next section, we discuss one of this situations: computing shortest paths.
In order to simplify the description of Dijkstra's algorithm, we assume, in the following, that the input graph $\mathcal{G}$ is undirected (that is, all its edges are undirected) and simple (that is, it has no self-loops and no parallel edges). Hence, we denote the edges of $\mathcal{G}$ as unordered vertex pairs (u, z).
In Dijkstra's algorithm for finding shortest paths, the cost function we are trying to optimize in our application of the greedy method is also the function that we are trying to compute the shortest path distance, see (3.1). This may at first seem like circular reasoning until we realize that we can actually implement this approach by using a bootstrapping trick, consisting of using an approximation to the distance function we are trying to compute, which in the end will be equal to the true distance.

3.2.2 Edge relaxation strategy

Let us define a label $D[u]$ for each vertex u in V, which we use to approximate the distance in $\mathcal{G}$ from v to u. The meaning of these labels is that $D[u]$ will always store the length of the best path we have found so far from v to u. Initially, $D[v] = 0$ and $D[u] = +\infty$ for each $u \neq v$, and we define the set C, which is our **cloud** of vertices, to initially be the empty set t. At each iteration of the algorithm, we select a vertex u not in C with smallest $D[u]$ label, and we pull u into C. In the very first iteration we will, of course, pull v into C. Once a new vertex u is pulled into C, we then update the label $D[z]$ of each vertex z that is adjacent to u and is outside of C, to reflect the fact that there may be a new and better way to get to z via u. This update operation is known as a **relaxation** procedure, for it takes an old estimate and checks if it can be improved to get closer to its true value. (A metaphor for why we call this a relaxation comes from a spring that is stretched out and then relaxed back to its true resting shape.) In the case of Dijkstra's algorithm, the relaxation is performed for an edge (u, z) such that we have computed a new value of $D[u]$ and wish to see if there is a better value for $D[z]$ using the edge (u, z). The specific edge relaxation operation is as follows:

Edge relaxation:

$$\textbf{if } D[u] + w((u,z)) < D[z] \textbf{ then}$$
$$D[z] \longleftarrow D[u] + w((u,z))$$

Pseudo-code for Dijkstra's algorithm:

Algorithm: ShortestPath($\mathcal{G}, v$)
Input: A simple undirected weighted graph $\mathcal{G}$ with nonnegative edge weights, and a distinguished vertex v of $\mathcal{G}$
Output: A label $D[u]$, for each vertex u of $\mathcal{G}$, such that $D[u]$ is the length of a shortest path from v to u in $\mathcal{G}$

Initialize $D[v] \longleftarrow 0$ and $D[u] \longleftarrow +\infty$ for each vertex $u \neq v$
Let a priority queue Q contain all vertices of $\mathcal{G}$ using the D labels as keys.
while Q is not empty **do**
    {pull a new vertex in the cloud}
    $u \longleftarrow Q.removeMin()$
    for each vertex z adjacent to u such that z is in Q **do**
        {perform the relaxation procedure on edge (u,z)}
        if $D[u] + w((u,z)) < D[z]$ **then**
            $D[z] \longleftarrow D[u] + w((u,z))$
            Change to $D[z]$ the key of vertex z in Q.
return the label $D[u]$ of each vertex u

Some illustrations of Dijkstra's algorithm are shown in the following Fig. 3.1 and Fig. 3.2.

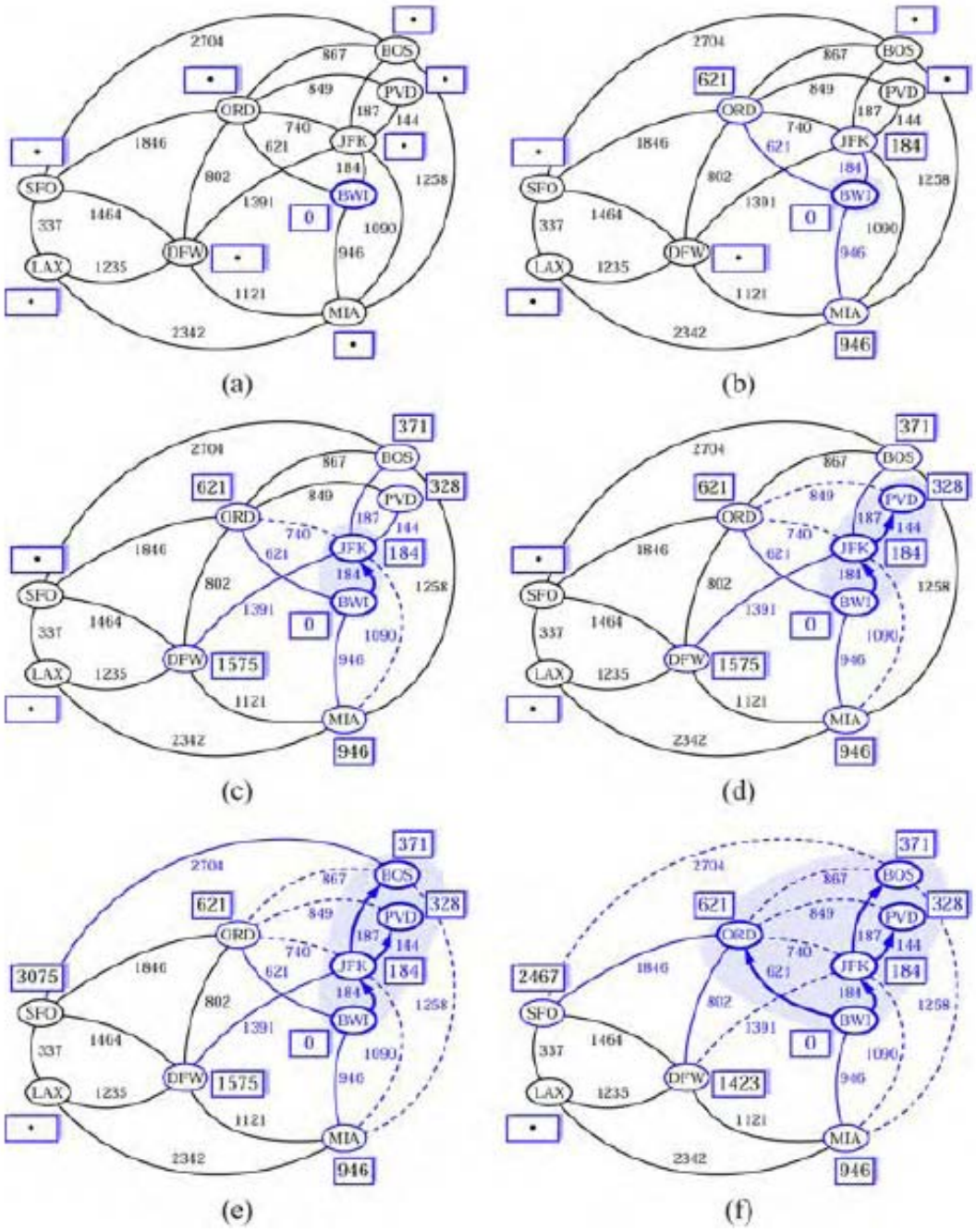


Figure 3.1.: An execution of Dijkstra's algorithm on a weighted graph. The start vertex is BWI. A box next to each vertex v stores the label $D[v]$. The symbol • is used instead of $+\infty$. The edges of the shortest-path tree are drawn as thick blue arrows, and for each vertex u outside the cloud the current best edge for pulling in u is shown with a blue solid line.

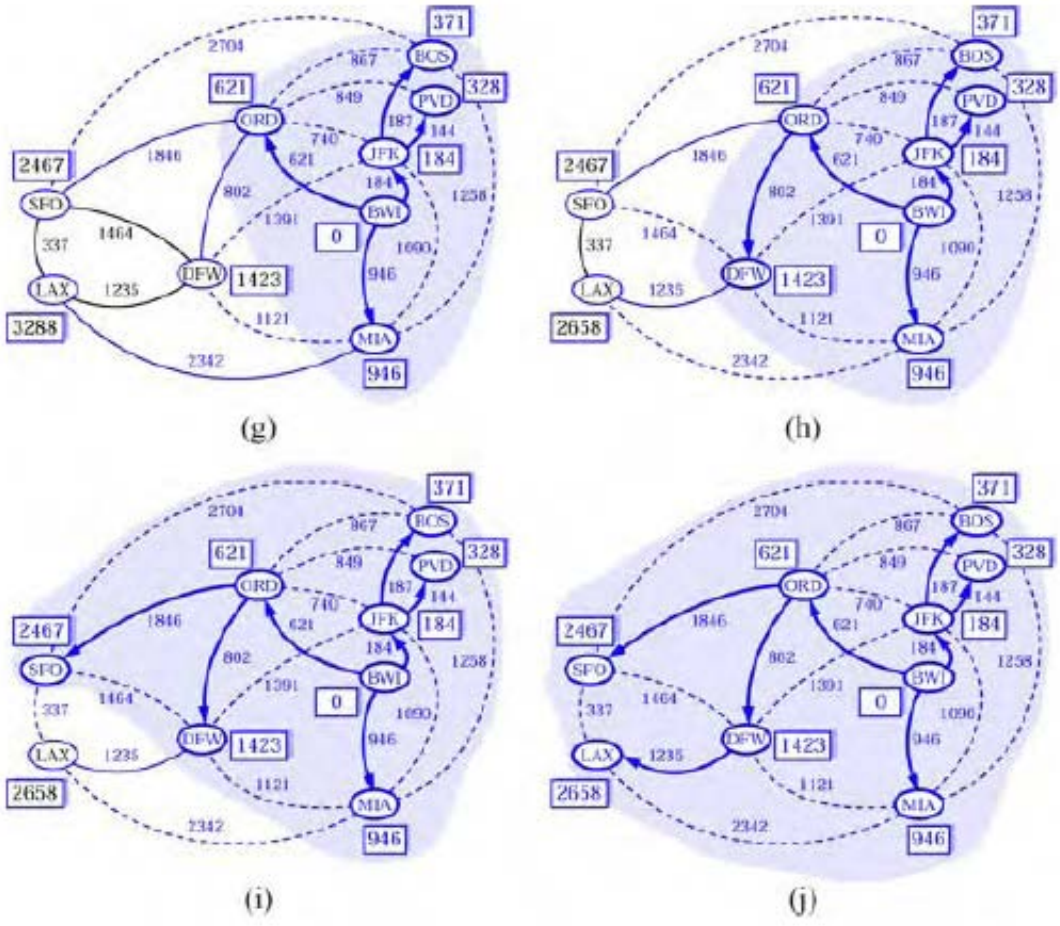


Figure 3.2.: Continued from Fig. (3.1): an example execution of Dijkstra's algorithm.

4

Coverage and robotic networks

Coverage is the task of determining a path that passes over all points of an area or volume of interest while avoiding obstacles. A Robotic Network is a system composed of robotic agents connected to a communications network such as the Internet or LAN, like in Fig. 4.1.
In many applications we have to consider the topological space in order to construct an **optimized network**, for instance when we are interested to capture an object using cameras as sensors. This is the reason why coverage techniques are so relevant for exchanging information when we are dealing, in general, with controlled interconnected dynamic systems (see [18]).
In the next sections, we are going to discuss about two well known topological tools to perform coverage: the Voronoi tessellations and the Vietoris-Rips Complex Theory. The former is the classical method used to perform coverage, while the latter is an approach applied in the recent years. Although we will use the Vietoris-Rips complex in our algorithms, we will also compute the Voronoi tessellation as a final result for comparison.

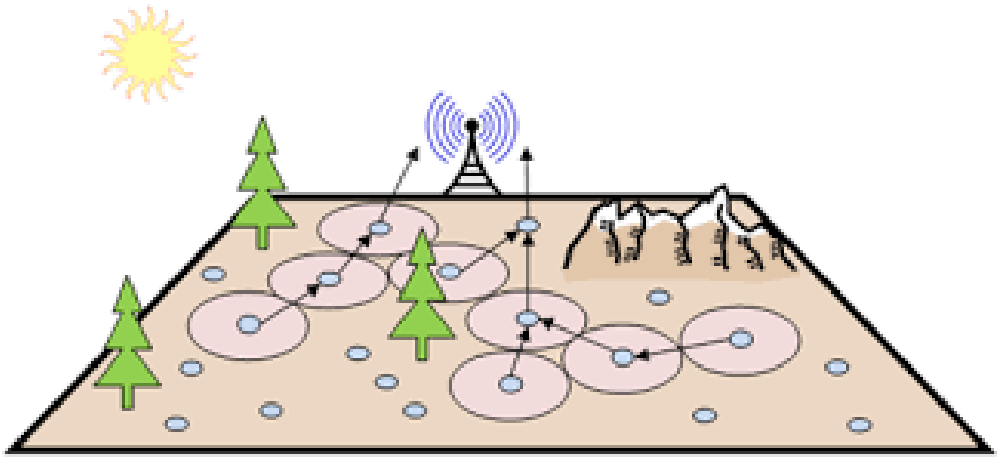


Figure 4.1.: Here, an example of an outdoor application. In order to be attained, coverage needs a sensor network. Once the latter is automatized, it is called robotic network. This means that coverage is performed automatically by the agents which deploy themselves and provide connection into the network and communicate with each other and the base station.

4.1 Voronoi tessellations

Given an open set $\Omega \subseteq \mathbb{R}^N$, the set $\{V_i\}_{i=1}^k$ is called a **tessellation** of Ω if $V_i \cap V_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^k \overline{V}_i = \overline{\Omega}$ (see [5]). Let $|\cdot|$ denote the Euclidean norm on $\mathbb{R}^N$. Given a set of points $\{\mathbf{z}_i\}_{i=1}^k$ belonging to $\overline{\Omega}$, the Voronoi region $\hat{V}_i$ corresponding to the point $\mathbf{z}_i$ is defined by

$$\hat{V}_i = \{\mathbf{x} \in \Omega \mid |\mathbf{x} - \mathbf{z}_i| < |\mathbf{x} - \mathbf{z}_j| \text{ for } j = 1, ..., k, j \neq i\}$$

The points $\{\mathbf{z}_i\}$ are called generators. The set $\left\{\hat{V}_i\right\}_{i=1}^k$ is a **Voronoi tessellation** or **Voronoi diagram** of Ω, and each $\hat{V}_i$ is referred to as the **Voronoi region** corresponding to $\mathbf{z}_i$. (Depending on the application, there exist many different names for Voronoi regions, including Dirichlet regions, area of influence polygons, Meijering cells, Thiessen polygons, and S-mosaics.) The Voronoi regions are polyhedra. These tessellations, and their dual tessellations (in $\mathbb{R}^2$, the Delaunay triangulations), are very useful in a variety of applications. Given a region $V \subseteq \mathbb{R}^N$ and a density function ρ, defined in V, the mass

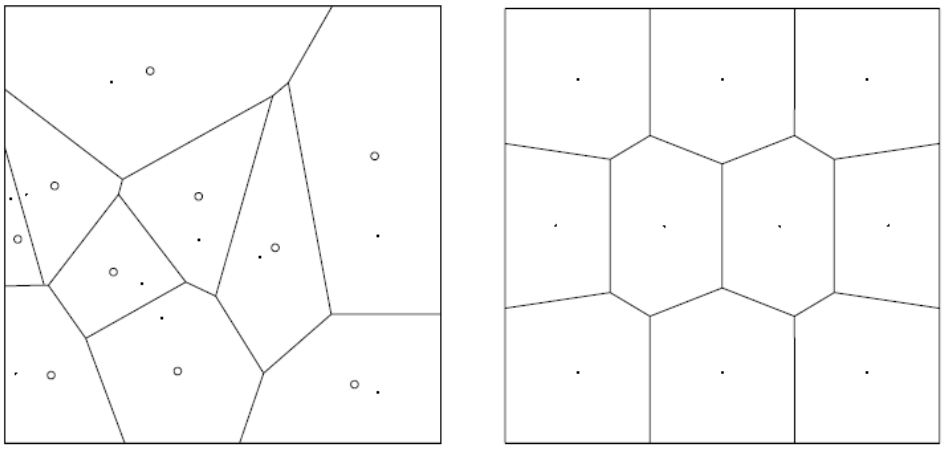


Figure 4.2.: On the left, the Voronoi regions corresponding to 10 randomly selected points in a square; the density function is a constant. The dots are the Voronoi generators and the circles are the centroids of the corresponding Voronoi regions. Note that the generators and the centroids do not coincide. On the right, a 10-point centroidal Voronoi tessellation. The dots are simultaneously the generators for the Voronoi tessellation and the centroids of the Voronoi regions.

centroid $\mathbf{z}^*$ of V is defined by

$$\mathbf{z}^* = \frac{\int_V \mathbf{y}\rho(\mathbf{y})d\mathbf{y}}{\int_V \rho(\mathbf{y})d\mathbf{y}}$$

Given k points $\mathbf{z}_i, i = 1, ..., k$, we can define their associated Voronoi regions $\hat{V}_i, i = 1, ..., k$. On the other hand, given the regions $\hat{V}_i, i = 1, ..., k$, we can define their mass centroids $\mathbf{z}^*, i = 1, ..., k$. Here, we are interested in the situation where

$$\mathbf{z}_i = \mathbf{z}_i^*, \quad i = 1, ..., k$$

i.e., the points $\mathbf{z}_i$ that serve as generators for the Voronoi regions $\hat{V}_i$ are themselves the mass centroids of those regions. We call such a tessellation a **centroidal Voronoi tessellation**. This situation is quite special since, in general, arbitrarily chosen points in $\mathbb{R}^N$ are not the centroids of their associated Voronoi regions. See Fig. 4.2 for an illustration in two dimensions.
One may ask, how does one find centroidal Voronoi tessellations?
Consider the following problem. Given

- a region $\Omega \subseteq \mathbb{R}^N$,
- a positive integer k,
- a density function ρ, defined for $\mathbf{y}$ in Ω,

find

- k points $\mathbf{z}_i \in \Omega$,
- k regions V_i that tessellate Ω,

such that simultaneously for each i,

- V_i is the Voronoi region for $\mathbf{z}_i$,
- $\mathbf{z}_i$ is the mass centroid of V_i.

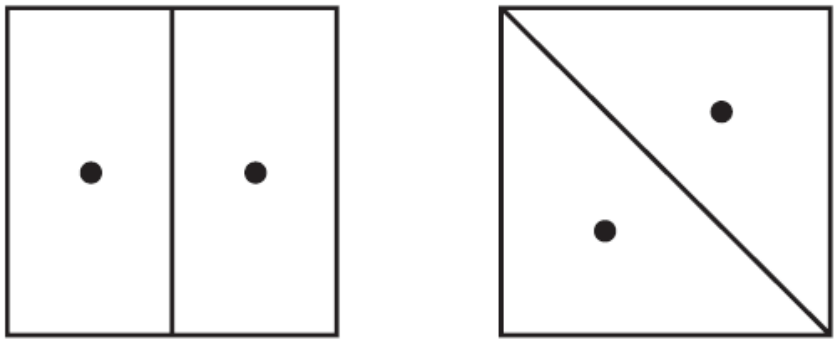


Figure 4.3.: Two centroidal Voronoi tessellations of a square. The points $\mathbf{z}_1$ and $\mathbf{z}_2$ are the centroids of the rectangles on the left or of the triangles on the right.

The solution of this problem is in general not unique. For example, consider the case of $N = 2$, $\Omega \in \mathbb{R}^2$ a square, and $\rho \equiv 1$. Two solutions are depicted in Fig. 4.3; others may be obtained through rotation. Another example is provided by the three regular tessellations of $\mathbb{R}^2$ into squares, triangles, and hexagons.

4.1.1 Lloyd's algorithm

Here, we discuss a deterministic approach for the determination of centroidal Voronoi tessellations of a given set: the Lloyd's Method.
Given a set Ω, a positive integer k, and a probability density function ρ defined on $\overline{\Omega}$,

1. select an initial set of k points $\{\mathbf{z}_i\}_{i=1}^k$, e.g., by using a Monte Carlo method;
2. construct the Voronoi tessellation $\{V_i\}_{i=1}^k$ of Ω associated with the points $\{\mathbf{z}_i\}_{i=1}^k$;
3. compute the mass centroids of the Voronoi regions $\{V_i\}_{i=1}^k$ found in step 2; these centroids are the new set of points $\{\mathbf{z}_i\}_{i=1}^k$.
4. If this new set of points meets some convergence criterion, terminate; otherwise, return to step 2.

Lloyd's algorithm may be viewed as a fixed point iteration. For example, consider the case of $\Omega \subset \mathbb{R}^N$. Let the mappings $\mathbf{T}_i : \mathbb{R}^{kN} \longrightarrow \mathbb{R}^N, i = 1, ..., k$, be defined by

$$\mathbf{T}_i(\mathbf{Z}) = \frac{\int_{V_i(\mathbf{Z})} \mathbf{y}\rho(\mathbf{y})d\mathbf{y}}{\int_{V_i(\mathbf{Z})} \rho(\mathbf{y})d\mathbf{y}}$$

where
$\mathbf{Z} = \begin{bmatrix}\mathbf{z}_1 & ... & \mathbf{z}_k\end{bmatrix}^T$ and $V_i(\mathbf{Z})$ = Voronoi region for $\mathbf{z}_i, i = 1, ..., k$.
Let the mapping $\mathbf{T}_i : \mathbb{R}^{kN} \longrightarrow \mathbb{R}^{kN}$ be defined by

$$\mathbf{T} = \begin{bmatrix}\mathbf{T}_1 & ... & \mathbf{T}_k\end{bmatrix}^T$$

Clearly, centroidal Voronoi tessellations are fixed points of $\mathbf{T}(\mathbf{Z})$.

4.2 Vietoris-Rips Complex Theory

In topology, the Vietoris-Rips complex, also called the Vietoris complex or Rips complex, is an abstract **simplicial complex** that can be defined from any metric space K and distance ε by forming a simplex for every finite set of points that has diameter at most ε (see [14]). Thus, we need to introduce before the concept of "simplex" and then explain simplicial complexes and their relations with graphs.

4.2.1 Simplexes

In geometry, a **simplex** (plural: simplexes or simplices) is a generalization of the notion of a triangle or tetrahedron to arbitrary dimensions. Specifically, a k-**simplex** is a k-dimensional polytope which is the convex hull of its $k+1$ vertices. More formally, suppose the $k+1$ points $u_0, \ldots, u_k \in \mathbb{R}^k$ are affinely independent, which means $u_1 - u_0, \ldots, u_k - u_0$ are linearly independent. Then, the simplex determined by them is the set of points

$$C = \left\{ \delta_0 u_0 + \cdots + \delta_k u_k | \delta_i \geq 0, 0 \leq i \leq k, \sum_{i=0}^{k} \delta_i = 1 \right\} \tag{4.1}$$

For example, a 2-simplex is a triangle, a 3-simplex is a tetrahedron, and a 4-simplex is a 5-cell (see Fig. 4.4). A single point may be considered a 0-simplex, and a line segment may be considered a 1-simplex. A simplex may be defined as the smallest convex set containing the given vertices. A **regular simplex** is a simplex that is also a regular

Figure 4.4.: Some simplexes: point (0-simplex), segment (1-simplex), triangle (2-simplex), tetrahedron (3-simplex) and 5-cell (4-simplex).

polytope. A regular n-simplex may be constructed from a regular $(n-1)$-simplex by connecting a new vertex to all original vertices by the common edge length.
In topology and combinatorics, it is common to "glue together" simplices to form a simplicial complex. The associated combinatorial structure is called an abstract simplicial complex, in which context the word "simplex" simply means any finite set of vertices.

4.2.2 Simplicial complexes

A **simplicial complex** is a set K of finite sets such that if $\sigma \in K$ and $\tau \subseteq \sigma$, then $\tau \subseteq K$. For every $\tau \subseteq \sigma \in K$, we say τ is a **face** of σ, its **coface**. The (-1)-simplex $\emptyset$ is a face of any simplex. A simplex is maximal if it has no proper coface in K. If $\sigma \in K$ has cardinality $|\sigma| = k+1$, we call σ a k-simplex of dimension k, $\dim(\sigma) = k$. This name stems from our ability to realize a k-simplex geometrically as a k-dimensional subspace of $\mathbb{R}^d$, $d \geq k$, namely, the convex hull of $k+1$ affinely independent points. Given this view, a k-simplex is called a vertex, an edge, a triangle, or a tetrahedron for $0 \leq k \leq 3$, respectively. A simplicial complex may be embedded in Euclidean space as the union of its geometrically realized simplices such that they only intersect along shared faces.. In other terms, as stated by [16], a finite collection K of such simplices that maintains closure with respect to faces (i.e. $\sigma \in K$ implies that all faces are included in K) is called a simplicial complex, see Fig. 4.5.

