POLITECNICO DI MILANO
School of Industrial and Information Engineering
MSc in Computer Science and Engineering

A Distributed Approach to Multirobot Persistent Patrolling in Communication-Restricted Environments

Supervisor: prof. Francesco Amigoni
Co-Supervisor: ing. Jacopo Banfi

Master thesis by:
Marta Romeo, ID 824238

Academic year 2015-2016

Contents

Abstract **3**

Sommario **5**

1 Introduction **7**

2 State of the art **11**
2.1 Centralized patrolling strategies 12
2.2 Distributed patrolling strategies 16

3 Problem setting **21**
3.1 Environment definition 21
3.2 Performance metric definition 22

4 Patrolling strategies **25**
4.1 Globally optimal strategy 26
4.2 Individually optimal strategy 27
4.3 Reactive strategy 29

5 System architecture **31**
5.1 ROS: Robot Operating System 31
5.2 Nodes implementation 33

6 Experimental evaluation **39**
6.1 Experiments setup 39
6.2 Experiments and results 43
6.2.1 Comparison between G-OPT and distributed strategies . . 43
6.2.2 Variable number of robots 44
6.2.3 Variable planning horizon 48
6.2.4 Variable number of *c*-type vertices 48
6.2.5 Variable probability of communication failure 50

7 Conclusions **51**

Abstract

This thesis addresses the field of multirobot patrolling, where a team of mobile robots is deployed in an environment of interest with the aim of keeping under observation a set of predefined locations. In this field of research, a typical assumption is that robots behave as autonomous entities capable of performing a "cleaning action" once a problem in one of the locations to patrol is detected. However, there are some realistic mission scenarios, in which the deployed robots cannot identify a particular problem or cannot autonomously act in order to solve it (think about a leak of contaminants or the presence of an intruder), and in which it is more realistic to assume that the data must be reported to a base station to assess the situation and decide what to do. In such applications, most of the time, human operators supervising the evolution of the mission, working at the base station, are only able to identify the problem through the data acquired by the robots. Moreover, in many mission scenarios, only some regions of the environment are covered by a communication infrastructure able to allow message exchange with a sufficient bandwidth between the robots and the base station. In this thesis we want to propose and evaluate some patrolling strategies in such environments in which communication is restricted to some regions only. We first extend the classical multirobot persistent patrolling framework, and the related evaluation metrics, in order to account for the presence of a limited number of such "communication zones". Then, we present some centralized and distributed patrolling strategies we devised to deal with this new communication-restricted patrolling setting. We implement the proposed strategies within the Robot Operating System (ROS) framework and evaluate them in simulation in a number of different settings.

Sommario

Questa tesi si colloca nell'ambito del pattugliamento multirobot, in cui una squadra composta da più robot, mobili e autonomi, ha il compito di visitare continuamente un ambiente, con il compito di tenere sotto osservazione alcuni luoghi di interesse. In queste applicazioni, solitamente, i robot vengono considerati entità autonome capaci di far fronte a un problema rilevato in una delle aree da pattugliare. Esistono però scenari in cui può accadere che i robot impiegati per la missione non siano in grado di identificare un problema o non siano in grado di risolverlo (ad esempio una perdita sostanze contaminanti o la presenza di un intruso). In tali casi è perciò più realistico supporre che i robot debbano riportare i dati acquisiti durante la loro esplorazione ad una base station, dove gli operatori umani che supervisionano l'andamento della missione sono in grado di identificare un problema, grazie ai dati raccolti dai robot. Inoltre, può anche accadere che la comunicazione sia consentita solo da alcune regioni dell'ambiente, poichè è possibile che solo in queste specifiche regioni l'infrastruttura di comunicazione sia in grado di garantire una sufficiente larghezza di banda che permetta ai messaggi inviati dai robot di raggiungere la base station (e viceversa). Con questa tesi vogliamo presentare delle strategie di pattugliamento che permettano ai robot di completare la missione in ambienti in cui la comunicazione è possibile solo nelle aree appositamente adibite per essa. Per prima cosa, abbiamo esteso il framework del tradizionale pattugliamento multirobot e le relative metriche di valutazione in modo da prendere in considerazione la presenza di un numero limitato di "zone di comunicazione". Successivamente, abbiamo implementato delle strategie distribuite che i robot possano seguire per portare a termine la missione nel nuovo framework. Queste strategie sono state implementate usando il framework Robot Operating System (ROS) e sono state valutate tramite una serie di simulazioni che prendono in considerazione diverse situazioni.