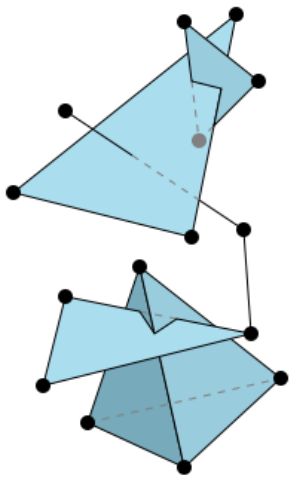


Figure 4.5.: A simplicial complex of order 3.

A **subcomplex** is a subset $L \subseteq K$ that is also a simplicial complex. An important subcomplex is the k-**skeleton** consisting of simplices in K of dimension less than or equal to k.

A **filtration** of a complex K is a sequence of nested subcomplexes $\emptyset = K_0 \subseteq K_1 \subseteq \cdots \subseteq K_m = K$.
A complex with a filtration is a **filtered complex**.

Suppose we are given a finite set of d-dimensional points $S \subseteq \mathbb{R}^d$, such as the set of 13 points in the plane in Fig. 4.6. The **Vietoris-Rips complex** (VR complex) $\mathcal{V}_\varepsilon(S)$ of S at scale ε is

$$\mathcal{V}_\varepsilon(S) = \{\sigma \in S | dist(u,v) \leq \varepsilon, \forall u \neq v \in \sigma\}$$

where d is the Euclidean metric. In other words, each simplex σ in $\mathcal{V}_\varepsilon(S)$ has vertices that are pairwise within distance ε.

4.2.3 Vietoris-Rips neighborhood graph

Associating the simplicial complexes to a neighborhood graph, it is possible to construct a Vietoris-Rips complex using an algorithm called "Two-phase construction". We do not enter in details of this method, illustrated in Fig. 4.6, such as the combinatorial process of expanding. However, it is useful to underline the relation between a Vietoris-Rips complex and a graph.

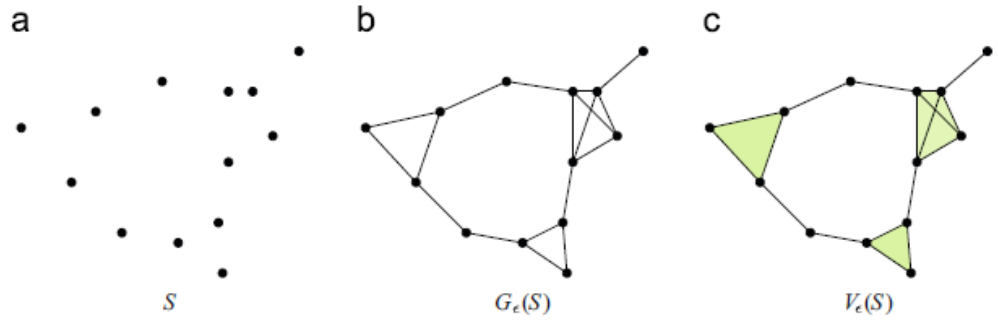


Figure 4.6.: Construction of the Vietoris-Rips complex in a 3D space. Our input (a) is a set of points S. First phase: the geometric process of going from (a) to a neighborhood graph(b). Second phase: the combinatorial process of expanding from the graph(b) to the Vietoris-Rips complex(c).

A **neighborhood graph** is $(\mathcal{G}), w$, where $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is an undirected graph,and $w : \mathcal{E} \longrightarrow \mathbb{R}$ is its weight function, defined on its edges. The 1-skeleton of the VR complex is a neighborhood graph.

In particular, given $S \subseteq \mathbb{R}^d$ and scale $\varepsilon \in \mathbb{R}$, the **VR neighborhood graph** is a neighborhood graph $(\mathcal{G}_\varepsilon(S), w)$, where $\mathcal{G}_\varepsilon(S) = (S, \mathcal{E}_\varepsilon(S))$ and
$\mathcal{E}_\varepsilon(S) = \{\{u,v\} | dist(u,v) \leq \varepsilon, u \neq v \in S\}$, $w(\{u,v\}) = dist(u,v), \forall \{u,v\} \in \mathcal{E}_\varepsilon(S)$.
Fig. 4.6 shows a VR neighborhood graph with 18 edges, where w is the length of the edges. We may expand any neighborhood graph up to a weight-filtered complex.

5

Theoretical coverage bounds

Some theoretical bound estimates about coverage in a rectangular scenario are well-known. Defined the rectangle with dimensions $a \times b$, we analyzed the two following types of coverage. We decided to set random dimensions in order to better show the behaviours in our simulations. For instance, $a = 99.1700$ and $b = 65.2236$ have been randomly chosen. Moreover, in this chapter, agents are modelled as a point with a sensing radius r. It has been set a sensing radius of $r = 10$, chosen in a proportional way to a and b.

5.1 Optimal coverage

This is a non-connected theoretical coverage which can allow to cover a rectangle with the minimum number of agents. In this type of coverage it is not necessary that robots are neighbours: they just have to cover all points of the given space with their visibility disks, see figures 5.1 and 5.2.

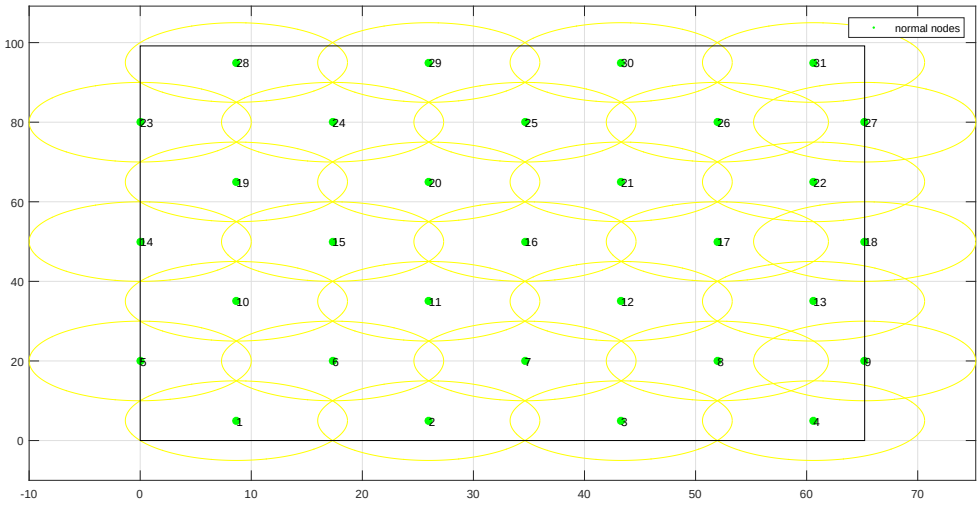


Figure 5.1.: Optimal coverage performed in a rectangle: 31 robots are needed.

It is possible to obtain an estimate of the minimum number of robots N_M utilized to accomplish an optimal coverage (i.e. a lower bound). These formulas have been found

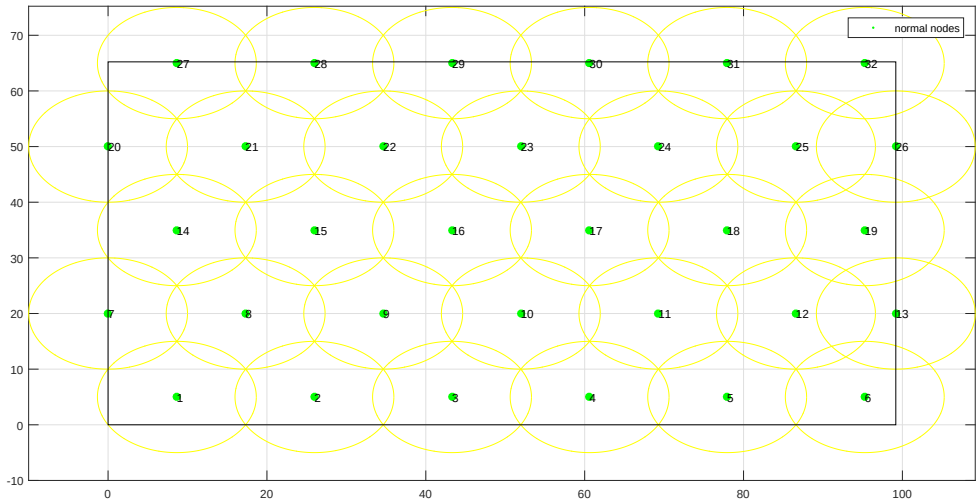


Figure 5.2.: Optimal coverage performed in the same rectangle with a swap of dimensions: 32 robots are needed. This simulation has to be discarded at light of Fig. 5.1, because a larger number of robots has been used.

trying to minimize the number of barycentres for each local[1] triangular structure created:

$$N_M = \begin{cases} \left\lceil \frac{2a+r}{3r} \right\rceil \left\lceil \frac{b}{\sqrt{3}r} \right\rceil - 1 & a > r, b > r \\ \left\lceil \frac{a}{\sqrt{4r^2 - b^2}} \right\rceil & a > r, b \leq r \\ \left\lceil \frac{b}{\sqrt{4r^2 - a^2}} \right\rceil & b > r, a \leq r \\ 1 \quad otherwise \end{cases} \tag{5.1}$$

Note that, in order to apply (5.1), one must repeat twice the calculations swapping the dimension a and b and keeping $N_M = min(N_{M1}, N_{M2})$. For instance, in simulations shown in figures 5.1 and 5.2 we respectively obtain $N_{M1} = 27$ and $N_{M2} = 29$. Therefore, at least $N_M = 27$ agents are needed to cover the rectangle (actually 31 agents have been deployed).

5.2 Optimal connected coverage

This is a theoretical coverage which can allow to cover a rectangle with the minimum number of connected agents. In this type of coverage it is necessary that robots compose a connected network. They also have to cover all points of the given space with their visibility disks, see figures 5.3 and 5.4. Even here, it is possible to obtain some formula about an estimate of the minimum number of robots N_{MC} utilized to accomplish an optimal coverage. These formula have been found trying to minimize the number

[1] in terms of proximity over the given space

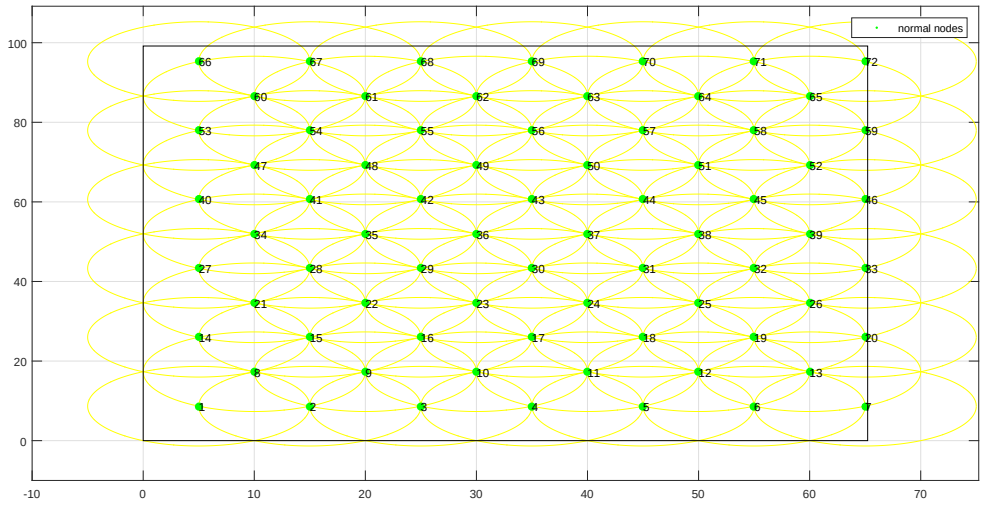


Figure 5.3.: Optimal connected coverage performed in a rectangle: 72 robots are needed.

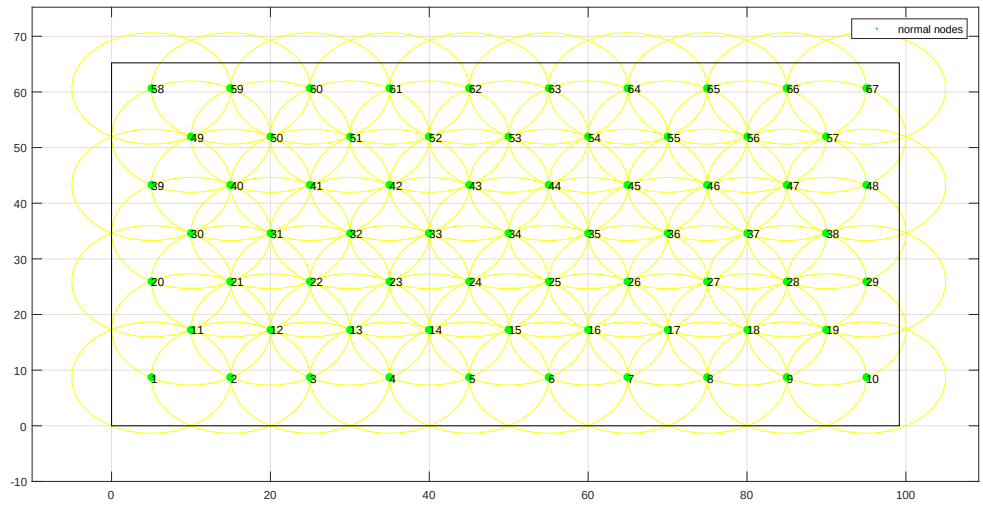


Figure 5.4.: Optimal connected coverage performed in the same rectangle with a swap of dimensions: 67 robots are needed. The previous simulation has to be discarded at light of Fig. 5.3 because a larger number of robots is used there.

of intersections which are generated by the circumferences with radius r during the deployment:

$$N_{MC} = \begin{cases} \left\lceil \dfrac{2a - \sqrt{3}r}{\sqrt{3}r} \right\rceil \left\lfloor \dfrac{b}{r} \right\rfloor + 1 & a > r, b > r \\ \left\lfloor \dfrac{a}{r} \right\rfloor & a > r, b \leq r \\ \left\lfloor \dfrac{b}{r} \right\rfloor & b > r, a \leq r \\ 1 & otherwise \end{cases} \tag{5.2}$$

Note that, in order to apply (5.2), one must repeat twice the calculations swapping the dimension a and b and keeping $N_{MC} = min(N_{MC1}, N_{MC2})$. For instance, in simulations shown in figures 5.3 and 5.4 we respectively obtain $N_{MC1} = 67$ and $N_{MC2} = 64$. Therefore, at least $N_{MC} = 64$ agents are needed to cover the rectangle (actually 67 agents have been deployed).
Important observation: one can immediately realize that in the context of optimal connected coverage more robot are needed to well attain the task than in the optimal coverage background, that is $N_M \leq N_{MC}$. Since our algorithm works using connections, we will use (5.2) only to build up some comparisons. Moreover, we will often set up scenarios where the sensing radius is negligible with respect to the room dimensions ($r \ll a, b$); therefore, (5.2) boils down to

$$\boxed{\tilde{N}_{MC} \cong \left\lfloor \frac{2ab}{\sqrt{3}r^2} \right\rfloor = \left\lfloor \frac{2\pi A_{scenario}}{\sqrt{3}A_{sensing}} \right\rfloor}$$

which can be understood as an approximate estimate of theoretical lowest bound for rectangular scenarios (A indicates the area of a surface).

6

Controllability for nonlinear systems

In this chapter we introduce some elements of Non-linear System Theory (see [17]): this will be used in order to obtain a proper dynamic model for agents which need to move independently for attaining a complete coverage on a surface.

6.1 Basic concepts

Consider a system, fully characterized by a finite number of real variables $x_1, x_2, ..., x_n$, satisfying a given set of equality or inequality constraints. If our system is to represent a vehicle or a rocket, these real variables might specify its position and orientation in space, but also more delicate matters, such as the amount of remaining fuel or covered distance, might occasionally be suitable to consider. These real numbers are called **state variables or generalized coordinates**. It is convenient to think about the state of the system as a point $x = (x_1, x_2, ..., x_n)$ in an n-dimensional space. A point that fulfills all the constraints on the state variables is called an admissible state. The set of all admissible states is called state space and denoted $\mathcal{X}$. Next, we shall introduce the state dynamics and the control. The **state dynamics**, is a prescribed set of rules that governs the time evolution of the state variables. It indicates how the state variables will change over time, given a current state and current input. It is mathematically appealing and physically motivated, to focus our attention on systems whose behavior are governed by nonlinear ordinary differential equations (ODEs)

$$\dot{x}(t) = f(x(t), u(t)) \tag{6.1}$$

where $f(x, u) \in \mathcal{C}^\infty$ is the **system dynamics** and $u = (u_1, u_2, ..., u_m)$ is the **control vector** that can be thought of as a point in an m-dimensional space, the control space. More generally, we shall assume that the control u, belongs to a prescribed subset $\mathcal{U} \in \mathbb{R}^m$ and that the **control manifold**, $\mathcal{U}$, is constant, i.e. independent of current state x and time t. In most control problems, the constraints on the control u, commonly arise from technological limitations. For instance, a rocket's thrust magnitude as well as a car's turning radius, are bounded. This type of limitations typically restrict the control space to polyhedra or polyhedral cones in $\mathbb{R}^m$. One (often overlooked) way of handling such control constraints, is replacing $\mathcal{U}$ by a non-constrained control space. Not seldom, one encounters the following type of control constraint

$$|u_i| \leq 1, \quad i \in 1, 2, ..., m$$

Then by defining $u_i = \cos\tilde{u}_i$, we have moved the control into a space in which $\tilde{u}_i \in \mathbb{R}$, thus unconstrained. The price that one has to pay for using this transformation trick, is that, apart from the obvious redundancy, any possible intuitive or physical interpretation of the control input might be lost. Another restriction imposed on the control is that $u(t)$ has to be a piecewise continuous function. A control that fulfills both of the criteria mentioned, i.e. belongs to the control manifold $\mathcal{U}$, and is piecewise continuous, is called an **admissible control**. Assuming that the system dynamics is a smooth function, i.e. belongs to $\mathcal{C}^\infty$, specifying an admissible control on a time interval $[t_i, t_f]$, corresponds to fully determining the time history of the state vector in that time interval. The time evolution of $x(t)$ is called the path or state trajectory, along which the system moves. The state trajectory is denoted $x(\cdot)$ while the corresponding control function $u(t)$, $t \in [t_i, t_f]$, is denoted as $u(\cdot)$. If the path stays within the system's state space $\mathcal{X}$, we call it an admissible path. Concerning the system dynamics particularized by equation (6.1), we shall pay extra attention to systems, governed by **control affine system dynamics** of form

$$\dot{x} = f(x(t), u(t)) = \Upsilon(x(t))u(t) + \Upsilon_0(x(t)) \tag{6.2}$$

where $\Upsilon_0(x(t)) \in \mathcal{R}^n$ is called the drift and $\Upsilon(x(t)) \in \mathcal{R}^{n\times m}$ is a matrix with **vector fields** $\Upsilon_i, i = 1...m$, as its columns. A vector field Υ , is a function that, with each point x in the state space $\mathcal{X}$, associates a vector belonging to that point's tangent space, $T_x\mathcal{X}$.

6.2 Definitions

Some basic definitions on Manifold Theory according to [17] follow.

6.2.1 Tangent space and Tangent Bundle

The tangent space *at a point x, $T_x\mathcal{X}$, is defined as the space of all possible velocities of trajectories, passing through the point x. Throughout this report, the tangent space is identical with the space $\mathbb{R}^n$ of column vectors.*
Originating from this, we define the tangent bundle as

$$T\mathcal{X} = \bigcup_x T_x\mathcal{X}$$

This object is needed to understand the virtual directions where the state can evolve.

6.2.2 Vector Field

A vector field Υ, on a manifold $\mathcal{X}$, is defined as a map $\Upsilon : \mathcal{X} \in x \longmapsto \Upsilon(x) \in T_x\mathcal{X}$.

Notice that $f(x, u) \in T_x\mathcal{X}$, is a special vector that for a fixed u, represents the possible infinitesimal change in the state variables with respect to time. It will be convenient to define the set $f(x,\mathcal{U})$ of all vectors $f(x, u)$, $u \in \mathcal{U}$. This set spans a subspace of $T_x\mathcal{X}$.

6.2.3 Distribution

A distribution *is an assignment* $\Delta(x)$ *of a subspace of* $T_x\mathcal{X}$ *for each* $x \in \mathcal{X}$*. Given the set of vector fields* $\Upsilon_1, \Upsilon_2, ..., \Upsilon_m$*, the distribution* $\Delta \in T_x(\mathcal{X})$ *is defined by*

$$\Delta = span\,\{\Upsilon_1, \Upsilon_2, ..., \Upsilon_m\}$$

The rank of a distribution at a point, equals the maximum number of linearly independent vectors among $\Upsilon_i, i \in 1, 2, ..., m$ *at this point.*

A distribution Δ, can thus be considered to assign a vector space to each point x, namely the space of all accessible velocities at the configuration represented by x.
From definition (6.2.3), it follows that any vector field in Δ, can be expressed as a linear combination of the Υ_is, which, when independent, serve as a basis for Δ.

The subspaces $\Delta(x)$, typically do not join or fit together in a coherent manner to form the tangent bundle of a smooth submanifold of $\mathcal{X}$.
The distribution is then not the tangent bundle of a state submanifold. In such cases, if the task is to transfer the system between two prescribed configurations $\mathcal{X}_i$ and $\mathcal{X}_f$, it is possible to reach final points $\mathcal{X}_f$ in a set of higher dimension than $dim\Delta(Xi)$. To study this more formally, one of the most fundamental operations than can be performed on vector fields, namely the Lie bracket, has to be introduced.

6.2.4 Lie brackets

Given two vector fields X *and* Y*, another vector field, called the Lie bracket and denoted by* $[X, Y]$*, can be defined. The Lie bracket is computed by*

$$[X, Y] = DY \cdot X - DX \cdot Y$$

where DX *and* DY *are the Jacobian matrices of* X *and* Y *respectively, i.e.*

$$DX = \begin{bmatrix} \frac{\partial X_1}{\partial x_1} & \cdots & \frac{\partial X_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial X_n}{\partial x_1} & \cdots & \frac{\partial X_n}{\partial x_n} \end{bmatrix}$$

Lie brackets have the following two basic properties

1. $[X, Y] = -[Y, X]$ *(skew-symmetric)*

2. $[[X, Y], Z] + [[Y, Z], X] + [[Z, X], Y] = 0$ *(the Jacobi identity)*

Lie Brackets are the key mathematical tool used for understanding the controllability of a non-linear system: with some computations it will be possible to determine a base in a Control Lie Algebra.

6.2.5 Control Lie Algebra

The Control Lie Algebra associated with a distribution Δ, is denoted $CLA(\Delta)$ and is the smallest distribution containing Δ that is closed under the Lie bracket operation.

Finding a basis of the CLA, is generally a tedious process. There exist however several systematic approaches for generating such a basis, one of which is called the Phillip-Hall basis. The reason why we need to seek for a base in a CLA is to compute the Lie-Algebraic Condition, which tells whether an Affine Drift-less Symmetric System is controllable or not.
We may now proceed to define the concepts of *reachability, accessibility and controllability*. In each case, three different types will be considered, *viz. exact-time*, *small-time* and *finite-time*. In addition, each of these concepts will appear in a more restricted version (termed *local*), where we, not only pay attention to what points are reachable, accessible or controllable, but also restrict the corresponding state trajectory $x(\cdot)$ to stay in a *prescribed neighborhood* of the point.

6.2.6 Lie-Algebraic rank condition

The Lie-Algebraic rank condition is denoted by

$$\dim(CLA(\Delta(x))) = \dim(\mathcal{X})$$

which is equivalent to require $\dim(T_x\mathcal{X}) = \dim(\mathcal{X})$.

LARC is the main condition that must hold in order to prove the validity the propositions about Locally Exact-time Accessibility, Small-time Controllability, Controllability and Controllability of Affine Drift-less Symmetric Systems presented on the next section.