Chapter 1

Introduction

In the last few years, a lot of effort has been devoted, in the robotics and AI communities, to the study of effective patrolling strategies in different settings and under different assumptions. More specifically, "to patrol" means to walk or travel around an area, at regular intervals, in order to protect or supervise it. In this context, the term "persistent" is added to underline that robots must continuosly patrol the environment, for the entire duration of the mission, in order to periodically cover the whole area. In the classical formulation of the *multirobot persistent patrolling problem*, a team of robots is deployed in an environment represented as an undirected, weighted graph, with the aim of minimizing the time between two subsequent visits to the same location (vertex) to be patrolled. The main performance metric measure used to quantify coverage received by a location is thus *idleness*, defined as the time elapsed since that location has received the visit of a robot [1]. Evaluation metrics related to this notion, such as *average* and *worst-case idleness*, have been introduced and investigated for different patrolling strategies and in a number of different conditions [1–5].

The implicit underlying assumption of these approaches is that the deployed robots have the possibility of identifying a particular problem in any vertex they visit, and of autonomously act in order to solve it. However, there are mission scenarios in which this assumption does not hold, think about a leak of contaminants or the presence of an intruder. For this reason, in these cases, it is realistic to assume that the data collected by the robots must be reported to a base station to assess the situation and decide what to do, as most of the time, in such applications, the human operators supervising the evolution of the mission, who are working at the base station, are only able to identify the problem through the data acquired by the robots. If one thinks about possible existing communication infrastructures that could be exploited by the robots to send reports back to the base station, such as tactical networks [6] or even cellular networks [7], it could be the case that only

some regions of the environment are covered by a sufficiently strong signal. The presence of such "communication zones" in the environment to patrol has been recently investigated in [8]. In that work, the focus is on centrally building a single inspection tour for the team of robots with the aim of minimizing the time lag between the visit of a location and its report to the base station.

In this thesis, we propose new distributed strategies to fulfill the patrolling task in such communication-restricted environments. Beyond a centralized strategy (calculated at the base station) that does not scale to realistic-size settings, we propose two distributed strategies. These strategies rely on the robots calculating the plan they must follow by themselves, using the base station only as a collector for informantions essential to the computation of the plan. Moreover, they are online, and hence particularly suited for coping with unpredictable events like, e.g., robots interfering with each other while executing their planned paths, or temporary unavailability of a communication link with the base station. The objective of the team of robots is to minimize the *average graph communication idleness*. The classical definition given in [1] describes the *idleness* as the time elapsed between two consecutives visits of a vertex. This definition does not fully fit in our communication-restricted framework beacause it does not take into account the restrictions on the possibility of communicating. For this reason, we extend this concept to *communication idleness*, defined as the time elapsed between two different reports to the base station on the same vertex. In this way, both the time of visit of a node and that of its actual communication to the base station are considered.