6.2.7 Reachability

The set of exact-time reachable *points from x, is defined as*

$$\mathcal{R}^*(x,T) = \{\hat{x} \in \mathcal{X} | \exists u(\cdot) : x(0) = x \text{ and } x(T) = \hat{x}\}$$

Originating from this definition, we have:
set of small-time reachable *points from x* $\mathcal{R}(x,T) = \bigcup_{t \leq T} \mathcal{R}^*(x,t)$
set of finite-time reachable *points from x* $\mathcal{R}(x) = \bigcup_{t \in \mathbb{R}^+} \mathcal{R}^*(x,t)$

6.2.8 Local reachability

The set of locally exact-time reachable points from x, is defined as

$\mathcal{R}^*(x,T) = \{\hat{x} \in N | \exists u(\cdot) : x(0) = x, x(T) = \hat{x}$ *and the corresponding trajectory* $x(\cdot) \in N\}$

where N is a prescribed neighborhood of x.

Originating from this definition, we have:
set of small-time reachable *points from* x $\mathcal{R}(x,T,N) = \bigcup_{t\leq T,N} \mathcal{R}^*(x,t)$
set of finite-time reachable *points from* x $\mathcal{R}(x,N) = \bigcup_{t\in\mathbb{R}^+} \mathcal{R}^*(x,t,N)$

6.2.9 Accessibility and controllability

Here follows a recap of the previous definitions in Table 6.1. Now, we can proceed by

	Exact-time	Small-time	Finite-time
Reachability	$\mathcal{R}^*(x,T) = \{\hat{x} \in \mathcal{X} \mid \exists u(\cdot) : x(0) = x \text{ and } x(T) = \hat{x}\}$	$\mathcal{R}(x,T) = \bigcup_{t\leq T} \mathcal{R}^*(x,t)$	$\mathcal{R}(x) = \bigcup_{t\in\mathbb{R}^+} \mathcal{R}^*(x,t)$
Local reachability	$\mathcal{R}^*(x,T) = \{\hat{x} \in N \mid \exists u(\cdot) : x(0) = x, x(T) = \hat{x} \text{ and the } \textit{corresponding trajectory } x(\cdot) \in N\}$	$\mathcal{R}(x,T,N) = \bigcup_{t\leq T,N} \mathcal{R}^*(x,t)$	$\mathcal{R}(x,N) = \bigcup_{t\in\mathbb{R}^+} \mathcal{R}^*(x,t,N)$

Table 6.1.: Definition of reachability.

defining accessibility and controllability. Because of the similarities in the formulation of these definitions, they will be presented by means of Table 6.2.

Accessible	for all $x \in \mathcal{X}$, $\mathcal{R}^*(x,T)$ contains a non-empty open set for all $T > 0$.	for all $x \in \mathcal{X}$, $\mathcal{R}(x,T)$ contains a non-empty open set for all $T > 0$.	for all $x \in \mathcal{X}$, $\mathcal{R}(x)$ contains a non-empty open set.
Locally accessible	for all $x \in \mathcal{X}$, $\mathcal{R}^*(x,T,N)$ contains a non-empty open set for all $T > 0$ and any prescribed neighborhood N.	for all $x \in \mathcal{X}$, $\mathcal{R}(x,T,N)$ contains a non-empty open set for all $T > 0$ and any prescribed neighborhood N.	for all $x \in \mathcal{X}$, $\mathcal{R}(x,N)$ contains a non-empty open set for any prescribed neighborhood N.
Controllable	for all $x \in \mathcal{X}$, $\mathcal{R}^*(x,T)$ contains a full neighborhood of x for all $T > 0$.	for all $x \in \mathcal{X}$, $\mathcal{R}(x,T)$ contains a full neighborhood of x for all $T > 0$.	for all $x \in \mathcal{X}$, $\mathcal{R}(x)$ contains a full neighborhood of x.
Locally controllable	for all $x \in \mathcal{X}$, $\mathcal{R}^*(x,T,N)$ contains a full neighborhood of x for all $T > 0$ and any prescribed neighborhood N.	for all $x \in \mathcal{X}$, $\mathcal{R}(x,T,N)$ contains a full neighborhood of x for all $T > 0$ and any prescribed neighborhood N.	for all $x \in \mathcal{X}$, $\mathcal{R}(x,N)$ contains a full neighborhood of x for any prescribed neighborhood N.

Table 6.2.: Definition of accessibility and controllability.

6.3 Propositions

Some relevant propositions on Manifold Theory according to [17] follow.

6.3.1 Locally Exact-time Accessibility

If an affine control system satisfies $CLA(\Delta) = T\mathcal{X}$, then the system is locally exact-time accessible.

The simplest approach to show small-time controllability for the special important case of *affine systems* is by studying the linearized system. It should be noted that small-time controllability naturally implies controllability and even (small-time) accessibility.

6.3.2 Small-time Controllability

If an affine system of form (6.2), is drift-free at $\hat{x}$ (i.e. $\Upsilon_0(\hat{x}) = 0$) and the linearization at $\hat{x}$ and $u = 0$

$$\dot{z} = \frac{\partial \Upsilon_0}{\partial x}(\hat{x})z + \Upsilon(\hat{x})u$$

is controllable (i.e. satisfies the Kalman rank condition), then the system is small-time controllable from $\hat{x}$.

However, no conclusion can be drawn about the controllability properties of the nonlinear system, if the linearized counterpart fails to be controllable. In such cases, the following preposition can be exploited.

An affine system with no restriction on the size of the control is small-time controllable *if the rank of the control Lie algebra $CLA(\Delta)$ equals n (the dimension of $\mathcal{X}$), for all $x \in \mathcal{X}$, i.e. LARC holds.*

6.3.3 Controllability

A drift-less affine system remains controllable (but not necessarily small-time controllable) if
a) $CLA(\Delta) = T\mathcal{X}$ and
b) the convex hull of the control set U contains the origin in $\mathbb{R}^m$

By virtue of the foregoing two propositions, checking the controllability properties of a system requires the analysis of the Control Lie Algebra associated with it. Checking the LARC, on a control system, is a very fundamental and useful tool for determining a systems controllability properties. In section (7.5.3), the controllability analysis of a free-wheeled car will be carried out by means of LARC.

6.3.4 Controllability of Affine Drift-less Symmetric Systems

If the vector fields of a drift-less symmetric control system of form

$$\dot{x} = \Upsilon(x)u$$

have LARC at all $x \in \mathcal{X}$, then it is locally controllable.

This is the proposition we need to use during the continuation to prove the controllability of the front-wheeled car: the model we adopted to describe the robots' dynamics.

Part III

Algorithm design

7

Models

In this chapter we present notation, preliminaries and models assumed.

7.1 Scenario

A **scenario** is a virtual space where actions take place and in our case it is modelled as a set of $S > 2$ segments: these can be designated as walls or inner barriers. A segment has to be simply understood as a couple of two vertices. Each vertex is represented by a couple of two coordinates (x_s, y_s), $s = 1...V$ on the plane, where V is the number of vertices which can be arbitrary decided: a closed scenario, that is forcing $S = V$ for the external polygon structure, is the unique constraint for guaranteeing a coverage problem which makes sense.

7.1.1 Base station

A **base station** is the point from which all robots depart during coverage. It also represents the first agent. Base station coordinates (x_0, y_0) must be chosen in the inner part of the scenario for guaranteeing a coverage that can be terminated in a finite time. We assume that base station cannot be removed during the research for redundant robots.
A scenario can have one ore more base stations.

7.2 Robots

A robot or agent is an automatic device which has to be deployed in a hypothetical real coverage. In our algorithm it is modelled by a circle with radius R, diameter ϕ and centre C. Robots are also provided with a camera installed on the centre C with a **sensing radius** r_v. These are used to measure angles a robot creates with its neighbours. Two robots a_1, a_2 are neighbours if and only if the two following statements hold:

1. their centres are distant less than the visibility radius, *i.e.* $\|C_{a_1} - C_{a_2}\|_2 < r_v$, see Fig. 7.1;

2. there is no barrier between them, *i.e.* $\overline{C_{a_1} C_{a_2}}$ does not intersect any segment of the scenario.

Two robots can be neighbours even though between them another robot has already been placed, that is robots do not occlude each other.

Figure 7.1.: (a) Robots can't see each other, and hence have no way of detecting that their disks of visibility overlap. (b) Robots can see each other, and hence know that their disks of visibility overlap. Visibility is represented by the dotted magenta line.

7.2.1 Angle measurements

Angle measurements are performed by cameras installed on the robots. Let $\theta_{ij}^k \in [-\pi, \pi)$ be the angle $i\hat{k}j$, where i, j are neighour of k. Then, we define $\theta_{ij}^k = \theta_j^k - \theta_i^k$ measured in a clockwise direction from i to j. See Fig. 7.2.

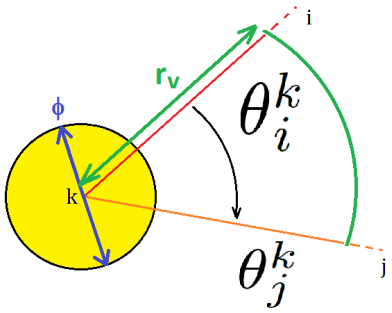


Figure 7.2.: Local bearing angle measurement

7.2.2 Local Bearing-Based Controller

The robots are controlled usinga bearing-based visual homing controller. This controller utilizes a gradient descent approach where desired bearing angles to landmarks are used to drive robots. The distances between a robot and its respective landmarks are not known. The only known information is the bearing angles. With the proper selection of the cost functional for the optimization process, the gradient of the path from start to goal (the velocity control command for robot i) is given by $v^i = K \sum_{j \in \mathcal{N}^i} \left(\theta_{j,des}^i - \theta_j^i\right)$ where, $\mathcal{N}^i$ is the list of robots that are neighbors of i, and which can be used as landmarks, $\theta_{j,des}^i$ is the desired bearings with landmark j, and K is a gain. Note that this velocity can be computed in the local coordinate frame of robot i. This controller converges to the goal configuration when the number of landmarks is greater than or equal to two and not co-linear with the goal location. In our implementation, the controller incorporates adaptive gain scaling in order to obtain faster convergence. We also use adaptive landmark detection depending on a robot's neighbor list while moving.
In our simulations we assume that controllers always work properly without any virtual implementation of the velocities.

7.2.3 Contacts and touching sensors

Each robot is equipped by N_s touching sensors which allow to understand if a contact occurs and provide a rough estimate of the impact direction (within an angle error of $\frac{\pi}{N_s}$).

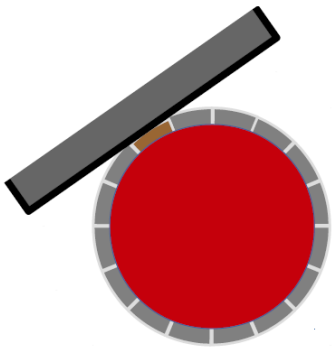


Figure 7.3.: The touch/contact sensors (grey protrusions) at the base of a robot (red). Contact with an obstacle or another robot triggers only one touch sensor providing a rough estimate of the direction of contact.

7.3 Robotic network

A robot can be pushed, in order to be deployed, by following one of the shortest paths from the base station to the fence, which is defined as the set of most external frontier robots already placed. Thus, a graph $G = (\mathcal{N}, E)$ is very useful in order to cope with the covering robotic network structure. Each node $\nu \in \mathcal{N}$ represents a robot and an edge $e \in E$ between two robots (ν_1, ν_2) is created if only if (ν_1, ν_2) are neighbours.

7.3.1 Coverage policy

We adopt an **hexagonal-packing-based coverage policy** discussed in [19] because it is the best approach in an obstacle-free planar environments. For this reason and since we use robots described above, we decided to apply the Vietoris-Rips Complex in order to achieve deployments. This theory is based on assembling simplicial complex: geometrical structures that allow us to build up a good strategy to create the graph $\mathcal{G}$. Simplicials used here are:

- type 0: a point, which represents a node on a plane;
- type 1: a segment, which represents an edge on a plane;
- type 2: a triangle, in particular equilateral triangles to guarantee hexagonal packing policy.

Geometrical structures made up by simplicial of order less or equal to m are called H_m homology. A simplicial complex is an homology with a metric. In our case the latter is

imposed to be r_v, the radius of visibility, so we need to work using a particular simplicial complex, say R_{r_v}, over the H_2 homology.

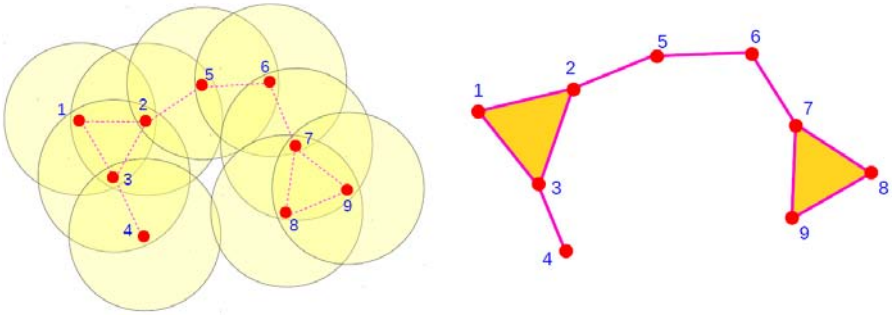


Figure 7.4.: Nine robots, their disks of visibility and the corresponding abstract simplicial complex, R_{r_v}.

7.4 Environment interactions

An important issue in this framework is managing impacts between two robots and between scenario and a robot.

7.4.1 Impacts between a robot and a wall

Impacts between a robot and a wall can be modelled by intersections between a segment $\overline{AB}$ with coordinates $(x_A, y_A), (x_B, y_B)$ and a circumference given a precise trajectory vector $v = \lambda u$ such that $\lambda \geq \phi$, $u \in \mathbb{R}^2$, $\|u\|_2 = 1$, of the translation of the centre C of the latter. As Fig. 7.5 shows we need to solve this geometrical problem:

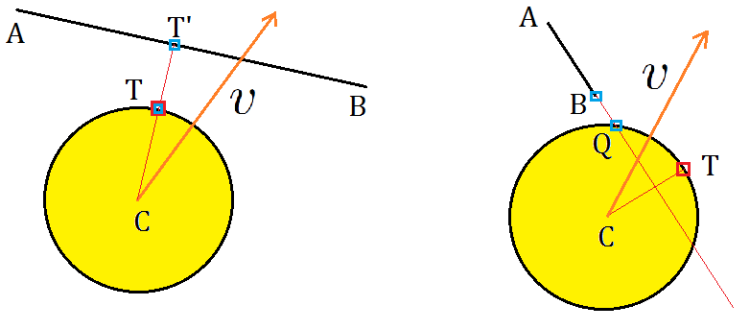


Figure 7.5.: T is generally the impact point of a robot centred on C moving towards a wall (segment $\overline{AB}$) along a vector v, but, if the projection T' of T does not belong to $\overline{AB}$, then the impact point becomes one of the vertices A or B, in particular the closest one to the circumference.

- find T as the tangent point to the circumference according to the closest parallel line to $\overline{AB}$ and secondly the distance $dist(T, T')$, even though the projection $T' \notin \overline{AB}$;

- compute solutions of this system of equations for each vertex $\mathrm{v} \in \{A, B\}$:

$$\begin{cases} \|C - Q\|_2 = R \\ \mathrm{v} = Q + \mu_{\mathrm{v}} u \\ \mu_{\mathrm{v}} \geq 0 \end{cases} \tag{7.1}$$

  in order to find the minimum distance point $Q^* = \underset{\mathrm{v} \in \{A,B\}}{\mathrm{argmin}} \mu_{\mathrm{v}}$ between the circumference and a vertex and the minimum distance $\mu^* = min(\mu_{\mathrm{v}})$ between Q^* and the circumference;

- compute the maximum translation along v as the minimum of $dist(T, T')$ and μ^*.

In conclusion, an important observation should be done. When points T and T' or A, B and Q^* are coincident, one has to be careful in deciding whether a robot can move or not along v during the computations. In fact, one must verify if the destination, head of the trajectory vector v, can be reached in a geometrical locus of the plain where movement of the robot starting from C is permitted. See Fig. 7.6 for analysing the case[1] $T = T'$. In

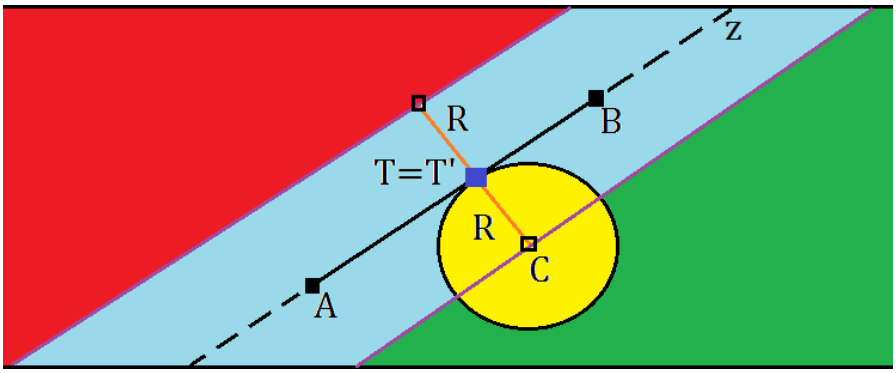


Figure 7.6.: Head of vector v can lays on red, light blue, green areas, purple lines or segment $\overline{AB}$. Over and between red zone and light blue zone movement of robot in C is not allowed. Otherwise, a test drive for deployment can be performed.

order to correctly understand the possibility of movement from C, simply linear equation systems have to be solved. Say D the head of the given trajectory vector v and z the extension of the segment $\overline{AB}$ as a straight line. Firstly, we check if $dist(D, z) < R$. If last statement holds then movement is not allowed. Secondly, if the previous assertion doesn't occur, we check for another condition: vector v, understood as a segment $\overline{CD}$, must not intersect z in order to have movement. Finally, in the rare case that D belongs to the purple line passing through C, we decide to accomplish movement.

7.4.2 Impacts between robots

Impacts between robots can be modelled by intersections between three circumferences and a precise moving direction of the centre of one of them, see Fig. 7.7. Let S, D, C

[1] We describe this case only because it is more difficult and interesting than the one about vertex coincidences. In order to solve the latter, one should verify if (7.1) has two particular solutions in μ_{v}: if $\mu_{\mathrm{v},1} = 0$ and $\mu_{\mathrm{v},2} > 0$ exist then movement from C along v is not allowed.

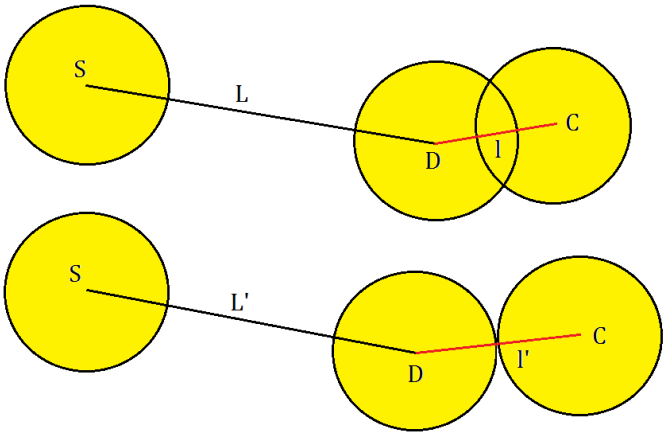


Figure 7.7.: Collision between two robots: the one centred in D, expanded from the stating position S, impacts on the one centred in C. The distance l should be less than the diameter d: in order to obtain this, we have to compute a new distance $L' < L$ which permits to get $l' \leq d$.

be respectively a source, a destination and an already-deployed robots. Define algebraic vectors $lu = D - C$ and $Lv = S - D$ such that $l, L > 0$ and $\|v\|_2, \|u\|_2 = 1$. We need to solve this geometrical problem in order to find $L' < L$ that guarantees $l' = \phi$. To do that, we set $\|S - C - L'v\|_2 = \phi$. Finally we compute the new optimal distance[2] $L^* = min(|L'_1|, |L'_2|)$, which is the destination robots should keep from its source in order to avoid collisions between other robots. Remember that L^* greater than the diameter d has to be guaranteed, otherwise destination robot collides with its same source.

7.5 Front-wheeled car model

In this section we introduce a dynamic model adopted for the robots. In particular, we chose the front-wheeled car model which is the evolution of the car-like robot model. Before starting, we need to add two important hypotheses:

1. **Rolling without slipping**: it is a customary to assume that the robot wheels do not slip. This assumption is legitimate in all moderate-speed scenarios.

2. **Maximum steering angle**: we further assume that we have a limitation on the maximum steering angle $|\delta| \leq \delta_{max}$, meaning that the turning radius for our car is lower bounded.

7.5.1 Car-like robot

From a driver's point of view, a vehicle has two control possibilities: the accelerator/brake pedal and the steering wheel. The accelerating factor is the linear velocity v, while the steering wheel specifies the angle between the front wheels and the main direction of the vehicle. We define this as the steering angle γ, which is our second control variable (see figure 7.8). In practice, the two front wheels are seldom exactly

[2] Two solutions for L', say L'_1 and L'_2, can be found.

parallel, why we might set γ to be the average of these two angles. Augmenting the steering angle, obviously captures the characteristics of a car even better. The car-like

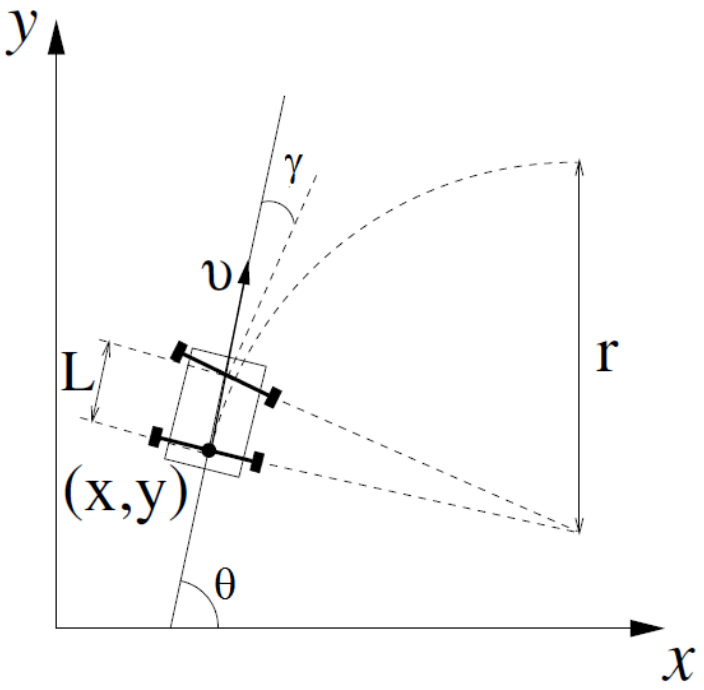


Figure 7.8.: The car-like robot model.

robot model, has the same state variables as a unicycle robot, i.e. $x = (x, y, \theta)$. The task is to find a set of differential equations f_1, f_2 and f_3, that properly describes the kinematics of the car.
Since the "rolling without slipping" assumption is valid even in this case, the motion in the $\mathbb{R}^2$-plane is dictated by the same equations as before, so that $\dot{x} = v\cos\theta$ and $\dot{y} = v\sin\theta$. Proceeding to the time evolution of the orientation angle $\dot{\theta}$, let's denote the distance traveled by the vehicle. Then $\dot{s} = v$, which is the speed of the car. As shown in figure, r represents the radius of a circle that will be traversed by (x, y), when the steering angle is fixed. Consequently $ds = rd\theta$. From simple trigonometry, we have $\tan\gamma = \dfrac{L}{r}$, which implies

$$d\theta = \frac{\tan\gamma}{L} ds$$

Dividing by dt and using the fact that $\dot{s} = v$, yields

$$\dot{\theta} = \frac{v}{L}\tan\gamma$$

Thus the dynamic for the car-like robot is

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \dfrac{v}{L}\tan\gamma \end{cases} \tag{7.2}$$

Since the linear velocity appears in the time evolution of all the state variables, this car model does not have the possibility to make arbitrary rotations while standing still in $\mathbb{R}^2$. It is only able to follow paths that are at least continuously differentiable.