In order to model the communication-restricted environment in which we develop our framework we adopt the representation used in [8]. In this work the environment is discretized by an undirected graph in which vertices represent all the locations robots can visit, while edges encode physical connections between them. The vertices are divided in two categories: m-type vertices, representing the locations the robots must monitor, and c-type vertices, representing the locations from which it is possible to commununicate. Notice that this two sets are not necessarily disjoint. Even though we use the same discretization, the objective of this thesis differs from the one in [8] where the patrolling problem was solved in a centralized way, by the base station. Instead, the distributed strategies we develop are based on the interaction between robots and base station, which is possible only in the c-type vertices. Once a robot visits a c-type vertex, it queries the base station on the current values of the *vertex communication idlenesses* and on the plans its teammates are currently following. After having received the information it needs, it compute its own plan, accordingly to the strategy it is using: a reactive one, leading it to the node displaying the highest vertex communication idleness, or a strategy enumerating all possible plans and choosing the one that guarantees the lower *average graph communication idleness* at the end of its execution.

Finally, the proposed patrolling strategies are implemented within a Robot Operating System ROS architecture, and ROS/Stage simulations are run to shed light on their effectiveness. In order to test our system and the implemented strategies we carried out several experiments in three environments. The main objective of these experiments is to compare the distributed strategies and their performances in the different settings, not only among themselves, but also against the centralized strategy. The results obtained proved that the distributed strategies we introduce in this thesis perform as good as the centralized one, even if they are in general sub-optimal, lowering down the computational load for the calculation of the joint plan by dividing it among the robots in the team.

Since our communication-restricted framework has not been fully explored, there is still work to do. Possible future work could investigate new distributed strategies or focus on implementing a good recovery strategy, in case a message is lost or a robot of the team breaks down.

The structure of this thesis is the following: In Chapter 2 we discuss a number of works that are related to ours and that were of inspiration for this study. In Chapter 3 the model of our system is discussed, together with a more formal definition of the problem we are dealing with. In Chapter 4 we introduce centralized and distributed strategies we develop to solve the patrolling task, pointing out their differences. In Chapter 5 the software architecture used to implement the strategies is presented. In particular, we first give a short overview of the ROS framework and the Stage simulator, in which we build and test our system, and then we give the details of the implementation of the architecture. In Chapter 6 we show the experiments we perform to test our framework and compare the patrolling strategies, along with the obtained results. In Chapter 7, we conclude by summarizing the purposes and the final evaluations of this thesis. Some suggestions for future works are also proposed.

Chapter 2

State of the art

Persistent multirobot patrolling is a rather modern issue in the field of robotics and multiagent systems. During the last years a variety of strategies and algorithms have been developed to try to find proper solutions to this problem which, is far for being trivial, given the fact that its solution isn't univocal but it may depend on the assumptions of a specific setting. There are in fact a lot of aspects that play an important role in determining the outcome of the patrolling task.

First of all, there are different kinds of possible discretizations of the environment that robots have to patrol. The representation of the environment can influence the choice of the algorithm used for the motion of the robots and also the kind of monitoring we want to perform. The constraints imposed to the robots are another important element that can differentiate the strategies. They can include both physical limitations for the robots, such as the maximum physical velocity reachable or battery level restrictions, and constraints on the plan they have to follow, that can be subjected to communication constraints or time-budget constraints. The performance metric used to evaluate the final results is another key factor to point out the differences between patrolling strategies. Also the presence or absence of adversarial entities can be significant for the final result.

The purpose of this chapter is to present the main features differentiating from one work to another, mainly dividing the relevant literature in two big categories of algorithms: centralized and distributed. The reason of this choice is to better underline the original contribution of this thesis: working to find a distributed strategy and compare it against existing centralized ones in a novel communication-restricted patrolling framework. A more comprehensive summary of all features found in the literature of multiagent patrolling can be found in Figure 2.1 at the end of the chapter.

2.1 Centralized patrolling strategies

Centralized patrolling strategies are those that rely on a central entity to compute the plan that robots have to follow to fulfill the patrolling task. The ability of producing the best plan possible is entirely in this coordinator and what robots have to do is simply to follow its instructions and complete the assigned plan. What they have to take care by themselves is, most of the time, path planning with obstacle avoidance. Most of the works encountered during our literature review follow this strategy. As communication is an important feature for us, we decided to organize the presentation of the following works presenting the ones in which communication is not modeled, or not needed, in the first place, and leaving the only centralized work we examined that models it as the last one.

In [2] Chevaleyre provides two examples of centralized planning strategies for the patrolling problem, aiming at continuously visiting all the graph vertices following a predefined path, minimising the idleness of visits to the locations. The idleness is defined as the amount of time elapsed since two consecutive visits to a vertex, as introduced in [1]. The proposed startegies are: *cyclic strategies*, augmented with an heuristic so that they compute a Travelling Salesman Problem (TSP) cycle (which is proven to be optimal for one robot performing the task), and *partitioning strategies*. *Cyclic strategies* starts and ends in the same vertex while *partitioning strategies* divide the environment and assign each patrolling region to a different robot, with the path robots have to follow in their region being cyclic too. Althought the main goal of the paper was to find a proper bound for these two different techniques, in terms of worst-case idleness, rather than to compare them, it is still possible to do so. First of all, the discretization of the environment done in this case is through an unidirected graph whose vertices are locations in the environment and edges represent the physical topology of the environment. Since the graph is metric, the cost associated to each edge represent the distance between vertices, which corresponds to the time taken by a robot to move, assuming uniform robot speed. No particular constraint is imposed on the robots except the one that guarantees a fixed distance gap between them while following the same cyclic path. Communication issues are not addressed in this work. To compare the performances of the strategies, in particular of the cyclic ones, in different settings, the choice of which vertex visiting next is performed through two different approches: the *conscious reactive* one, in which next vertex in the plan is the one with maximum idleness among each robot local neighbours, and the *cognitive coordinated* one, in which next vertex is the one having highest idleness in all the graph.

In [9] the objective is guaranteeing that each target is visited at the same optimal frequency. This is once again obtained following minimal costs cyclic paths. Spanning trees are used to generate the cyclic paths in uniform grid-based terrains.

Also in this case robots need to move along the path in equidistant relative positions. The limitations they are subjected to are mainly velocity limitations while the important constraint on the whole strategy is that each point must be visited at the same frequency. There is no need to communicate for the robots in this work as well. A cost is associated with each point and direction of the terrain, making the grid directionally non-uniform. The solution is then based on minimal Hamiltonian cycles that guarantee maximal uniform frequency in the cycle. The main feature of this strategy is that it is robust to the number of robots following it: as long as there is at least one robot working properly, uniform frequency of the patrolling task is will still be achieved.

Another interesting work is [10]. Here the objective is to periodically visit a discrete set of predefined sites, with varying priority or visit frequency guarantees. These periodic visits are planned in a flexible and robust way in order to face losses of robots or battery problems. The framework for this problem is the one of the Routing Problem with Time Windows, a variant of the Vehicle Routing Problem in which multiple robots need to visit all vertices in a graph with the constraint that each vertex must be visited within a certain time window. Like in [2], the edges of the graph specify the traversal time from one vertex to another and define the basis of potential routes, but in this case the graph is directed. In this case too, as in [9], communication is not needed. Introducing the time windows in the graph is done in steps: each site priority level, possibly being high, low, or medium, induces a required period between site visits. This is implemented as a time window constraint on the visit of each node. The deadlines for reaching a site are modified based on the time since the site was last visited. Finally, in order to incorporate longer patrol horizons, copies of the nodes with offset time window constraints corresponding to their required visit periods are introduced in the graph. As multiple visits to the same vertex are allowed and inevitable over long periods of time, the problem may grow very quickly and global optimality is given up in favour of the local optimization of a series of more tractable problems by introducing a receding-horizon framework. Once this variation of the routing problem is considered, the result is a set of routes at minimal and equal costs covering all vertices: this means that what is actually achieved is a set of Pareto-optimal solutions with increasing costs and decreasing times.