7.5.2 Front-wheeled car

In the car-like robot model (equation (7.2)), we had the steering angle as an input and assumed to have direct control over its value. This corresponds to being able to move the front wheels instantaneously, which obviously contradicts our intuition about cars in general, and in many real-life applications this assumption is an unrealistic one. Whenever the value of γ changes discontinuously, the path traced out in the plane by (x, y), will have a discontinuity in its curvature. To put this right and make a car model that only generates smooth paths (i.e. belongs to $\mathcal{C}^2$), we might add the steering angle as an extra state variable and consider its derivative to be one of the input signals. Introducing this integrator-chain, results in a delay in the lateral control, such that γ is only allowed to change its values in a continuous manner.
We thus have to consider a four-dimensional state space, in which each state is represented as $\mathbf{x} = (x, y, \gamma, \theta) \in \mathcal{X} \subseteq \mathbb{R} \times \mathbb{R} \times \mathcal{S}^1 \times \mathcal{S}^1$. The system dynamics for the front-wheeled car model then becomes

$$\begin{cases} \dot{x} = v \cos\theta \\ \dot{y} = v \sin\theta \\ \dot{\gamma} = \omega \\ \dot{\theta} = \dfrac{v}{L} \tan\gamma \end{cases} \tag{7.3}$$

where, as usual, v represents the linear velocity, while ω is the angular velocity of the steering angle. By setting

$$\Upsilon_1 = \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \\ \dfrac{\tan\gamma}{L} \end{bmatrix} \quad \Upsilon_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

we may rewrite equation (7.3) as

$$\dot{\mathbf{x}} = \Upsilon_1 v + \Upsilon_2 \omega = \Upsilon \cdot u \tag{7.4}$$

where, as before, $u = \begin{bmatrix} v & \omega \end{bmatrix}^T$ and Υ has Υ_i as its i-th column. Although it holds true that the method of introducing the former input signal γ, as a state variable and rather consider its time derivative to be the new control input, gives a more realistic model for describing a real car, one should ask weather such approach is justified, as long as we do not take similar action for the linear control, v. Is it reasonable to consider that we are able to dictate the instant value of the linear velocity of a car? Because of its weight, the inertia along the platform's main axis, is generally orders of magnitude larger than the resistance existing in the steering device, why it may be motivated to introduce an integrator-chain in the linear direction as well. But on the other hand,

allowing the steering angle to change instantaneously, corresponds to matter moving in a discontinuous manner in $\mathcal{X}$, which is physically offensive.

7.5.3 Controllability analysis of the front-wheeled car

Using manifold theory, we can provide some results on the controllability of the front-wheeled car by starting from equations (7.3) and (7.4). Using Lie Brackets seen in section 6.2.4, we have that

$$[\Upsilon_1, \Upsilon_2] = D\Upsilon_2 \cdot \Upsilon_1 - D\Upsilon_1 \cdot \Upsilon_2 = -\begin{bmatrix} 0 & 0 & 0 & -\sin\theta \\ 0 & 0 & 0 & \cos\theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \dfrac{1+\tan^2\gamma}{L} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = -\begin{bmatrix} 0 \\ 0 \\ 0 \\ \dfrac{1+\tan^2\gamma}{L} \end{bmatrix}$$

and

$$[\Upsilon_1, [\Upsilon_1, \Upsilon_2]] = -\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \dfrac{2(\tan\gamma + \tan^3\gamma)}{L} & 0 \end{bmatrix} \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \\ \dfrac{\tan\gamma}{L} \end{bmatrix} +$$

$$\begin{bmatrix} 0 & 0 & 0 & -\sin\theta \\ 0 & 0 & 0 & \cos\theta \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \dfrac{1+\tan^2\gamma}{L} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dfrac{1+\tan^2\gamma}{L} \end{bmatrix} = \begin{bmatrix} -\sin\theta\left(\dfrac{1+\tan^2\gamma}{L}\right) \\ \cos\theta\left(\dfrac{1+\tan^2\gamma}{L}\right) \\ 0 \\ 0 \end{bmatrix}$$

In order to check the linear dependency of the four vectors Υ_1, Υ_2, $[\Upsilon_1, \Upsilon_2]$ and $[\Upsilon_1, [\Upsilon_1, \Upsilon_2]]$, we calculate

$$\det \begin{bmatrix} \cos\theta & \sin\theta & 0 & \dfrac{\tan\gamma}{L} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -\dfrac{1+\tan^2\gamma}{L} \\ -\sin\theta\left(\dfrac{1+\tan^2\gamma}{L}\right) & \cos\theta\left(\dfrac{1+\tan^2\gamma}{L}\right) & 0 & 0 \end{bmatrix} = -\left(\frac{1+\tan^2\gamma}{L}\right)^2$$

which is different from zero, for all configurations $(x, y, \gamma, \theta) \in \mathcal{X}$. Thus, the system in (7.4), fulfills LARC in every point in the state space, and is thereby small-time controllable. Moreover, for the same reasons and by proposition (6.3.4), the front-wheeled car is globally controllable.

7.6 Events and dispatch

Now we see how to model events (relevant situations that appear after accomplished coverage, e.g. a blaze inside the scenario) and the main actions to take in order to perform a final dispatch around an event.
Firstly, we show how a mathematical function called **event function** can model an event, explaining how the sensing task is performed.
Secondly, we present a computation of the target points to reach around an event exploiting the unitary circle in a complex plane.
Finally we introduce a suitable nonlinear control law for attaining the **dispatch** that try to minimize a time cost functional.

7.6.1 Event function

An event E can be associated to a real function $f_E : \mathbb{R}^2 \longrightarrow \mathbb{R}$ of the coordinates (x, y) of the plane such that:

1. takes positive or null values;

2. has a centre $\mu_E = \begin{bmatrix} x_E & y_E \end{bmatrix}^T$ belonging to the plane already covered by the agents that represents the place of the event: this centre must be the unique *global* maximum (we prefer a maximum because it is physically consistent and mathematically regular);

3. defined as F_E the minimum amplitude for which the event E can be detected, if $f_E(x, y) < F_E$ for some displaces (x, y) then the event cannot be detected from those displaces;

4. lays on a bounded (compact, if we choose f without singularities) support $D \subset \mathbb{R}^2$ because of the previous observation, it is zero in $\mathbb{R}^2 \setminus D$ and different from zero otherwise;

5. is a smooth function on D: $f_E \in \mathcal{C}^1(D)$.

The values of $f(x, y)$ represent the intensity of the event.
Define $z = \begin{bmatrix} x & y \end{bmatrix}^T$. We choose

$$f_E(z) = \begin{cases} \dfrac{\sqrt{\det A_E}}{2\pi} \exp\left(-\dfrac{1}{2}(z-\mu_E)^T A_E (z-\mu_E)\right) & \text{if } z \in D \\ 0 \quad \text{otherwise} \end{cases} \tag{7.5}$$

where $A_E = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \in \mathbb{R}^2$ is a positive definite matrix which gives the amplitude of the event E. The maximum peak is reached for $z = \mu_E$ and its value is $\dfrac{\sqrt{\det A_E}}{2\pi}$. It is possible to show that if we take $A_E = \lambda_E I_2$ (as we will assume) with $\lambda_E > 0$ we obtain

a contour set of f_E made of circumferences centred in μ with a maximum radius of

$$R_E = \sqrt{-\frac{2}{\lambda_E} \ln\left(\frac{2\pi}{\lambda_E} \min_{z \in D} f_E(z)\right)}$$

Indeed, if $A_E = \lambda_E I_2$ holds then (7.5) becomes

$$\boxed{f_E(z) = f_E(r) = \begin{cases} \frac{\lambda_E}{2\pi} \exp\left(-\frac{1}{2}\lambda_E r^2\right) & \text{if } r \leq R_E \\ 0 \quad \text{otherwise} \end{cases}} \tag{7.6}$$

by using polar coordinates (r, φ) centred in μ_E. See Fig. 7.9 to better understand the described event function.
As concerned the gradient of $f(z)$: it can be shown that

$$\nabla f_E(z) = f_E(z) \begin{bmatrix} 2a_{11}(x - x_E) + a_{12} + a_{21} \\ 2a_{22}(y - y_E) + a_{12} + a_{21} \end{bmatrix}$$

and choosing $A_E = \lambda_E I_2$ we obtain

$$\nabla f_E(z) = 2\lambda f_E(z) \begin{bmatrix} x - x_E \\ y - y_E \end{bmatrix}$$

which is zero if and only if $z = \mu_E$: this is a useful condition for searching the maximum μ_E if one decide to use some gradient-descent based optimization algorithms.

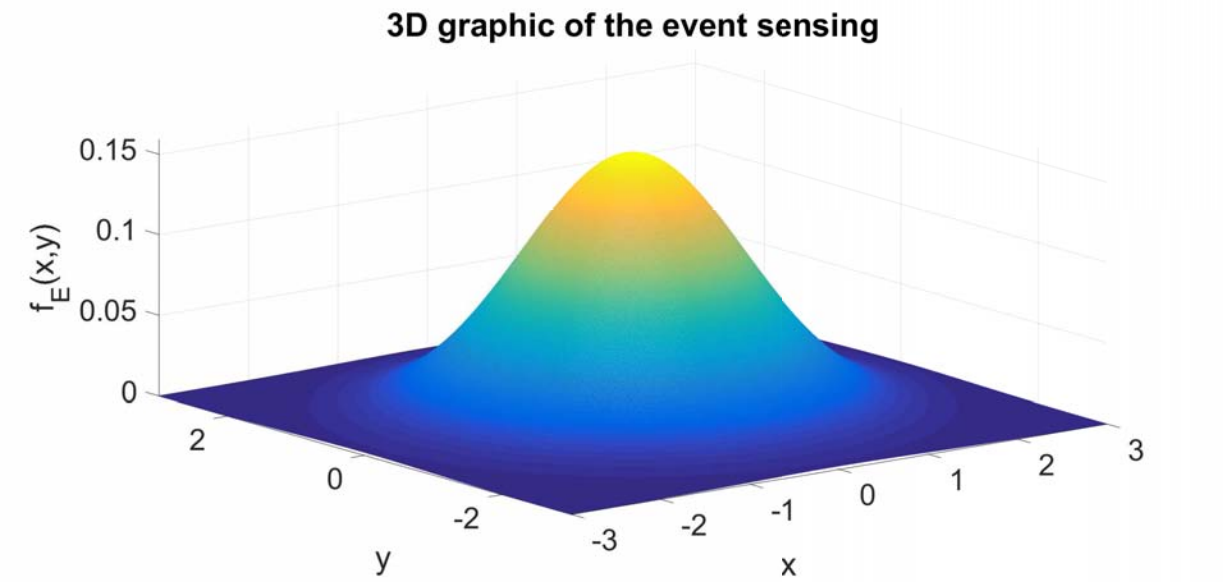


Figure 7.9.: $f_E(z)$ from (7.6) for a value of $\lambda_E = 1$: out of $R_E = 2$ the function is null.

7.6.2 Dispatch technique

The dispatch is the action of sending robots to a predetermined position after they have been selected during an event detection.
Firstly, we compute a target position for all robots belonging to a set called A: these locations are disposed circularly around the event. If, for instance, 6 robots are involved in the dispatch then targets ideally aim at forming an hexagon centred on the event.
Secondly, we design a trajectory to follow for the robots which have to move. These trajectories can be followed by an agent using a feedback control law as long as the communication with the leader robot, defined as A_0, persists.
Here, at the moment, we assume the hypothesis to manage coordinates of locations: since our robots are limited in sensing, actually, we need a localization strategy to obtain coordinates at least local, i.e. expressed with respect to a reference frame of a robot. This strategy will be explained later in section 8.3.2 regarding the algorithm design.

Target computation

In order to delimit a good region around the event we try to send robots close to it in a circular way. If we manage L_A robots for the dispatch then we try to form a regular polygon around the event.
This strategy is attained as follows:

1. consider $d_0 = dist(\mu_E, A_0)$, θ_E^0 and $E = E^0$, already computed and measured in the previous sections, where with the 0 superscript we intend that an object is seen from the reference frame attached to A_0;

2. for each robot $A_k, k \in [1, M]$ in $A \setminus A_0$ compute

$$z_k = d_0 * \exp\left(j\left(\frac{2\pi k}{L_A} - \theta_E^0\right)\right) + E_x + jE_y$$

   where j is the imaginary unit;

3. obtain the target coordinates A_k^{0*} w.r.t. the r.f. attached to A_0 taking

$$A_k^{0*} = (Re(z_k), Im(z_k))$$

Control law

Here we describe the algorithm used to compute the control law for the dispatch of the front-wheeled car: see table 7.1 to read the meaning of each variable.
Our aim is computing these control inputs: the velocity v of the vehicle (necessary for the dynamics of x, y, θ) and the angular velocity of the steer $\dot{\gamma} = \omega$.
The criterion we adopted to find such v and ω is based on a time minimization for the trajectory that a robot vehicle is going to accomplish. Say t the instant of time in which we have to compute new inputs to control our nonlinear system and say T the time needed to reach the target. Obviously, T depends on the current position, orientation,

steering angle and instant t. So, our goal is obtaining v and ω as stated in (7.7):

$$\boxed{\begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix} = \underset{x,y,\theta,\dot{x},\dot{y},\dot{\theta}}{\arg\min}\, T(x, y, \theta, \gamma, \dot{x}, \dot{y}, \dot{\theta}, \dot{\gamma}, t)} \tag{7.7}$$

Unfortunately, we have not found any proof yet that shows our control law minimizes cost functional in 7.7: this will be object of future studies. However, on the next lines we present a good attempt of a control law, followed by simulations in section 8.4 that prove its validity.

Parameter	**Meaning**
d	current distance between the vehicle and the target
(x, y)	current position of the vehicle
(x^*, y^*)	target position
s	sign of direction (forward = 1, backward = -1)
θ	planar orientation of the vehicle w.r.t. the world r.f.
θ^*	desired planar orientation for aligning the vehicle to the target
$v_{BACK/FOR}^{MAX}$	maximum backward/forward speeds
v	vehicle speed
k_v	proportional velocity constant
L	length of the car trolley
b	numeric constant: $b \in (0, 1)$
$(\dot{x}, \dot{y})$	velocity of the vehicle
w_{veh}	unit vector of the direction of the vehicle w.r.t. the world r.f.
w^*	unit vector of the direction taken from the vehicle to the target
p	scalar product factor between the two previous directions
p'	p corrected in function of a reverse movement of the vehicle
c	converging factor (important heuristic function)
ε	numerical constant: $0 < \varepsilon << 1$
$\dot{\theta}^*$	desired angular velocity w.r.t. to the world r.f.
$\dot{\theta}$	angular velocity w.r.t. to the world r.f.
k_o	proportional angular velocity constant
ω	input for the dynamics of the steering angle γ

Table 7.1.: Parameters involved in the control law for the front-wheeled car and their meaning.

Velocity control:

1. $d = \sqrt{(x^* - x)^2 + (y^* - y)^2}$
2. $s = \text{sign}\left(|\Delta\theta| - \frac{\pi}{2} - \left(1 + \frac{v_{BACK}^{MAX}}{v_{FOR}^{MAX}}\right)\frac{\pi}{2}\right)$, with $\Delta\theta = \theta^* - \theta$, $\Delta\theta \in (-\pi, \pi]$
3. $v = \begin{cases} sk_v d & \text{if} \quad d \geq L \\ (1-b)\sqrt{\dot{x}^2 + \dot{y}^2} & \text{if} \quad d < L \end{cases}$

Angular steering velocity control:

1. $w_{veh} = \begin{cases} \dfrac{\begin{bmatrix} 1 \\ \tan\theta \end{bmatrix}}{\left\| \begin{bmatrix} 1 \\ \tan\theta \end{bmatrix} \right\|} & |\theta| \neq \dfrac{\pi}{2} \\ \begin{bmatrix} 0 \\ \pm 1 \end{bmatrix} & \theta = \pm\dfrac{\pi}{2} \end{cases}$

2. $w^* = \dfrac{1}{d} \begin{bmatrix} x^* - x \\ y^* - y \end{bmatrix}$

3. $p = \langle w_{veh}, w^* \rangle$

4. $p' = \begin{cases} 0 & \text{if } p < 0 \;\&\&\; s = -1 \\ p & \text{otherwise} \end{cases}$

5. $c = \begin{cases} \dfrac{1}{\varepsilon} & \text{if } p' = 0 \\ 4 & \text{if } |p'| > \dfrac{1}{2} \\ \dfrac{1}{|p'^2 - |p'||} & \text{otherwise} \end{cases}$

6. $\dot{\theta}^* = \begin{cases} \dfrac{c}{d^2} \left(\dot{x}(y^* - y) - \dot{y}(x^* - x) \right) & \text{if } d \geq bL \\ 0 & \text{otherwise} \end{cases}$

7. $\omega = k_o \left(\dot{\theta}^* - \dot{\theta} \right)$

For other details (like heuristics, proofs and choices) of this control law, we refer to appendix A.

7.7 Summary about models

So far, we discussed about many model used in this thesis, as scenarios, robots, measurements, coverage policy adopted, environmental interactions, dynamics and event detection and dispatch.
The main aspects one should keep in mind are the following:

- we use closed random-preselected scenarios;
- robots that accomplish coverage are generated by one or more base station(s);
- robots have a sensing radius r_v to communicate each other, they can take measurements of the bearing angles θ_{ij}^k and the intensity of the event function $f_E(x, y)$ (by which the distance between an agent and an event can be computed);
- robots are provided by touching sensors used to manage environmental interactions;

- hexagonal packing is the coverage policy adopted: to perform it, Simplicial Complex Theory is needed;
- coverage is attained using a robotic network; therefore, Graph Theory is needed;
- events are modelled by an event function with a Gaussian shape for the intensity and appear in a point of the plain after the coverage is totally accomplished;
- the robot dynamics adopted is the front-wheeled car: by the Nonlinear Control Theory we can state that this model can reach any point of the plane using a suitable control law in order to perform the dispatch around an event.

8

Algorithm and illustrative ad hoc simulations

Here we present a coverage algorithm based on [2] and show some ad-hoc simulations to better explain the results.

8.1 Coverage algorithm

Simulations are executed in an environment defined by the user in function *Scenario*: it gives the vertices of each segment that represents a wall of the room or an obstacle inside the room. The algorithm starts by initializing the parameters and by deploying the base station, that is the first robot (1) in a specific point of the environment and it has been initialized as a fence robot. Then we start our deployment cycle: by pushing the robots in the scenario, with the attention of remove the useless robot first. For this purpose we run the function called $removeUseless$ if the useless subset is not empty. Secondly we use $computeRipsComplex$ function which returns the set R_{r_v} that is the union between $R_{r_v}0, R_{r_v}1$ and $R_{r_v}2$ subsets which contains respectively all the 0-simpleces, 1-simpleces and 2-simpleces of the graph structure. Then we update frontier set F (where $F = F0 \cup F1$) and obstacle set O (where $O = O1 \cup O0$). After that we check for the exit condition of the cycle: if the Fence subset F is empty, we break the cycle otherwise we compute the useless subset through the function $findRedundant$ and we deploy a new robot in the environment thanks to $deployNewRobot$.
Once out of the loop, we remove all the unnecessary robots remained: during the main cycle only one robot per cycle is removed, because if one removes a robot then it couldn't be true that other robots identified as useless still are unnecessary.
The tabular flowchart[1] below gives an idea of the main actions executed during coverage task.

[1] whose subroutines are presented in sections 8.1.1-8.1.4.

Algorithm 1 **Main (coverage part)**

Deploy basestation; $n \leftarrow 1$
while $F \neq \emptyset$
  if *useless* $\neq \emptyset$
    removeUseless (*useless*); $n \leftarrow n - 1$
  end if
  R_{r_v} = computRipsComplex(G)
  [F, O] = fenceSubcomplex(G, R_{r_v})
  if $F = \emptyset$
    break
  end if
  useless = findRedundat(R_{r_v})
  G = deployNewRobot(F, R_{r_v}); $n \leftarrow n + 1$
end while
while useless $\neq \emptyset$
  R_{r_v} = computeRipsComplex(G)
  removeUseless (*useless*); $n \leftarrow n - 1$
end while

8.1.1 Compute the Vietoris-Rips complex

The function named *computeRipsComplex* has to construct the set R_{r_v} composed by the 0-simplex subset $R_{r_v}0$ that contains all the robots in the environment, the 1-simplex subset $R_{r_v}1$ that contains all the couples of neighbour robots and the 2-simplex subset $R_{r_v}2$ which contains all the 3-tuple of neighbour robots. The input of the function is the graph structure G but the function just need the information about the neighbours of each robot to simulate a distributed behaviour.

Algorithm 2 [R_{r_v}] =**computeRipsComplex**(G)
Input: Local graph structure G
Output: R_{r_v}

$R_{r_v} = \emptyset$
for robot = 1,...,n **do**
  $R_{r_v}0 \leftarrow R_{r_v}0 \cup \{i\}$
  for robot j $\in \mathcal{N}_i$ **do**
    $R_{r_v}1 \leftarrow R_{r_v}1 \cup \{i, j\}$
    for robot $k \in \mathcal{N}_i$ **do**
      if $j \neq k$ and $j \in \mathcal{N}_k$ **then**
        $R_{r_v}2 \leftarrow R_{r_v}2 \cup \{i, j, k\}$
      end if
    end for
  end for
end for

8.1.2 Compute the Fence and the Obstacle subsets

This function computes the fence F and the obstacle O subcomplexes given as input the subsets of R_{r_v} and the graph structure G. We say that an edge is a fence edge if it has not completed the 2-simplices in both its side as shown in Fig. 8.1; it is an obstacle edge if it has not completed the 2-simplices in both its sides but it can't due to the contact with an obstacle, the collision with another agent or because there is not enough space for a new robot.

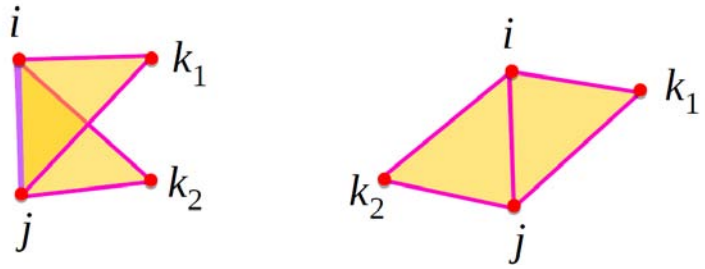


Figure 8.1.: In the left figure the edge $\{ij\}$ has all the 2-simplices lying on the same side so it is marked as a fence edge,; in the right one both side of $\{ij\}$ are covered with at least a 2-simplex so it is marked as a normal edge.

In this algorithm at first the exceptions have been computed thanks to the function *computeException* that finds all the exceptional edge that can not be expanded. Thus, for each edge that is not an exception, we search its uncovered side, so we can have two situations: there is not uncovered side or there is at least one uncovered side. In the first situation we understand $\{ij\}$ as a normal edge[2], in the second situation we call the function *isObstacle* that tells us which of the three $\{\{i\},\{j\},\{ij\}\}$ has to be included in the obstacle set O or if we have to insert them in the fence set F.

Algorithm 3 $[F,O]$=**fenceSubcomplex**(R_{r_v})
Input: G, R_{r_v}
Output: F, O

$F \leftarrow \emptyset$
$O \leftarrow \emptyset$
E = computeException(R_{r_v})
for $\{i,j\} \in R_{r_v} \setminus E$ **do**
  $unCov_{ij}$ = uncovered(R_{r_v}, i, j)
  if $unCov_{ij} \neq \emptyset$ **then**
    if isObstacle(G, i, j)
      $O \leftarrow O \cup \{\{i\},\{j\},\{ij\}\}$
    else
      $F \leftarrow \{\{i\},\{j\},\{ij\}\}$
  end if
end for

[2] as shown in the right part of Fig. 8.1

8.1.3 Find the redundant robots

With this function we find redundant robots that can be removed without losing any coverage capability. We say that a robot is useless when it lays inside a 2-simplex or it is the vertex in a degenerate 2-simplex; both the situation are shown in Fig. 8.2. We say those kinds of robot are useless because they increase the complexity of the Vietoris-Rips complex without providing a significant growth of coverage capability.

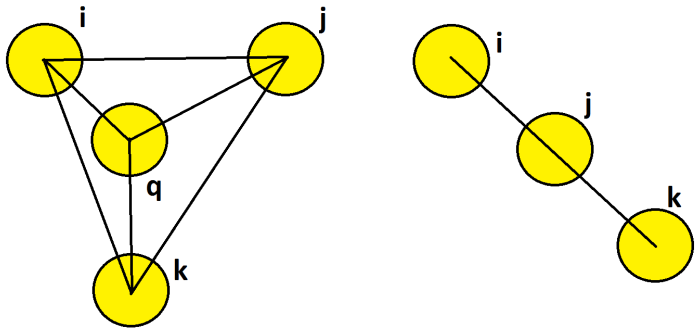


Figure 8.2.: In the left figure robot q can be removed because it lays inside a 2-simplex; on the right robot j is the vertex of a degenerate 2-simplex because i and k are neighbours.

We manage to detect all those situations in a complete distributed way: in the first case, using the collection of the 2-simplices $R_{r_v}2$ we search for a common neighbour called q of the trio in $R_{r_v}2$ that measures the angle $\beta = \theta^q_{ij} + \theta^q_{jk} + \theta^q_{ki}$. If $\beta = 2\pi$ we understand that q lays inside the trio $\{i, j, k\}$ and we mark it as redundant. The other situation can be easily detected when in a trio $\{i, j, k\}$ of $R_{r_v}2$: one of the robot measures an angle between the two others of π; if this occurs the robot is marked as redundant. This situation usually happens when the three robots are in contact with the same wall.
The only information that robots must exchange to find those redundant robot is local, in particular, a robot must know the neighbourhood of his neighbours and the angles that they measure. In every iteration, If there is a useless robot , we remove it with a probability p_{rim}.

8.1.4 Deployment of the new robot

We have just discriminated simplices belonging to the fence or to the obstacle subset. Now we have to add a new robot in order to expand the network and increase the coverage capability.

Algorithm 4 $[G]$ =**deployNewRobot**(G, F, R_{r_v})
Input: G, F, R_{r_v}
Output: G

shortestPath = findPath(G,F);
$i \leftarrow$ last node of shortestPath;
if $\exists\ j : \{i, j\} \in F1$
  try to expand from $\{i, j\}$
else $(i \in F0)$
  try to expand from $\{i\}$
end if
if can expand
  push robots through path in the network G;
  activate contacts of robots if needed;
  add new robot close to the base station; $n \leftarrow n + 1$
end if

Expansion policy

Our expansion strategy is based on searching the shortest path[3] between the base station and the fence subcomplex; then the new robot is deployed by pushing all the robots through the path and creating a new robot in the base station as shown in Fig. 8.3.

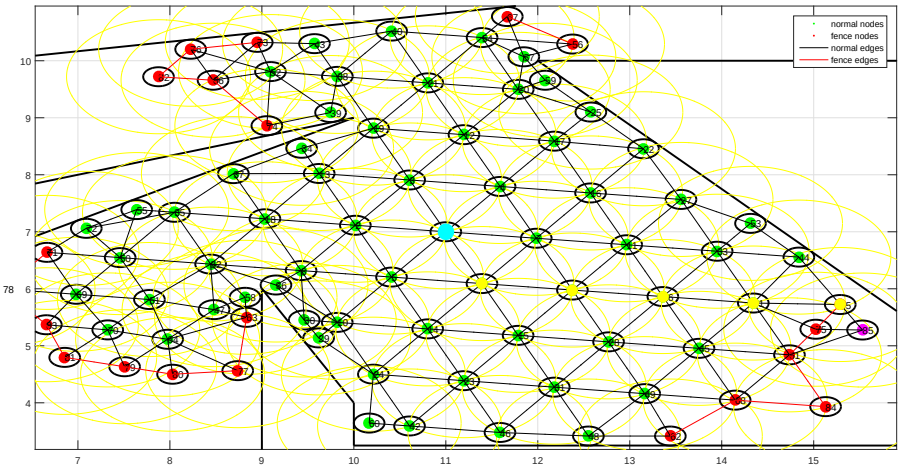


Figure 8.3.: In cyan blue the base station and in yellow the path for the pushing; finally in magenta the expanded new position.

In order to optimize the algorithm we decide not to consider the robots that are close to the base station and far from the fence subcomplex. In particular, let $f_{distance}$ be the length of the shortest path from the base station to an element in the fence subcomplex, we say that an edge is a fictitious exception if its nodes are both at a distance $\delta \leq f_{distance} - 2$. In this way we avoid the computation test of the fence subcomplex for

[3] a distributed research algorithm method for the shortest path can be found in [20]

edges that are far from the expanding frontier.
To avoid the possibility of losing coverage capability due to failure of single robots, we periodically compute the fence subcomplex in the whole graph in order to deploy a new robot if it has been broken down. In this way we do not let uncovered areas after a robot loss. Let $p_{failure}$ be the probability of a random robot in the network fails during each iteration.

Expansion from $\{i, j\} \in F1$

Expanding from a fence edge $\{i, j\}$ firstly requires a computation of the uncovered direction $\sigma \in \{+1, -1\}$. Now we need to find, in the local coordinates of i and j, the location for the new robot position. Figure 8.4 illustrates the uncovered side of 1-simplex $\{i, j\}$ in i's local coordinate. Our strategy for choosing the position to deploy next robot is to try and achieve a hexagonal packing (which is the most optimal packing on an obstacle-free plane) of robots as much as possible, only to be interrupted by the presence of obstacles or control's error. This essentially boils down to sending robots at an angle of 60Âř with respect to $\overline{ij}$ into the free region. Using *deploymentAngle* function we determine the closest other fence 1-simplices attached to i and j (e.g., $\{i, k\}$. If there is no other fence 1-simplex attached to i, we set $\theta^i_{new} + \sigma^i_j \frac{\pi}{3}$ the 60° angle for deployment in a hexagonal packing. Otherwise we set the angle to the minimum between the one for hexagonal packaging $\frac{\pi}{3}$ and the the one that bisects $\theta^i_{jk_i}$. Likewise for θ^j_{new}.

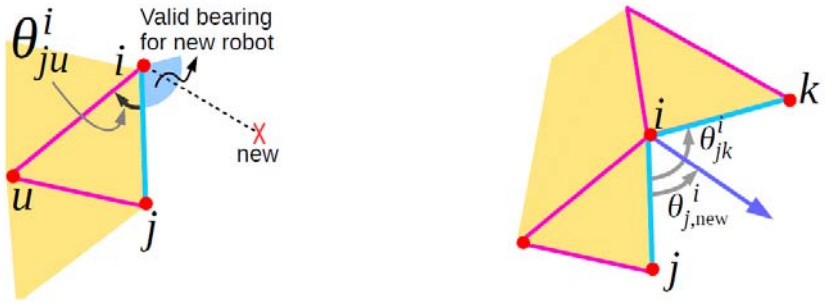


Figure 8.4.: On the left: the free side of $\{i, j\}$ where $\text{sign}(\theta^i_{j,new}) = \sigma$. On the right: the bearing to the new location, $\theta^i_{j,new} = \min\{(\frac{\pi}{3}, \frac{\theta^i_{j,k}}{2})\}$, in i's local coord.

Expansion from $\{i\} \in F0$

If i is not attached to a frontier 1-simplex (e.g., i is a frontier robot in a narrow passage with a single file of robots), then we simply choose a random angular direction away $\mathsf{d}_{a,0}$ from i for the deployment of the new robot k. Then a test drive from i is implemented: k tries to leave away until its visibility from i holds. If this movement is not allowed due to lacking of space then another leaving direction d' is tried with the following law: $\mathsf{d}' = \mathsf{d} + \frac{2\pi}{N_a}$, $N_a \geq 4$, see Fig. 8.5. This procedure is repeated N_a times, so the value of n-th computed direction is $\mathsf{d}_{a,n} = \mathsf{d} + n\frac{2\pi}{N_a}$ Thus we can attempt an expansion in every direction with a angular sensitivity of $\frac{2\pi}{N_a}$.

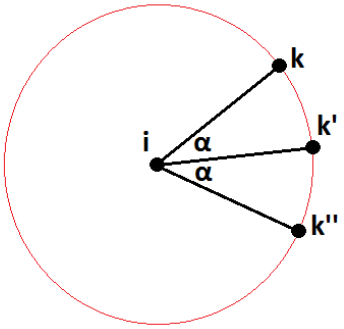


Figure 8.5.: Expansion strategy for $i \in F0$: $\alpha = \frac{2\pi}{N_a}$.

Pushing new robots from the base station

Our strategy to attain deployment is based on moving the frontier robot i towards the new location and nodes between base station and i one step-ahead through the path. Finally, a new robot is generated upon base station location by the latter. This procedure is shown in Fig. 8.6.

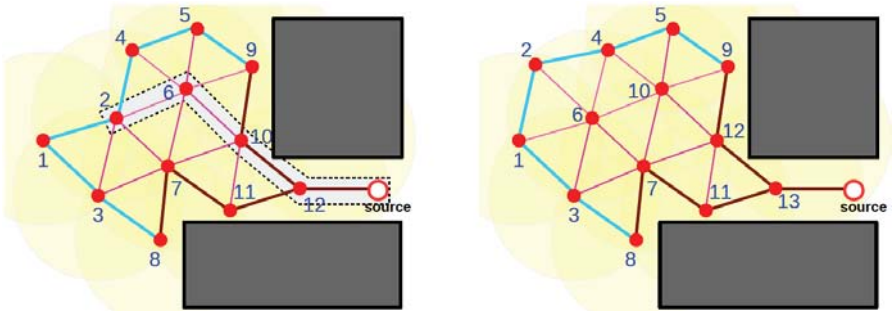


Figure 8.6.: Robot 2,6,10,12 are pushed along the path and robot 13 is generated upon the base station.

8.1.5 Multiple base station coverage protocol

In the case multiple base stations are placed in the environment we adopt the following coverage protocol. Sources can't share information because there is not an effective connection between them[4]; thus, each of them realizes coverage independently, which, in this phase, is faster. If between the base stations in the environment there are not narrow passages, at a certain point the covered areas will overlap and there will be a possible connection between the base stations. In this situation sources can share information and the following protocol will be adopted: the path through which the new deployment takes place will be the one generated thanks to the closest[5] source. In this way fence nodes that are closer to one of the sources will have expansion priority.

[4]There is not a path between base stations through deployed robot
[5]In terms of number of robots between base station and frontier

8.2 Simulations in absence of events

Now we analyse the performances of the algorithm presented in section 8.1. The experiment has been performed using MATLAB as numeric platform. If not specified: $p_{failure} = 0$, $p_{rim} = 0.8$.
In diagrams we are going to show we do not choose any unit of measurement to intend that simulations could be thought with any scale of length. However, chosen a unit of measurement for an axis, the other one has the same unit.

8.2.1 Rectangular scenario with theoretical analysis

We start by presenting a fundamental result: as Fig. 8.7 shows, hexagonal packing in normal conditions is achieved all over the space environment that is not influenced by border effects. These are inevitable effects which appear while deployment is attaining close to a wall of the scenario and prevent an ideal hexagonal packing not only in these zones but even in areas where coverage is going to be accomplished. As one can see, Voronoi partitions in Fig. 8.8 arrange hexagons in the middle of the room only, since the base station has been placed there.

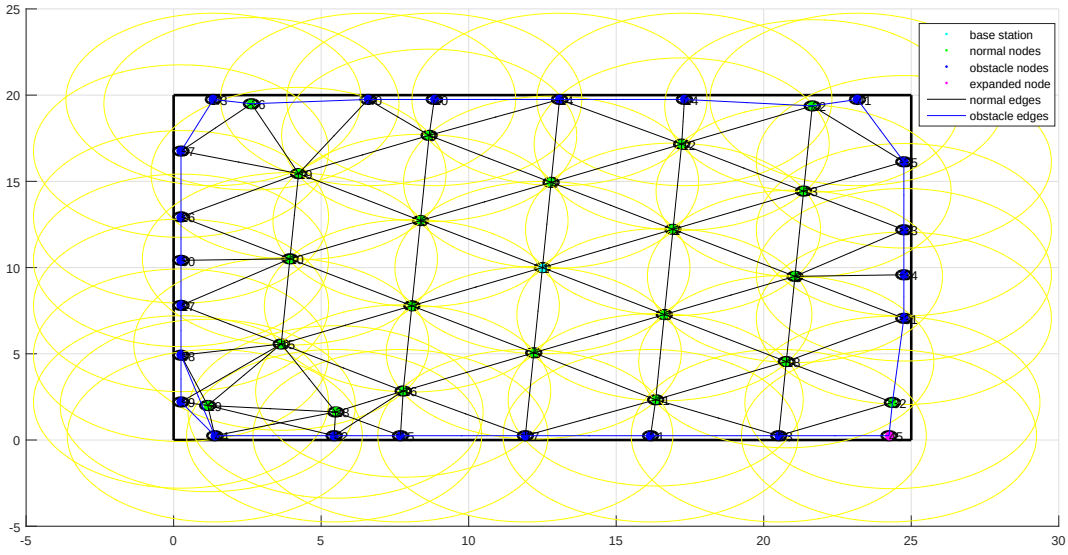


Figure 8.7.: Simulation in a rectangular environment without noise nor robot failures. Chosen scenario is a rectangle $a \times b$ with $a = 25$, $b = 20$. Sensing radius used is $r_v = 5$. 45 robots have been deployed.

In this context a theoretical comparison between bounds obtained in section 5.2 and deployment performed by our algorithm can be made. Here we get $N_{MC} = 21$, $\tilde{N}_{MC} = 23$ as estimates of lowest bound. The real lowest bound of the number of robot to deploy is $N_{MC,real} = 20$ (obtained by simulations about lowest bound as figures in section 5.2 show) and the number of the robots actually deployed by our algorithm is $N_{dep} = 45$; see Fig. 8.9 for more details. One should notice that $N_{dep} > 2N_{MC,real}$: this is due to the

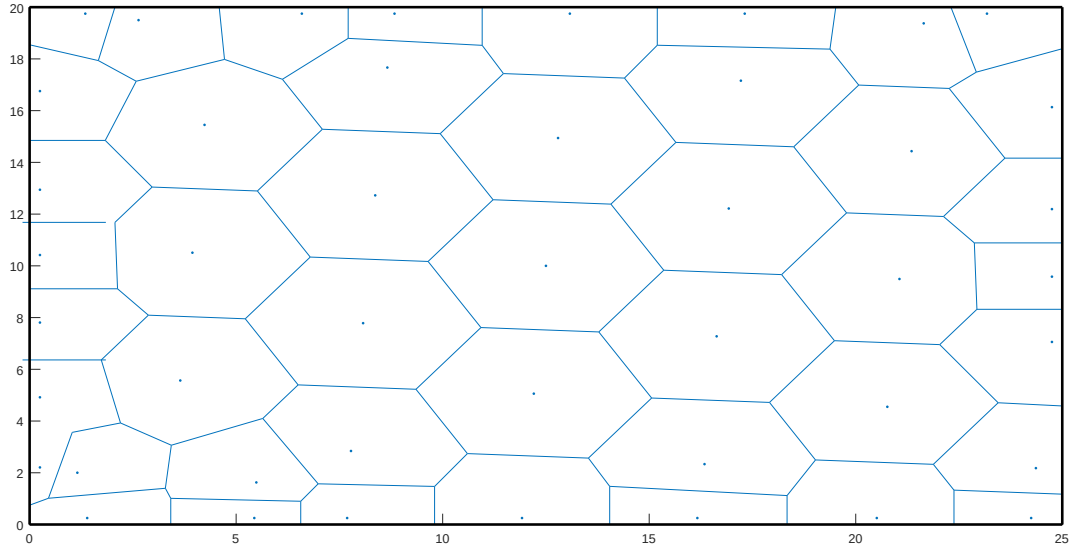


Figure 8.8.: Corresponding Voronoi partition of simulation in Fig. 8.7.

border effects which should be taken into account. Indeed our algorithm needs to expand the frontier of the sensing network as much it can since scenario is never known a priori; thus, another comparison can be made: since 23 robots in Fig. 8.7 are in touch with walls, then, we can consider them as an attempt for "identifying" scenario's shape by our algorithm. Therefore, we can roughly state the task of performing hexagonal packing is actually carried out by the other 22 robots inside the boundary of the rectangle, which is a result very close to the lowest estimated theoretical bound N_{MC} of agents to deploy.

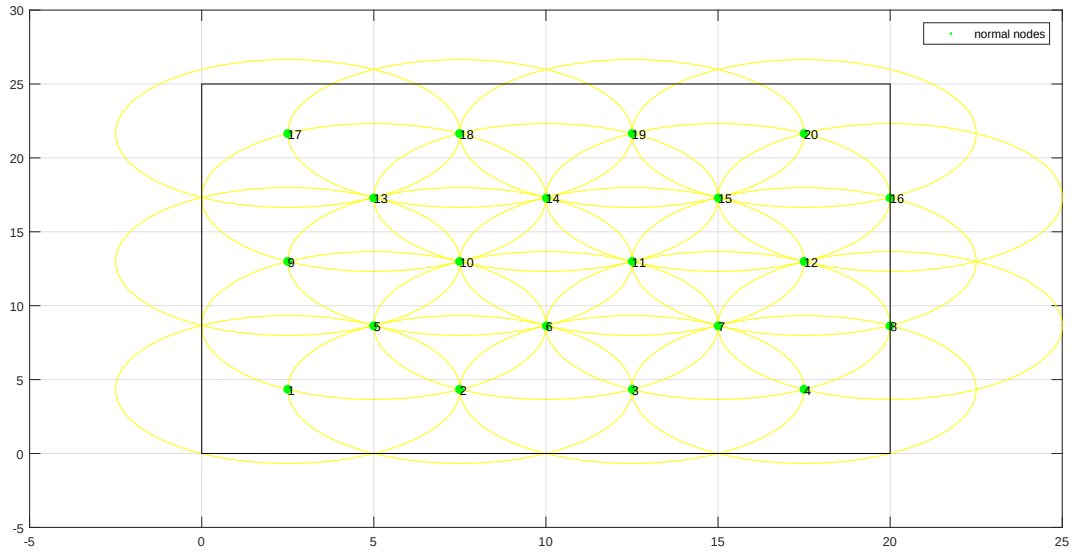


Figure 8.9.: Comparison with theoretical bound of simulation in Fig. 8.7. Here $N_{MC,real} = 20$, $N_{MC} = 21$, $\tilde{N}_{MC} = 23$.

By introducing a zero mean Gaussian white noise with variance $r_p^2 : 0 < r_p \ll r_v$ while deployments[6] of new robots are carrying out we notice that hexagonal structures are distorted. The higher is r_p with respect to r_v the less hexagonal packing on Voronoi partitions is reached. However, the robustness of the network connectivity is assured and coverage can be usually performed in a standard finite time if r_p is sufficiently small.

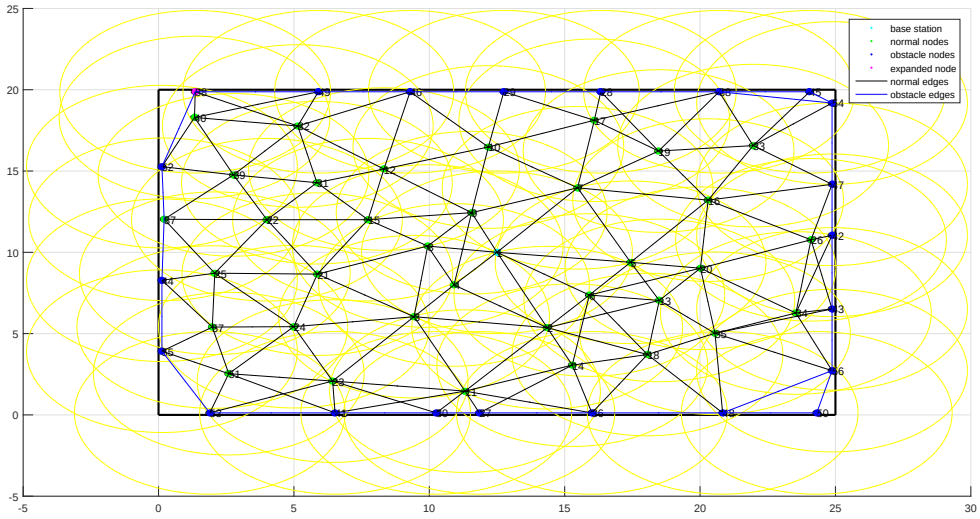


Figure 8.10.: Simulation in a rectangular scenario with noisy placement. $r_v = 5$, $\phi = 0.5$, $r_p = 0.05$.

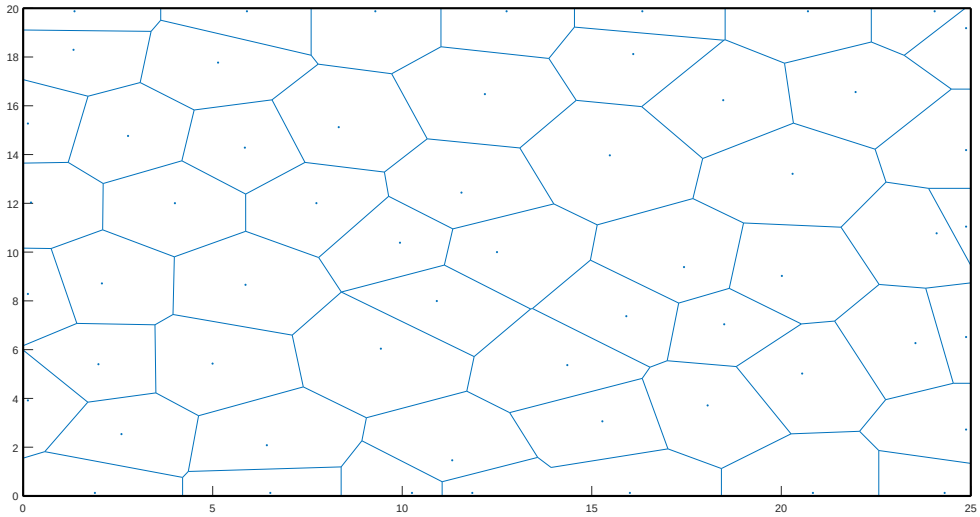


Figure 8.11.: Corresponding Voronoi partition of simulation in Fig. 8.10.

[6] In a noisy framework deployment coordinates are computing with this law: $newPlace = targetPlace + x$ where $x \sim \mathcal{N}(0, I_2 r_p^2)$. Note that there is the possibility that new robots can lose connections with the sensor network created since that moment.

In a real background it can happen that a robot fails the deployment or stops working. Thus, its node in the network disappears and it can occurs a disconnection of the whole graph. An important simulation, provided in Fig. 8.12, proves the robustness to robot failures. As one can see in Fig. 8.13, Voronoi partition tells us that a perfect hexagonal packing cannot be guaranteed in this case.

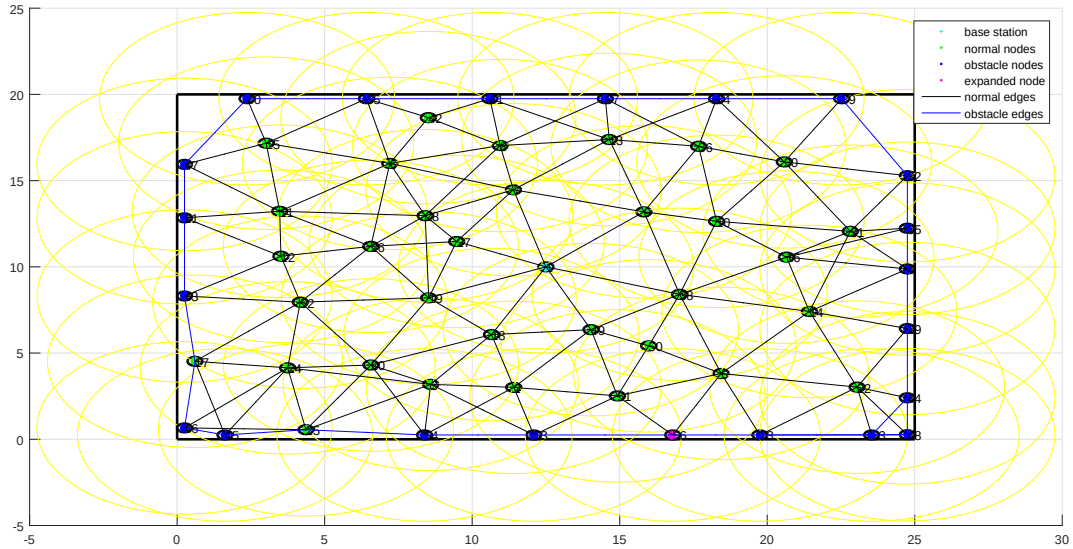


Figure 8.12.: Simulation in a rectangular noise-free environment with possible robot failures. $r_v = 5$, $\phi = 0.5$, $p_{failure} = \frac{1}{3}$

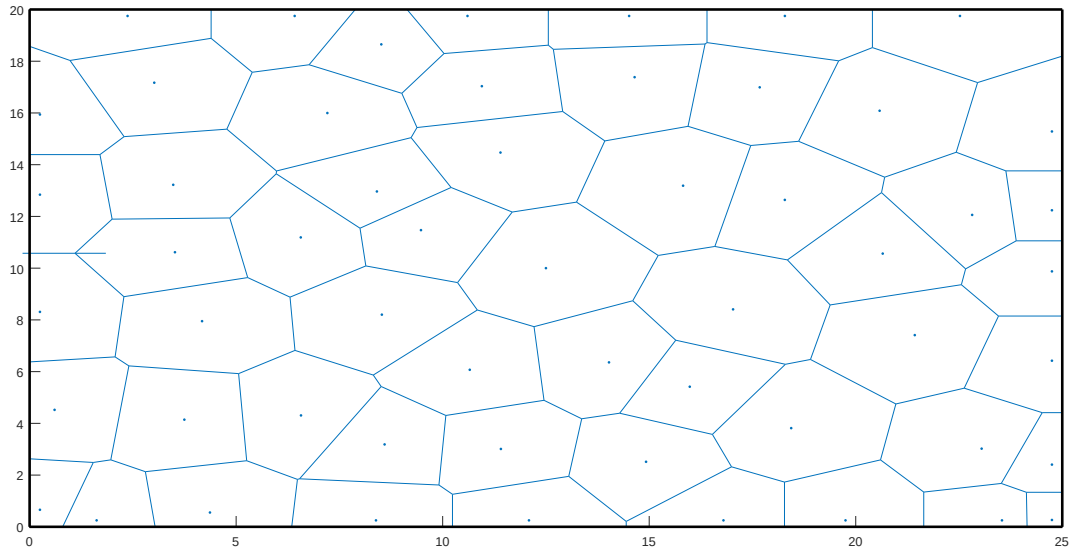


Figure 8.13.: Corresponding Voronoi partition of simulation in Fig. 8.12.

8.3 Event detection and dispatch algorithm

The event detection and dispatch operations start when all coverage is concluded. Two operation are needed to know how many robots are involved in the dispatch and where they are located before starting it, respectively: **detection** and **localization**. See the next tabular scheme to understand how our entire algorithm works.
Detection function *detect* selects some robots of the graph G close to the event and groups them in a set call A. If the cardinality of this set is at least 2 then the localization procedure starts with *localize* in order to get the poses and the target positions of all the agents involved in the dispatch. After that, the dispatch begins: the function $controlVehicles$ sends robots towards they targets and, finally, the connections in graph G are reconstructed with *reconstruct*.

Algorithm 5 **Main (global)**

Main (coverage part);
A = detect(G);
if cardinality(A) > 1
  $[A^0, Targets]$ = localize(A);
  G = controlVehicles(G,A^0,$Targets$);
  G = reconstruct(G);
end if

8.3.1 Detection

The detection is the common action of the robots that consists in pinpointing the event and creating a good localization strategy that allows to perform the dispatch.
More in details, the detection procedure is the following:

1. consider all robots A_i, $i \in [1, N]$ already deployed on the plane: all of them can be potentially involved in the detection, so we say that set A contains the detected robots and, at the moment, $A = \{A_i | i \in [1, N]\}$;

2. all robots compute the distance from the event knowing the function $f_E(x, y)$: if there exist robots A_j, for some $j \in [1, N]$, whose event function value is null then $A \leftarrow A \setminus \left\{A_j | j \in [1, N], f_E^{A_j}(x, y) = 0\right\}$;

3. find robot $A_{\bar{i}}$, $\bar{i} \in [1, N]$, that maximizes $f_E(x, y)$ i.e. minimizes the distance from the event: call it $A_0 = A_{\bar{i}}$ and say $d_0 = dist(\mu_E, A_0)$;

4. exclude the robots A_l from A which do not respect the inequality $d_0 + kd_l > r_v$ with $0 < k < 1$ where $d_l = dist(\mu_E, A_l)$ and k is a constant decided by the user[7]: in the end we get $A \leftarrow A \setminus \{A_l | l \in [1, N], d_0 + kd_l > r_v\}$;

[7] We set $k = \frac{1}{2}$. It's important to keep the value of $k < 1$, otherwise, in the reality, during the dispatch phase we may have communications problems between robots, loosing the distributed behaviour. We will recall this issue later, in the conclusions.

5. redefine the rest of the indexes[8] for the elements in A from 0 to M, $0 \leq M \leq N$, getting robots $A_i, i \in [0, M]$ ready for the next point (note that M could be fixed by the user, if, for example, one prefer not to involve too many robot in the dispatch and discard some of them);

6. since A_0 is the closest robot to the event then it becomes the "leader" robot: in other terms A_0 will be the agent from which the dispatch of the other agents of the current set A starts; thus, it is necessary to translate the coordinates of the other robots in A with respect to the reference frame attached to A_0 through a localization procedure.

8.3.2 Localization

This is a crucial procedure that allows as to reconstruct all the poses of the robots in the set A with respect to (w.r.t.) the reference frame (r.f.) attached to agent A_0 and to compute the target points to reach during the dispatch. The reason why we have to perform this method before dispatch is the lack of information about the location (x, y) of the robots: we recall that our robot are not provided by maps and only use bearing angle and event sensing measurements to move in the environment.

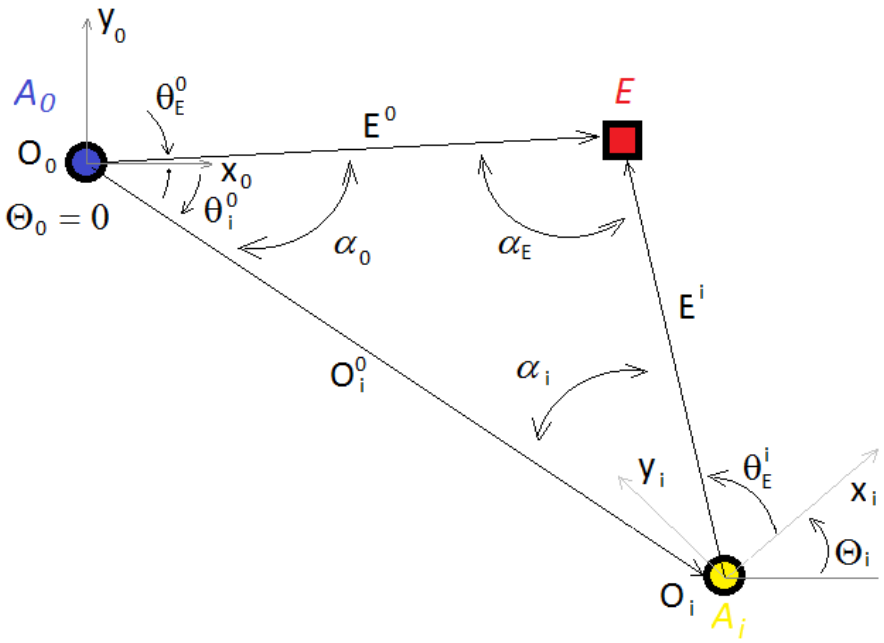


Figure 8.14.: Localization of the event E and the robot A_i. The poses are expressed with respect the reference frame attached to the closest robot A_0 to the event. Just in this case, for the sake of simplicity, the picture shows an angle $\Theta_0 = 0$.

Note that we use subscripts to indicate at which robot (or event) a geometrical object belongs and the superscript to recall the reference frame from where a geometrical object is seen and indicate local properties.
Before starting to read the procedure, we recall that a generic robot A_j has a status which

[8] keep index 0 for the robot which minimize $dist(\mu_E, A_i)$

is described by the four variables $A_j = (x_j, y_j, \theta_j, \gamma_j)$ we discussed in section 7.5.2. Since variable γ here is not relevant, we let the status reduce into $A_j = (X_j, Y_j, \Theta_j)$; moreover, here we write in capital letter and without superscript what is expressed w.r.t. the world r.f.. So, **just in this framework**, we use a state of dimension 3 (e.g. $A_j^0 = (X_j^0, Y_j^0, \Theta_j^0)$ for local states), capital letters for variables expressed in a global context and lowercase letters to indicate axes.
See Fig. 8.14 for helping the reading of the procedure that follows:

1. consider the r.f. $\{O_0, x_0, y_0\}$ located[9] in $O_0 = (X_0, Y_0)$, tilted with an angle of Θ_0 and attached to the robot A_0 s.t. $A_0 = (X_0, Y_0, \Theta_0)$: set the **local** status $A_0^0 = (X_0^0, Y_0^0, \Theta_0^0) = (0, 0, 0)$;

2. by the knowledge of the distance between O_0 and μ_E (acquired using the inverse of the event function) and the measurement of the bearing angle θ_E^0, using simple trigonometric formulas, we can construct a vector E^0 and obtain the position $E = E^0$ of the event w.r.t. the r.f. attached to A_0;

3. repeat step 2 for the robot A_i in order to get the angle θ_E^i and the vector E^i;

4. using the bearing angles θ_E^0 and θ_E^i measure other two angles in absolute value $|\alpha_0|$ and $|\alpha_i|$; then, compute $|\alpha_E| = \pi - |\alpha_0| - |\alpha_i|$;

5. apply Carnot's theorem for computing the length of the vector O_i^0

$$\left\|O_i^0\right\| = \sqrt{\|E^0\|^2 + \|E^i\|^2 - 2\,\|E^0\|\,\|E^i\| \cos|\alpha_E|}$$

   and measure the bearing angle θ_i^0 to obtain the vector $O_i^0 = \begin{bmatrix} X_i^0 \\ Y_i^0 \end{bmatrix} = \|O_i^0\| \begin{bmatrix} \cos(\theta_i^0) \\ \sin(\theta_i^0) \end{bmatrix}$ that expresses the position of O_i w.r.t. the r.f. of A_0;

6. compute the angle Θ_i^0 of the i-th frame w.r.t. the r.f. attached to A_0, paying attention to understand the correct sign[10] of α_E:

$$\Theta_i^0 = \theta_E^0 + \alpha_E - \theta_E^i \tag{8.1}$$

7. gather the state $A_i^0 = (X_i^0, Y_i^0, \Theta_i^0)$ and repeat points 2-7 for all the other robots belonging to set A;

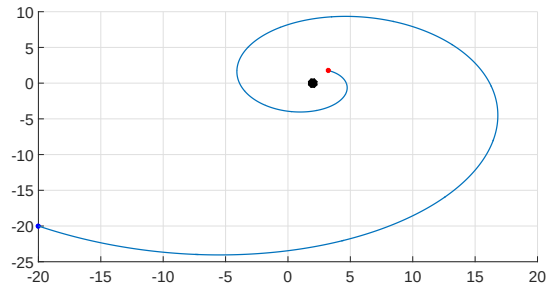
8. compute the targets for each robot A_i^0 as explained in 7.6.2.

8.4 Simulations on the control law for dispatch

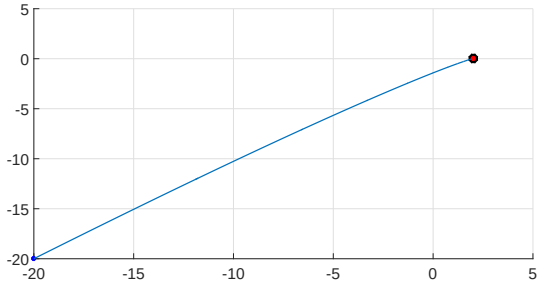
Now we would like to show some important results about the control law employed for the dispatch. This section offers some pictures that illustrate how one robot can move from a starting point (∘) to a final one (∘) trying to converge to a preselected target point (∘).

[9] We use coordinates X, Y to indicate a position expressed w.r.t a world r.f.
[10] in (8.1) α_E is taken from the vector E^i to the vector E^0 in counter-clockwise direction

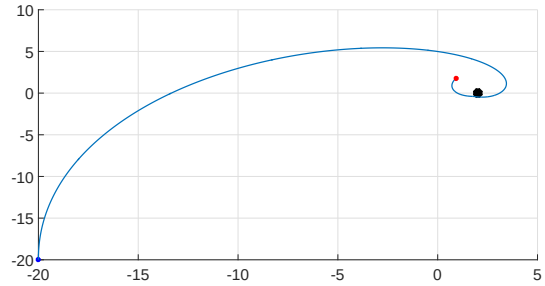
On the first set of images depicted in Fig. 8.15 one can understand the reason why we need a control law with some heuristics like the reverse movement and the converging factor: the trajectories of the vehicle are often too long and expensive (in matter of time cost) and the convergence is sometimes not guaranteed. The plots depict a control performed with a "rough" law reported in appendix A.2.



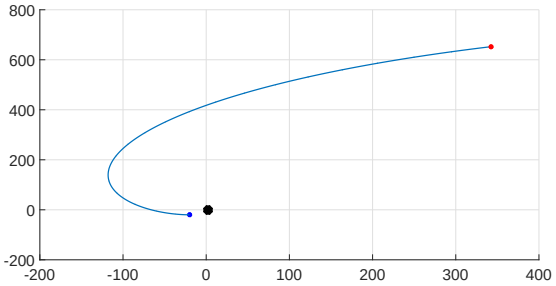
(a) $\theta(0) = -30°$



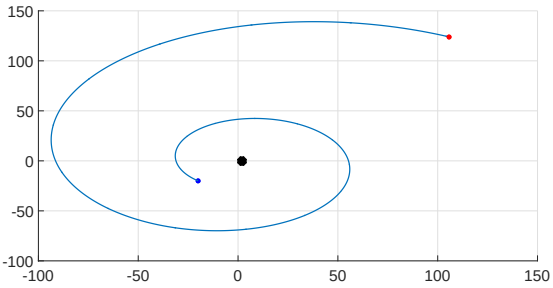
(b) $\theta(0) = 45°$



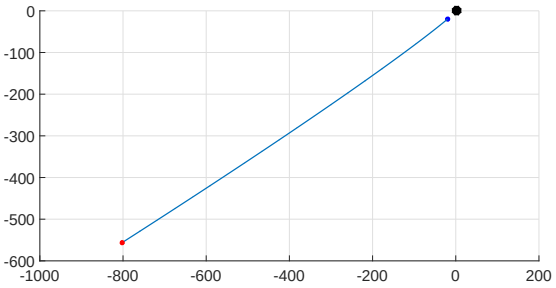
(c) $\theta(0) = 90°$



(d) $\theta(0) = 180°$



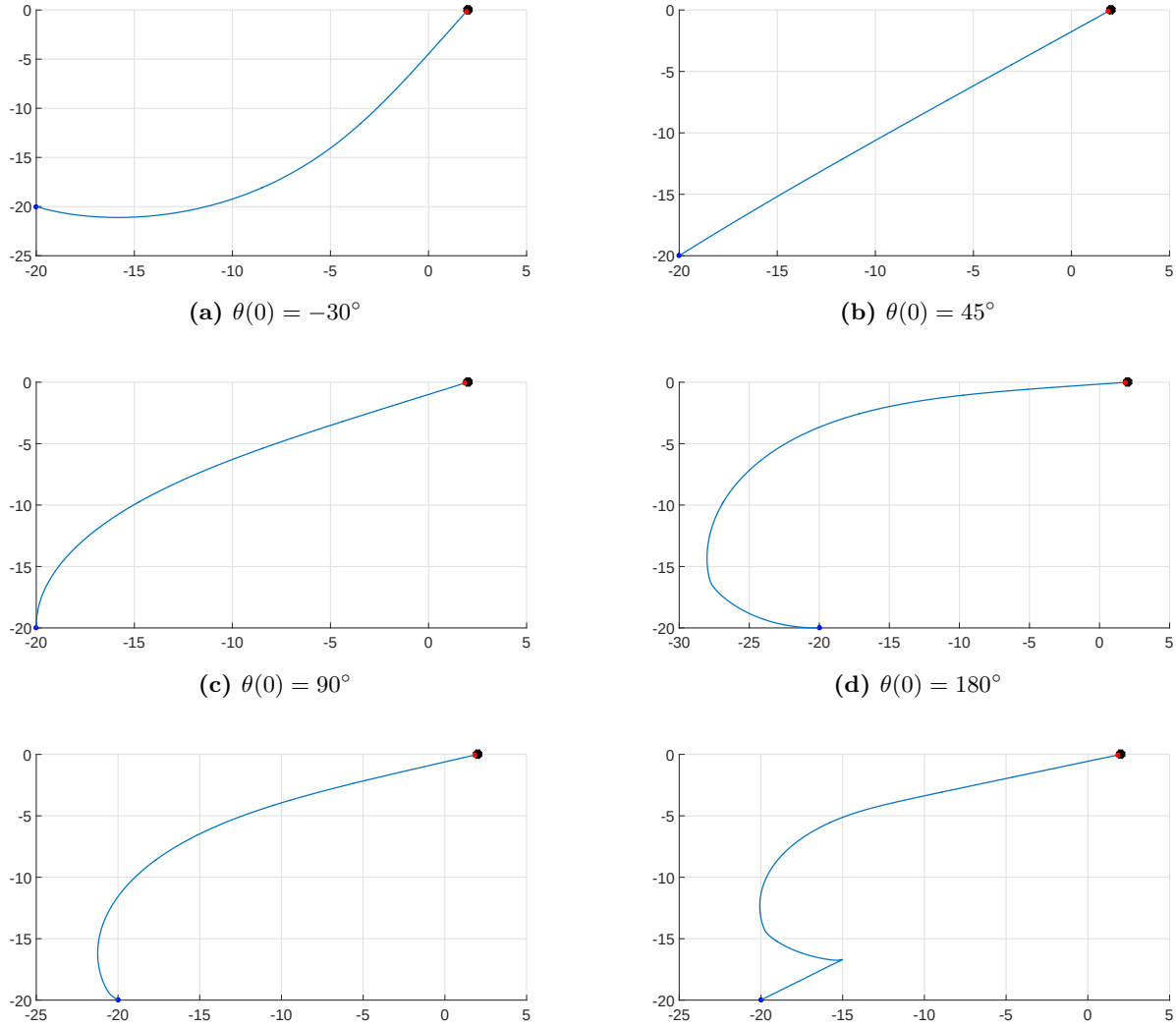
(e) $\theta(0) = 140°$



(f) $\theta(0) = -140°$

Figure 8.15.: Trajectories of a vehicle in functions of different initial angles $\theta(0)$. Here the control law is "rough": in this attempt no reverse movements have been implemented and converging coefficient (see c in section 7.6.2) has not been used. One can immediately recognize that the convergence highly depend on $\theta(0)$ and sometimes is not ensured, like in cases (8.15d)-(8.15f).

On the second set in Fig. 8.16, instead, we show the improvements that heuristic provides: the implementation of the reverse and the converging factor helps the vehicle to find a better way to reach the targets.



(a) $\theta(0) = -30°$ **(b)** $\theta(0) = 45°$

(c) $\theta(0) = 90°$ **(d)** $\theta(0) = 180°$

(e) $\theta(0) = 140°$ **(f)** $\theta(0) = -140°$

Figure 8.16.: Trajectories of a vehicle in functions of different initial angles $\theta(0)$. Here the control law is improved with some heuristics (see, for instance, c function in the control law). Note that reverse movements are actuated in Fig. 8.16f.

Part IV

Simulations

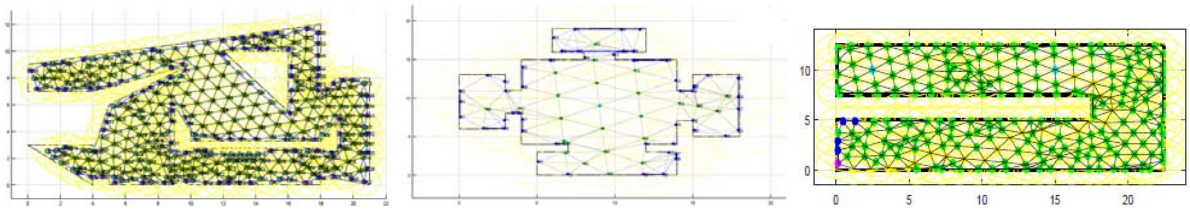
9

Complex structured scenarios

In this chapter we use different scenarios in order to show properties and limits of our proposed algorithm. These are the structures we adopted:

- type A structure (S_A), employed in three frameworks:
  - S_{A1} scenario: for showing the completeness of the attained coverage;
  - S_{A2} scenario: used for underling different robot models;
  - S_{A3} scenario: employed in the simulation with multiple base station;
- type B structure (S_B): this scenario well illustrates some limits of our algorithm regarding narrow passages;
- type C structure (S_C): the last scenario used to prove that the coverage policy (hexagonal packing) is guaranteed also in multiple base station frameworks and for exhibiting a lesser time cost with the multiple sources approach.

The three types of environment are represented in Fig. 9.1.



(a) Structure type S_A **(b)** Structure type S_B **(c)** Structure type S_C

Figure 9.1.: The three structures used in the following simulations.

9.1 Completion of the attained coverage

Now, we show how the algorithm works in a complex scenario. Figures 9.2 and 9.3 demonstrate that coverage is well accomplished respecting the constraints given by the environment S_{A1}. As one can see, expansion starts from the base station (robot 1, in cyan). Every node represents an agent with a visibility disk (yellow circles). Fence edges and nodes, in red, compose the current frontier which is expanding and magenta node displays the robot just deployed. Nodes in blue are evaluated as obstacles when robots touch walls. Edges in blue are set as obstacles when an expansion from two nodes cannot be attained in general: when their nodes are obstacle with the same touch sensor or when a corner of the scenario blocks communication. Green nodes and black edges exhibit the part of the network in which no computations occur.

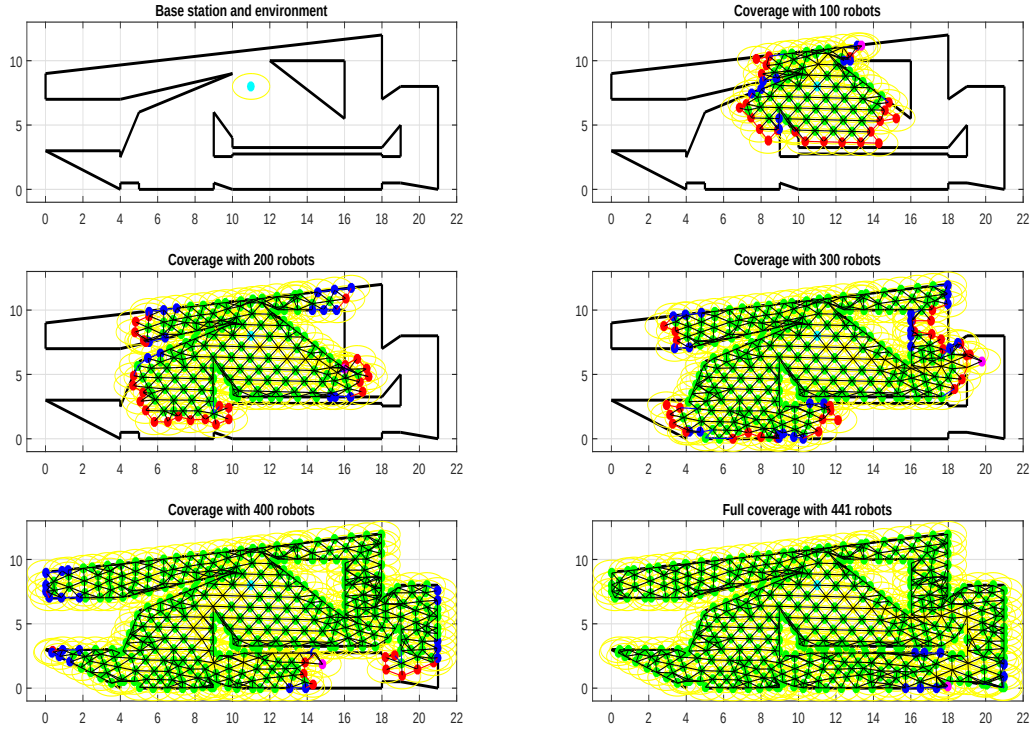
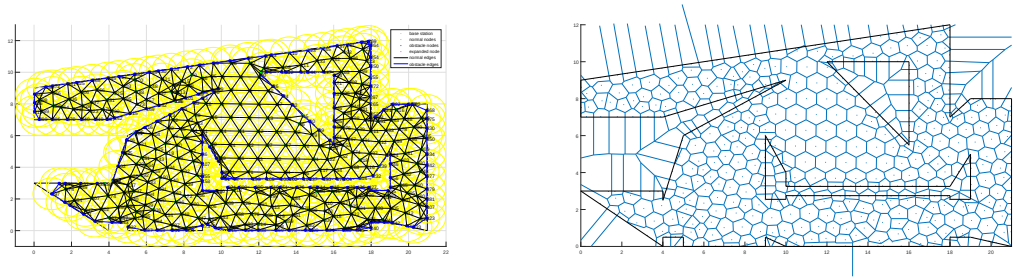


Figure 9.2.: Scenario S_{A1}. Step by step expansion coverage with different number of robots.



(a) Complete coverage in a structured environment

(b) Relative Voronoi partiotion

Figure 9.3.: Scenario S_{A1}. It can be seen the final configuration of the coverage shown on Fig. 9.2. $r_v = 1$, $\phi = 0.05$

9.2 Differences on physical model used for robots

Here, we show how robot model influences coverage capability through scenario S_{A2}. In particular in the first simulation shown in Fig. 9.4 a complete coverage is reached; on the other hand, in the second one illustrated in Fig. 9.5, robot size and the dimension of its visibility disk cause failure in coverage task.

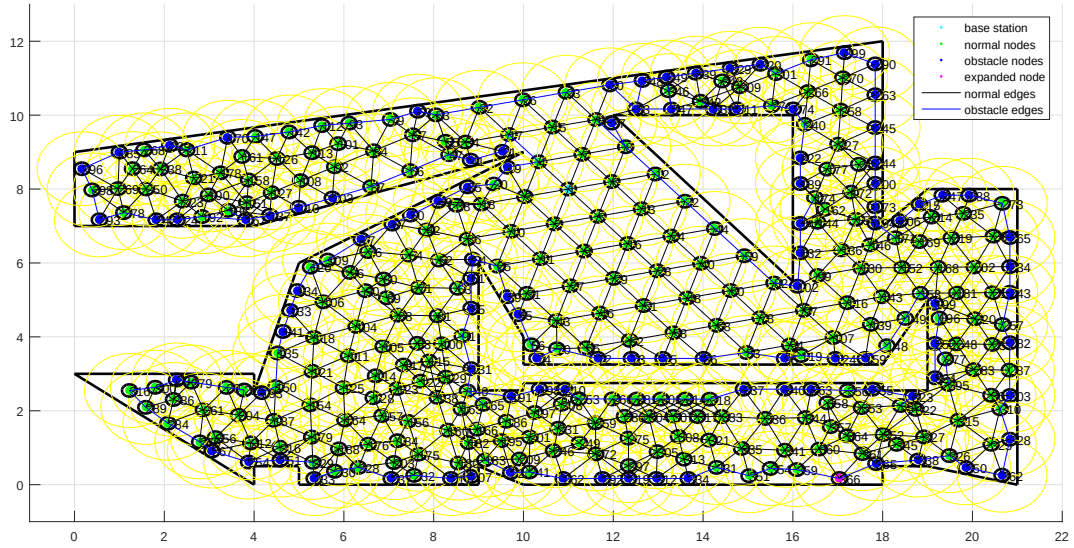


Figure 9.4.: Scenario S_{A1}. Simulation with $R = \frac{1}{6}$ and $r_v = 1$.

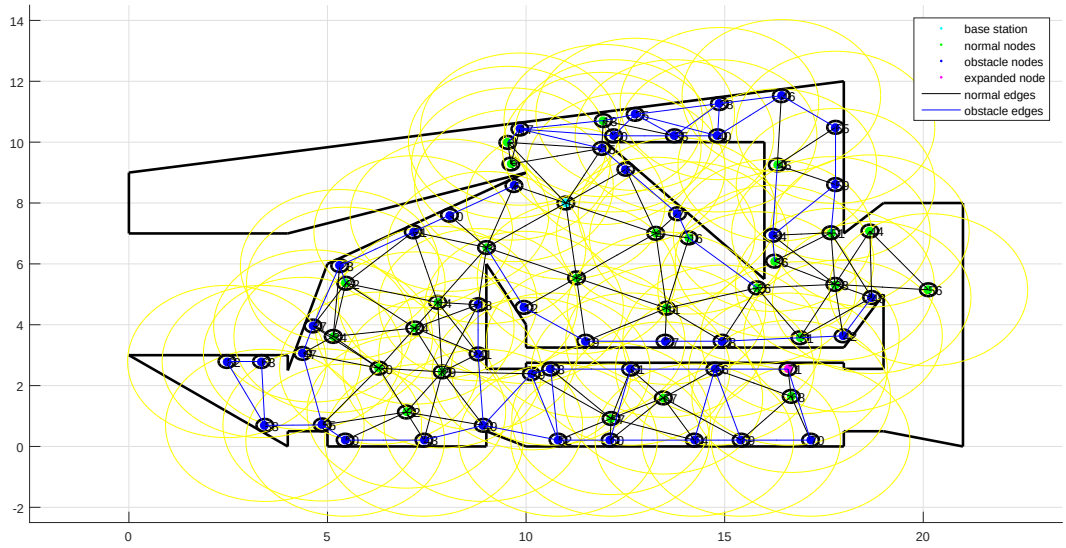


Figure 9.5.: Scenario S_{A2}. Simulation with $R = \frac{5}{12}$ and $r_v = \frac{5}{2}$.

9.3 Narrow passage limits

Our algorithm does not guarantee coverage in environment's parts connected to the base station through a narrow passage which dimensions are less than $r_v + \phi$, where r_v is the sensing radius of a robot and ϕ its diameter. In this section we adopt the scenario S_B for showing these limits. A lucky situation is presented in Fig. 9.6 and an example of a failed coverage completion is shown in Fig. 9.7. Here, the room has four narrow passages (left, right, up, down) respectively of 3, 4, 2, 5 and we recall that $r_v = 3$.

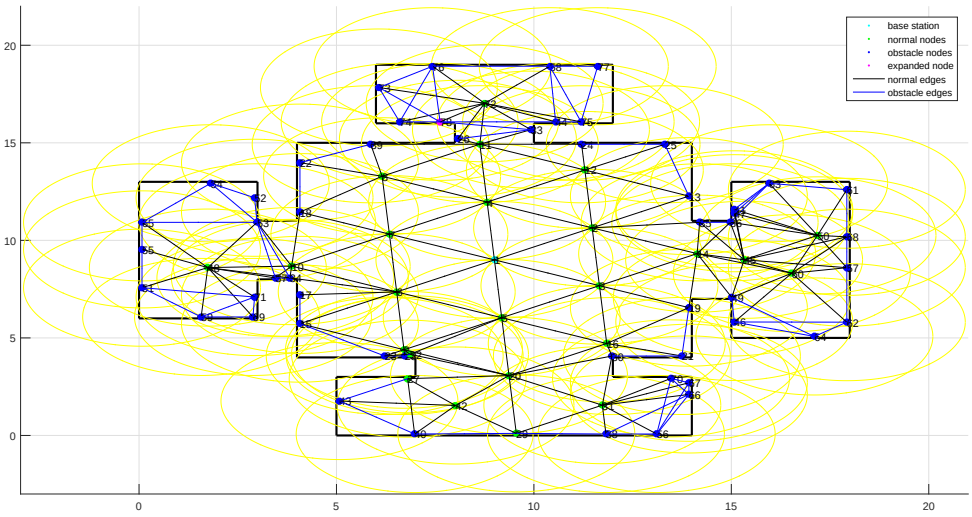


Figure 9.6.: Scenario S_B. Complete coverage luckily achieved. $r_v = 3$, $d = \frac{3}{20}$

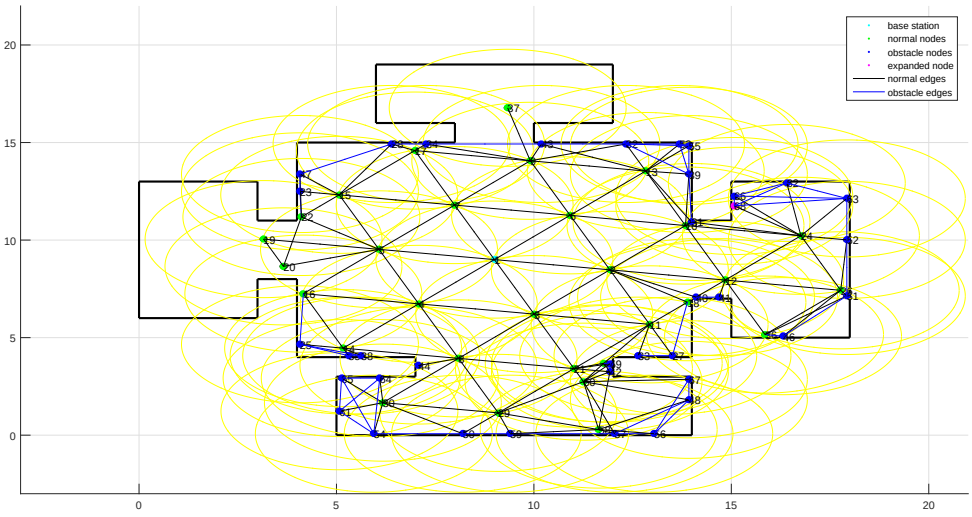


Figure 9.7.: Scenario S_B. Complete coverage failed because of narrow passages. $r_v = 3$, $\phi = \frac{3}{20}$

9.4 Multiple sources scenarios

In the following simulations multiple sources are employed and the environment S_{A3} is covered starting from two different location using protocol described in section 8.1.5. In the first situation presented in Fig. 9.8 the number of robots provided by the two base stations is more or less the same because an effective connection between sources is established after the deployment of a total of 144 robots.

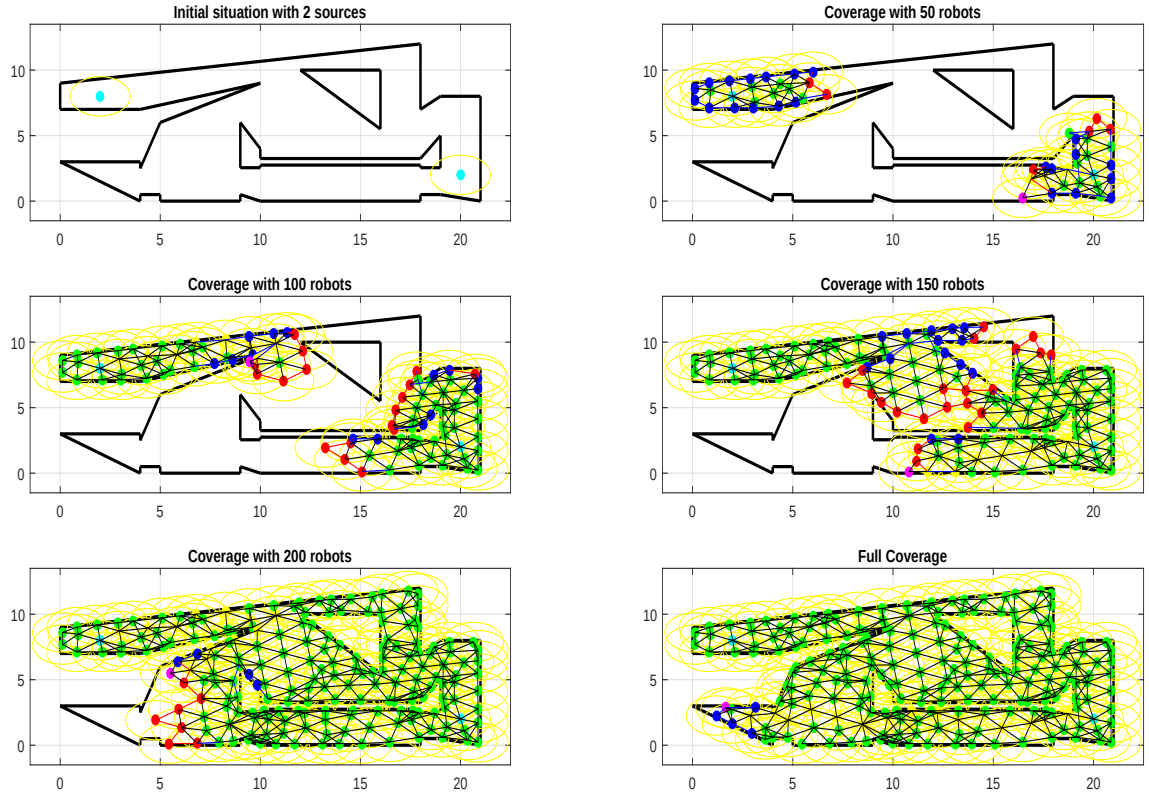


Figure 9.8.: Scenario S_{A3}. Step by step expansion with far multiple sources. $r_v = 1.5$, $\phi = \frac{r_v}{20}$

In the case S_C, shown in Fig. 9.9, two base stations are close each other and after 93 robots have been deployed, a connection is established. It is important to notice that after this connection, sources start to cooperate and all the robots deployed in the right and bottom part of the environment are generated by base station in $(15, 10)$ because frontier robots are further from base station in $(2.5, 10)$. Another important observation is that hexagonal packing is not guaranteed in the junction where different graphs merge. This can be solved by adopting a mesh/graph relaxing policy.

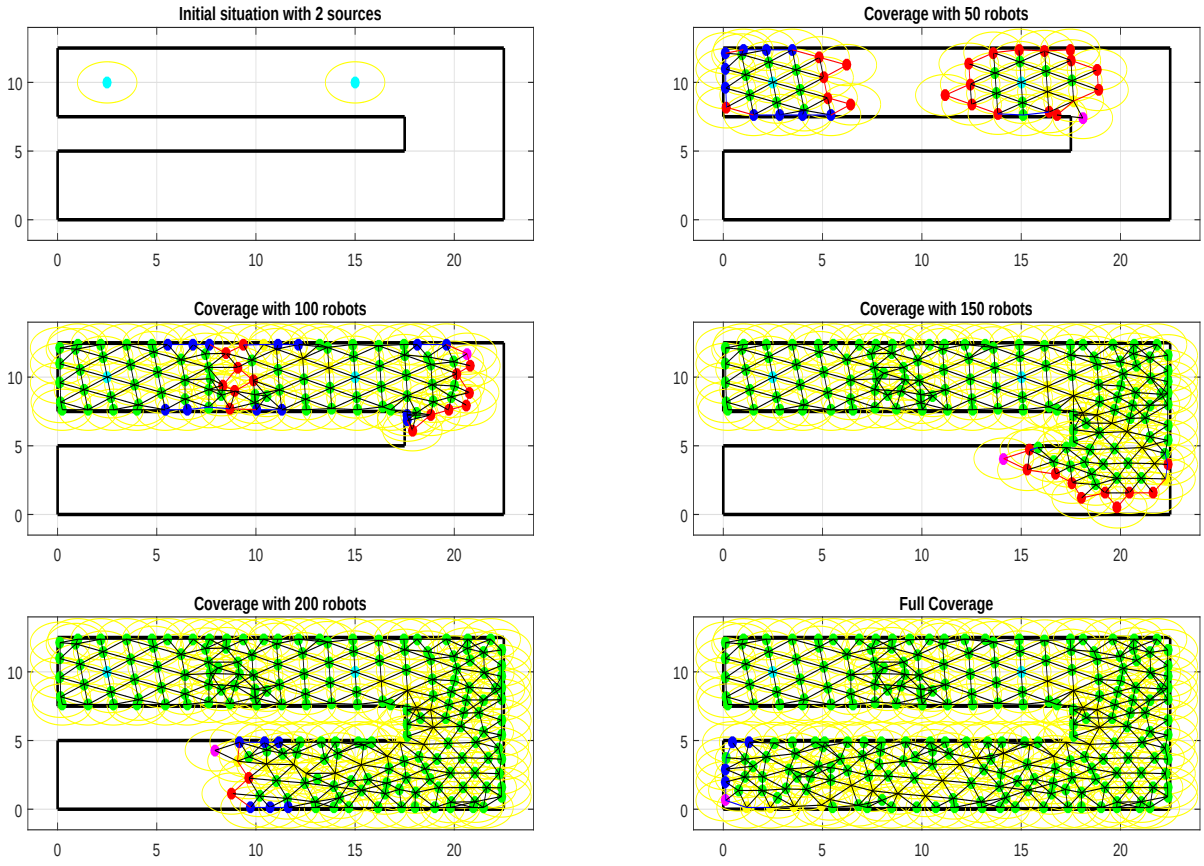


Figure 9.9.: Scenario S_C. Step by step expansion with close multiple sources. $r_v = 1$, $\phi = \frac{r_v}{20}$

Another important aspect to consider is the time cost, in terms of steps of computation, to deploy agents. In presence of one source, a step is intended as the operation to deploy another robot. Instead, with two sources, it is meant differently: if two robots are deployed by different sources there are no transient steps because, in reality, base stations can work simultaneously. In this case a step is counted only if two consecutive deployments are managed by the same base station.
Using scenario S_C, we report some results on the taken steps to attain complete coverage in figures 9.10 and 9.11: as one can expect, the more base stations we use the faster coverage is.

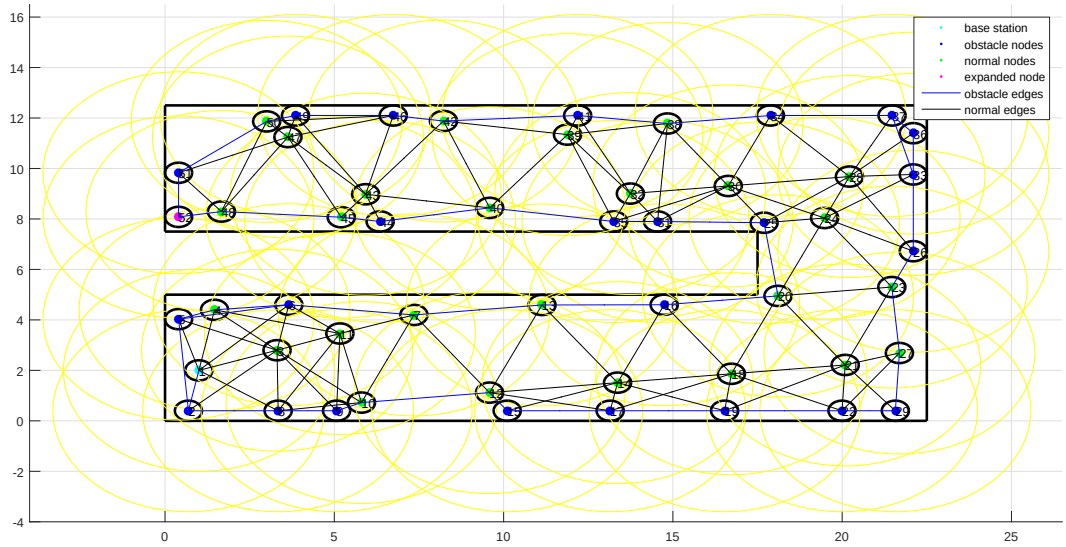


Figure 9.10.: Scenario S_3, 86 steps were needed in this simulation to cover it using 1 base station in $(1, 2)$. $r_v = 4$, $\phi = \frac{r_v}{5}$, $p_{rim} = 0.9$.

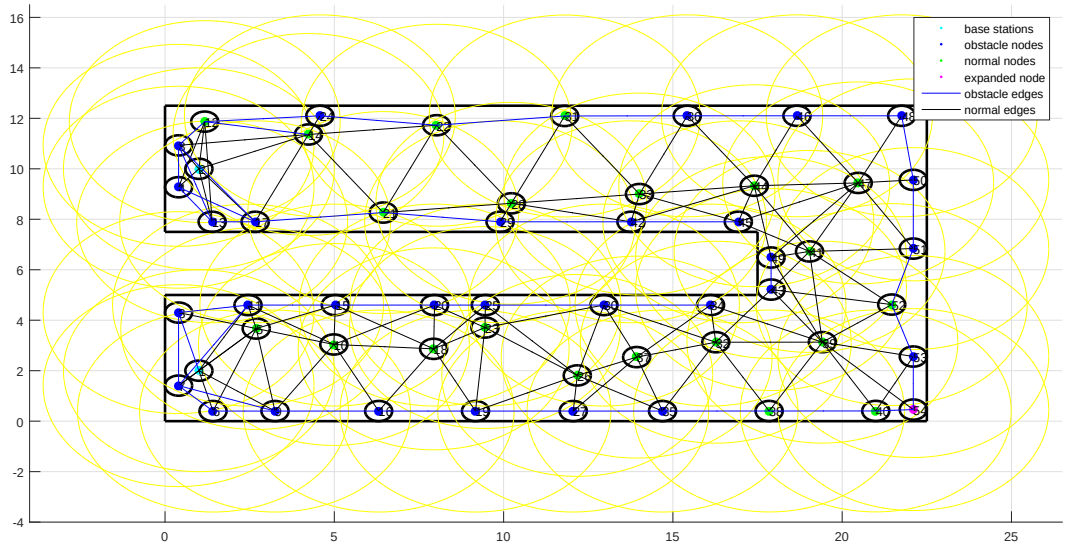


Figure 9.11.: Scenario S_3, 52 steps were needed in this simulation to cover it using 2 base stations in $(1, 2)$ and $(1, 10)$. $r_v = 4$, $\phi = \frac{r_v}{5}$, $p_{rim} = 0.9$.

10

Event scenarios

We have already seen that after coverage, when a communication sensor network has been created, robots can advise an event detecting it.
In this chapter two types of simulations are shown:

- robots that, after coverage is accomplished, detect an event and perform a dispatch;
- limits about the current control law: if an agent encounters a barrier or a wall between its trajectory, at the moment, it is not able to avoid it and bypass the mentioned obstacle.

10.1 Dispatch around an event

In this section, one can appreciate how agents pinpoint a randomly appeared event and how the dispatch is performed. See Fig. 10.1: here, a total coverage of the area has been accomplished and a graph shows that a sensor network has been created in a rectangular room.

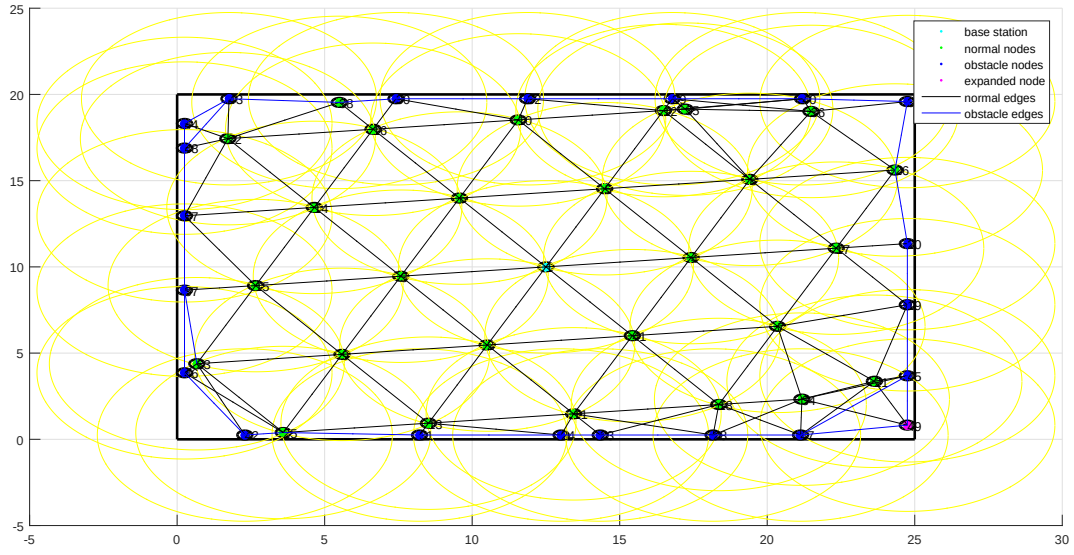


Figure 10.1.: Completely attained coverage in a rectangular room before the appearance of an event.

Then, with the fundamental hypothesis that robots have enough space to move, i.e. *no obstacles impede maneuvers to reach the selected target points*, we can observe which

trajectory the robots choose for dispatching[1] around the event arisen in the point $(20, 10)$. See Fig. 10.2: in this case, two robots have been selected for moving and forming a equilateral triangle that has the event as its baricentre.

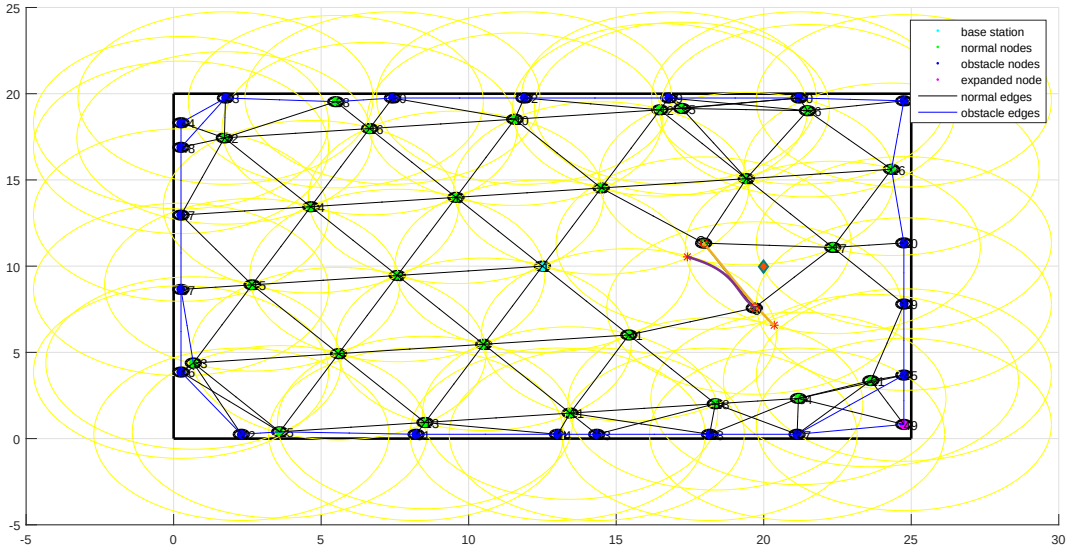


Figure 10.2.: Agent 17 coordinates agents 4 and 7 to form an equilateral triangle around the event arisen in $(20, 10)$. Note that 7 connections between the triangle and the reminder of the network are still active.

The next picture illustrated in Fig. 10.3 shows different choices of the number of robots for the dispatch: this number is set to be 6 at maximum in our algorithm, otherwise, with many attempts, we observed that the network could loose its connectivity at the end of the dispatch operations. In this case, forming a pentagon, we pass from 7 left connections to 5.
It is very clear that the more robots are employed to perform the dispatch the less the connectivity of the communication sensor network will be in the end. The reason why we set a maximum of 6 robots is well explained by Fig. 10.4: here only one connection holds on between the dispatched hexagon and the network. This is an important aspect if one wants to keep a distributed approach, since robots have to communicate via multi-hop passages of information.

[1] In this chapter we use the control law presented in section 7.6.2 for sending robots.

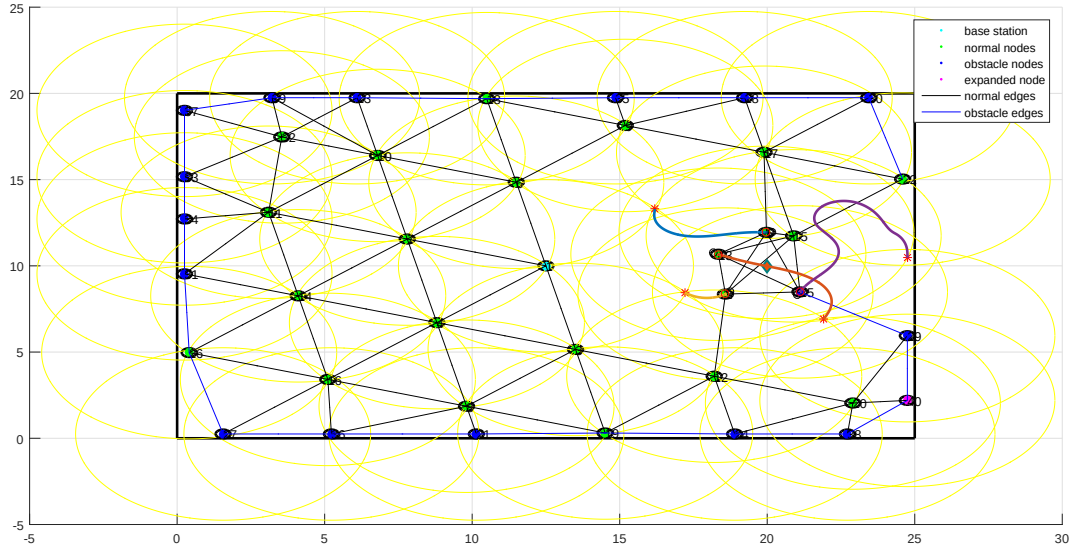


Figure 10.3.: Pentagonal formation of the dispatched robots around the event: 5 connections with the reminder of the network are still active.

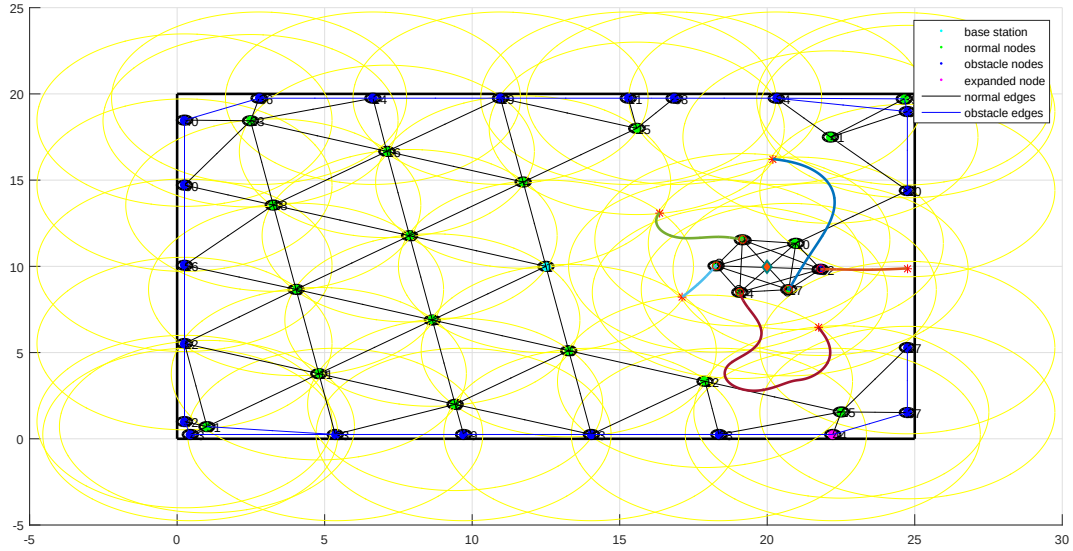


Figure 10.4.: Hexagonal formation of the dispatched robots around the event: 1 connection with the reminder of the network is still active.

However, studies on which is the real maximum number of dispatching agents are matter of future works: in this thesis we can just state that more than 6 robots sent close to the event may cause a failure of the distributed communications.

10.2 Dispatch control law limits

A remarkable limit of the dispatch operation is the impossibility of avoiding to break into walls and bypass barriers in the environment. This is due to the control law that, in this thesis, does not consider barriers yet. See Fig. 10.5 to better understand the fact.

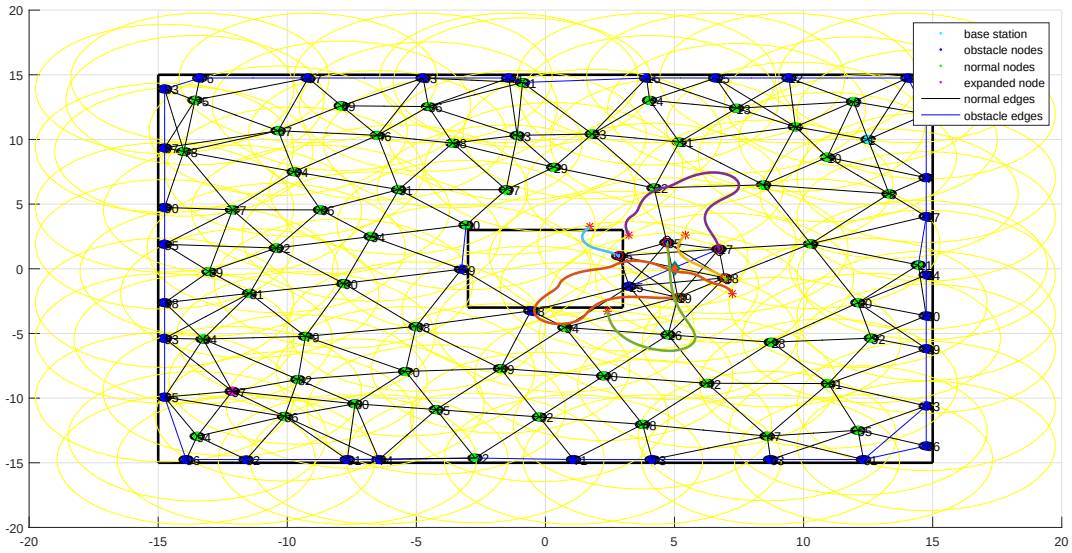


Figure 10.5.: Hexagonal formation of the dispatched robots around the event. The computed trajectories do not take into account the presence of a squared obstacle in the middle of the scenario.

Improving the current control law by considering also the environment constraints will be object of future studies.

Part V

Conclusions and appendix

11

Conclusions

In this thesis we proposed a coverage algorithm based on the article in [2]. We developed a strategy which is able to perform a deployment in static-obstacle-structured environments in a context of trade-off between local communication and optimal coverage.
The particular aim and tasks we chose are

- the maximization of local communication: we reformulated the algorithm in order to remove the assumption of a centralized server which collects local information from individual robots and performing the distributed approach as far as possible;
- the definition of an expansion policy when we have multiple source of robot swarms;
- the insertion of the possibility to detect small non-convex features in the environment which usually result "blind-spots" for some obstacles, assuming a space structure model for robots (e.g. giving them a round shape) in order to define the limitation of smallest non-convex coverable features;
- the possibility of event detection and focusing by sending a group of robots.

11.1 Guarantees and limits

These are guarantees of our proposed algorithm.

- **Algorithmic distributed completeness**. Algorithm never gets stuck in an infinite cycle because in every new placement coverage area expands. Before each deployment we check the possibility of a creation of a hole inside the graph due to robot failure, noise or error accumulation. In this situation a new robot is placed for filling lacking area. The algorithm uses local information only and terminates only when the fence subcomplex F is empty. This always happens if the area we are covering is finite and we have sufficient number of robots.
- **Optimal local coverage**. Far from obstacles hexagonal packing is obtained; therefore, we ensure local optimal coverage. Finding and removing redundant robots allows us to use the minimum number of agents the environment needs to be covered, with this kind of algorithm.
- **Robustness to failure**: if a single robot fails and it is close enough to the expanding frontier then the sensor network is able to adapt to this lack. In the case that a robot failure occurs far from the expanding frontier the hole can be detected from a periodic check of the whole graph connections between agents.

- **Collision avoidance**. Two robots neither lay on the same place nor crash to each other, according to the collision policy adopted. When robots hit walls they respect the interaction model described in section 7.4. The collision avoidance protocol is adopted only during deployments.
- **Multiple source improvement**. If two or more base stations are used then coverage speed greatly increases until networks connect the sources.
- **Focusing on an event**. If a robot has enough space of maneuver, it easily can be sent in a target point near an event for monitoring.

The main limitations of our work follow.

- Without using an accurate inspection of obstacle edges, basic part of the algorithm does not guarantee coverage in feature with dimension smaller than $r_v + \phi$, where r_v is the sensing radius and ϕ is the diameter of a robot body.
- During the dispatch operations, barriers and walls cannot be avoided because the control law does not take into account obstacles yet;

11.2 Future developments

Several developments should be studied in the future. Here some examples are written.

- Finding features smaller than $r_v + \phi$ is still an open challenge. During some recent collateral works, we thought about a solution that could be a starting point for the inspection of inlets in a highly complex structured scenario. See Fig. 11.1: sending a "probe" of robots could be a good idea to complete coverage in all areas belonging to an environment.
- In this thesis, measurements have been considered 100% reliable in terms of errors. In a real context, we should think about noisy measurements of the bearing angles and the event intensity and start to imagine a good strategy to perform deployments and moving robots. This fact is important, because if one wants to use real robots, for instance, dispatch around an event could fail since target positions are not computed correctly.
- We did not prove rigorously[1] the validity of the control law yet. We should move in two directions for verifying it and to be sure that our law is optimal:
  - prove the asymptotic (or, at least, simple) stability with Lyapunov methods, as described in [21] and [22]: this aspect is very important because a robot should not leave too far away from its "leader" during the dispatch operation, otherwise it could loose signal, information and communication could not be exchanged and feedback could not be used to fulfill the task;
  - try to find optimal trajectories that minimize (7.7) and make some comparisons with our developed control law;

[1] however, its formal derivation is explained in appendix A

- Improve our collision avoidance protocol in every situation, from coverage to dispatch operations. In particular, we should insert the model of obstacles in the control law and ensure the robots are able to reach a target point, even tough impacts occur.

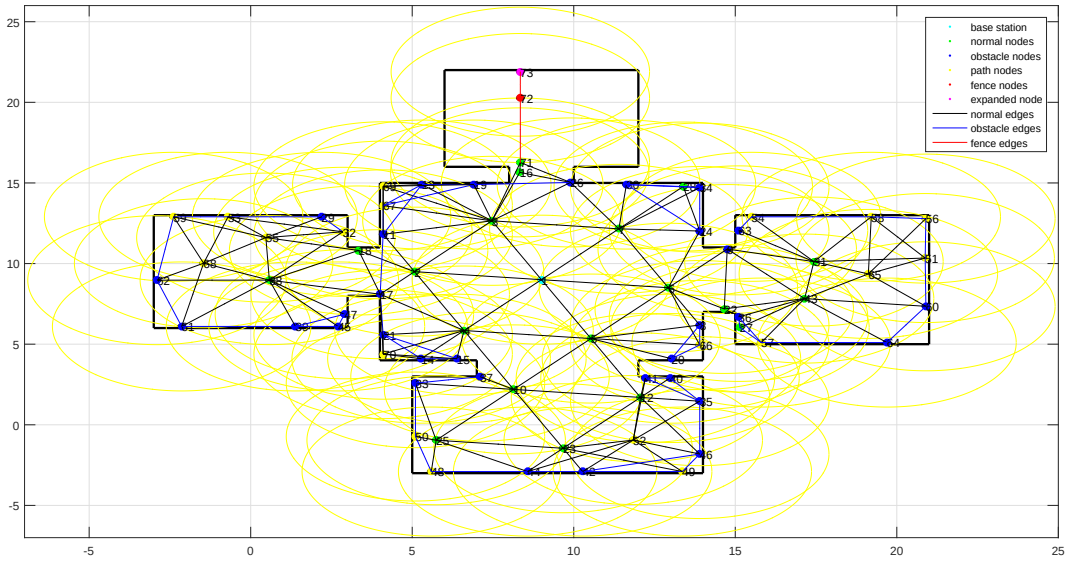


Figure 11.1.: Simulation with $\phi = 0.2$ and $r_v = 4$: a "probe" (in red) has been used to discover the upper room in this scenario with four inlets.

A

Control law for a front-wheeled car robot

The control law used for front-wheeled car robots is obtained by a combination of two elements. The first part is derived by a basilar control law for car-like robots we have taken from [23]; however, since the latter model is not the one we use, we had to manipulate it. The second part is made by heuristics formulas that help[1] to find a convergence to targets and improve the stability of the controlled model.

A.1 Formal derivation

According to [23], there exists a simple control law for the car-like robot model. We recall that its dynamics is described by nonlinear system in (A.1):

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \dfrac{v}{L}\tan\gamma \end{cases} \tag{A.1}$$

where v and γ are respectively the input velocity (more precisely speed with sign) and input steering angle. The mentioned control consists in the following algorithm.
Velocity control:

1. $d = \sqrt{(x^* - x)^2 + (y^* - y)^2}$ (distance between current position and target);
2. $v = k_v d$.

Steering angle control:

1. $\theta^* = \arctan\left(\dfrac{y^* - y}{x^* - x}\right)$ (desired planar angle of the vehicle);
2. $\gamma = k_\gamma(\theta^* - \theta)$ where k_γ is a positive proportional constant.

The velocity control has just to be modified as regarding the sign because k_v is a positive constant: on the next section we will show how to design reverse movements.
The steering angle control is radically modified, since, for a font-wheel car model, this

[1] in section 8.4 we showed some problems related to a control law without heuristics

input becomes a state. Indeed, we have to think about its dynamics described by (A.2):

$$\begin{cases} \dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \dfrac{v}{L}\tan\gamma \\ \dot{\gamma} = \omega \end{cases} \tag{A.2}$$

Here, we have to design ω, which is the input steering angular velocity. In this case we assume that

$$\omega = k_o\left(\dot{\theta}^* - \dot{\theta}\right)$$

similarly to point 2 in the previous steering angle control. Then, assuming that variable $\dot{\theta}$ is observable, we try to compute $\dot{\theta}^*$: the desired planar angular velocity for the vehicle. What follows seems to be a valid idea, since simulations showed in section 8.4.
From point 1 of the steering angle control:

$$\theta^* = \arctan\left(\frac{y^* - y}{x^* - x}\right)$$

that is equal to

$$\tan\theta^* = \frac{y^* - y}{x^* - x} \tag{A.3}$$

Differentiating both sides of (A.3) we get

$$(1 + \tan\theta^*)\dot{\theta}^* = \frac{-\dot{y}(x^* - x) + \dot{x}(y^* - y)}{(x^* - x)^2} \tag{A.4}$$

and substituting (A.3) in (A.4) we obtain

$$\dot{\theta}^* = \frac{\dot{x}(y^* - y) - \dot{y}(x^* - x)}{(x^* - x)^2 + (y^* - y)^2}$$

therefore

$$\dot{\theta}^* = \frac{1}{d^2}\left(\dot{x}(y^* - y) - \dot{y}(x^* - x)\right)$$

A.2 Heuristic corrections

So far, the control law we have got is the following.
Velocity control:

1. $d = \sqrt{(x^* - x)^2 + (y^* - y)^2}$ (distance between current position and target);
2. $v = k_v d$.

Angular steering velocity control:

1. $\dot{\theta}^* = \dfrac{1}{d^2}\left(\dot{x}(y^* - y) - \dot{y}(x^* - x)\right)$;
2. $\omega = k_o\left(\dot{\theta}^* - \dot{\theta}\right)$.

This is the control law we said to be "rough" in section 8.4.
As regards the heuristic corrections, we proposed two improvements:

- introducing reverse movements, i.e. compute a sign $s = \pm 1$ to insert in the velocity control $v = sk_v d$;
- introducing a converging function c that helps to turn the steer when the vehicle is not aligned with the target or it is moving backward.

The sign we choose to compute in section 7.6.2 is provided by (A.5):

$$s = \text{sign}\left(|\Delta\theta| - \frac{\pi}{2} - \left(1 + \frac{v_{BACK}^{MAX}}{v_{FOR}^{MAX}}\right)\frac{\pi}{2}\right) \tag{A.5}$$

where the meaning of the quantities is explained in table 7.1. Note that the term $\eta = 1 + \frac{v_{BACK}^{MAX}}{v_{FOR}^{MAX}}$ is a positive coefficient lesser than 1 because v_{BACK}^{MAX} is a negative or null speed and $\left|v_{BACK}^{MAX}\right| < v_{FOR}^{MAX}$. Since we have $0 \leq \eta < 1$ then we are making a comparison with the angle $|\Delta\theta|$ and the critic angle $\theta_s = (1+\eta)\frac{\pi}{2} \leq \pi$. If, in this comparison $|\Delta\theta|$ is the minor angle, we state that the vehicle is "bad-oriented", in the sense that the direction of the vehicle falls into a blind cone of non-visibility for reaching the target, as shown in Fig. A.1.
As second aspect for corrections, we consider the c function because it is linked to the convergence to the target: c is a non linear function of p, which, basically, is the scalar product between the unit vector of the direction of the vehicle w_{veh} and the unit vector representing the desired direction to reach the target w^*. In Fig. A.1, it is illustrated by using different colors which values of c modify the first equation for the angular steering angle control.

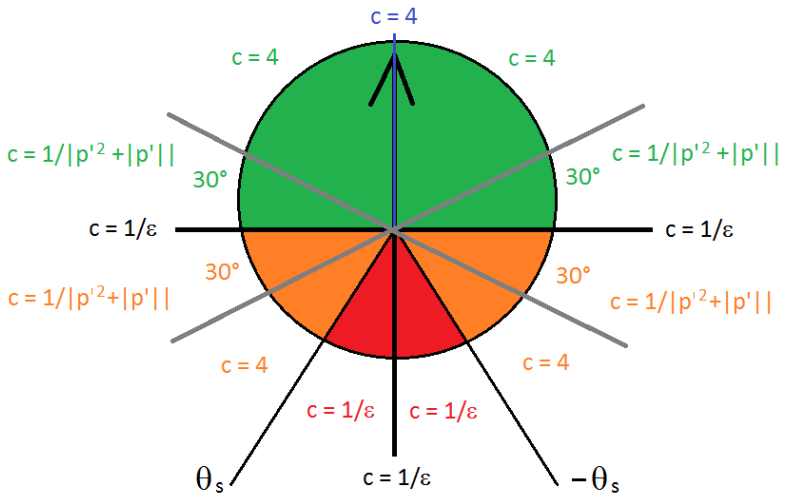


Figure A.1.: Blind cone of non-visibility in red, contribution of η for describing the angle θ_s (taken from the blue line) in orange. The arrow indicates the desired direction for the target. If one decides not to set reverse movements in general, i.e. $v_{BACK}^{MAX} = 0$, the non-visibility cone becomes wide as both red and orange parts, since θ_s would be equal to $\frac{\pi}{2}$.

Bibliography

[1] M. Lukic and I. Stojmenovic, "Energy-balanced matching and sequence dispatch of robots to events: Pairwise exchanges and sensor assisted robot coordination," Dec 2013, pp. 249–253.

[2] S. B. R. Ramaithitma, M. Whitzer and V. Kumar, "Sensor coverage robot swarms using local sensing without metric information," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Washington State Convention Center. Seattle, Washington, May 26-30, 2015.

[3] J. Hespanha, "A survey ofrecentresults in networked control systems," vol. 95 No. 1, Jan 2007, pp. 138–162.

[4] K. Romer and F. Mattern, "The design space of wireless sensor networks," in *Wireless Sensor Networks*, Dec. 2004, pp. 54–61.

[5] V. F. Q. Du and M. Gunzburge, "Centroidal voronoi tessellations: Applications and algorithms," in *SIAM Review*, vol. 41 No. 4, 1999, pp. 637–676.

[6] Z. K. M. Z. N. Dziengel, S. Adler and J. Schiller, "Cooperative event detection in wireless sensor networks," Dec 2012, pp. 124–131.

[7] S. F. T. P. Lambrou, Christos G. Panayiotou and B. Beferull, "Exploiting mobility for efficient coverage in sparse wireless sensor networks," vol. 54, 2010, pp. 87–201.

[8] N. C. Samuel Rutishauser and A. Martinoli, "Collaborative coverage using a swarm of networked miniature robots," in *Robotics and Autonomous Systems*, vol. 57(5), 2009, pp. 517–525.

[9] V. de Silva and R. Ghrist, "Coordinate-free coverage in sensor networks with controlled boundaries via homology," in *The International Journal of Robotics Research*, vol. 25(12), 2006, pp. 1205–1222.

[10] V. D. Silva and R. Ghrist, "Coordinate-free coverage in sensor networks with controlled boundaries via homology," vol. 25 No 12, 2006, pp. 1205–1222.

[11] J. D. R. Ghrist, D. Lipsky and A. Speranzon, "Topological landmark-based navigation and mapping," 2012.

[12] V. D. Silva and R. Ghrist, "Coverage in sensor networks via persistent homology," vol. 7, 2007, pp. 339–358.

[13] S. P. F. A. K. SeoungKyou Lee, Aaron Becker and J. McLurkin, "Exploration via structured triangulation by a multi-robot system with bearing-only low-resolution sensors," vol. CoRR, abs/1402.0400, 2014.

[14] A. Zomorodian, "Fast construction of the vietoris-rips complex," in *Computers and Graphics*, vol. 34, 2010, pp. 263–271.

[15] M. T. Goodrich and R. Tamassia, "Data structures and algorithms in java, 5th edition," 2011, pp. 594–641.

[16] V. K. J. Derenick and A. Jadbabaie, "Towards simplicial coverage repair for mobile robot teams," May 2010, pp. 5472–5477.

[17] D. A. Anisi, "Optimal motion control of a ground vehicle," July 2003, pp. 1–24.

[18] G. Antonelli, "An overview on distributed control," in *Interconnected Dynamic Systems*, Feb. 2013, pp. 76–88.

[19] H.-C. Chang and L.-C. Wang, "A simple proof of thue's theorem on circle packing," ArXiv e-prints, September 2010.

[20] S. H. M. Sghaier, H. Zgaya and C. Tahon, "A distributed dijkstra's algorithm for the implementation of a real time carpooling service with an optimized aspect on siblings," in *In Intelligent Transportation Systems (ITSC), 13th International IEEE Conference*, Sept 2010, pp. 795–800.

[21] H. K. Khalil, in *Nonlinear Systems, third edition*, 2002, pp. 1–194.

[22] E. Fornasini, in *Appunti di Teoria dei Sistemi*, Libreria Progetto.

[23] P. Corke, in *Robotics, Vision and Control. Fundamental algorithms in MATLAB.*, 2011, pp. 65–86.