A non linear extension of the orienteering problem is developed in [11], which is then used to plan informative paths for persistent monitoring of the environment. The paths used are, once again, cyclic as robots must leave and return to a fixed base station. Communication is not considered: the constraints on the strategy are a limited time budget to complete the task and the need to have tours of a predefined length. The objective is to make each robot follow the planned path whose total cost must not exceed the given travel budget. Robots must visit as many vertices

as they can to maximize the amount of collected information, measured through a reward function. Like in [2] and [10], a graph is used to model the environment. Robots can move between any two vertices with a cost proportional to the distance between them.

Another graph-based work that is important for this thesis is [8], here a fleet of unmanned aerial vehicles (UAVs) must accomplish the task of cyclically patrolling an environment represented as a unidirect graph in which vertices are locations of the environment and edges represents their physical connections. In this model, robots move on the graph by covering the edges. The interesting difference of this work, with respect to previous ones, is that communication plays an importat role. A communication infrastructure does exist, providing only some regions of the environment with a communication link to a mission control center. The vertices of the graph are divided into two groups: *m*-type vertices, that robots need to monitor and report, and *c*-type vertices, from which robots can communicate with the mission control center. The main goal is to compute a joint patrolling strategy that minimizes the communication latencies, defined as the delays between the inspection of a location of interest and its report to the mission control center, taking the average latency of an *m*-type vertex as the performance metric for a tour. The constraint each joint plan must satisfy is a time constraint, like in [11]: a walk on a graph assigned to a robot must be completed not exceeding the given time budget. Two different strategies have been used to solve this problem: a mixed integer linear programming approach, and an heuristic algorithm. The latter tries to satisfy coverage by building an initial budget-compliant walk on the graph for each robot. Then, if an initial feasible solution is found, local moves are applied to lower down the value of the objective function. The results obtained through this work show that, in the case of the mixed integer linear programming, an optimal fomulation is possible but doesn't scale properly. Ont the other hand, the heuristic approach is more scalable and it can be seen that robots do not visit many *m*-type vertices withouth first passing in a *c*-type vertex and not all *c*-type vertices are visited, being too far away from the targets. Moreover, it can be noticed that increasing the time budget leads to an improvement of the communication latency.

All the papers discussed so far aim at finding the best strategy in non-adversarial settings, since the patrolling task is carried out to monitor target vertices without adversarial entities to be detected and neutralized. Although this thesis does not consider it, there are important works on adversarial settings that need to be cited.

In [12], a reactive motion plannig strategy is proposed to maximize the area covered by sensors mounted on each UAV, to provide persistent surveillance, to maintain high sensor data quality, and to reduce detection risk. This risk can be modeled as a costmap over a grid representation of the ground level. UAVs need to fly at higher altitudes to reduce the probability of being detected over areas with

high ground-level risk. Persistency of surveillance is achieved by giving priority to the regions displaying the largest wait time, defined as the elapsed time since the region was last surveyed. Scalability is obtained by avoiding explicit coordination among the UAVs. Communication is not modeled and is needed only between UAVs and the central planner. The cost function used used to build the costmap is based on an uncertainty measure that the planner keeps updated. This uncertainty embeds the regions that have been surveyed and the time in which they were surveyed; the longer a region is not visited, the most this uncertainty increases. It has been observed that the planner adjusts the altitude of the UAVs only in a second stage, after the persistent area coverage has been carried out, to minimize the risk while maximizing the sensor data quality.

Another interesting result can be found in [13], where UAVs are used to localize mission-related enemies. The innovation of this work lies in the representation of the environment: probabilistic quadtrees extended by adding stochastic arrivals of intruders are used instead of uniform grids. The results obtained confirm the usefulness of this approach. When an enemy enters the environment, the patroller gets a penalty that linearly grows over time and is proportional to the importance of the area the intruder broke into. In order to stop paying the penalty, the patroller needs to find out where the intruder is and eliminate it. Arrivals of intruders are not immediately detected but need to be searched by the patroller, based on the prior knowledge on the probability that intruders are located inside each cell. The objective is to minimize the penalty occurred over time by minimizing the waiting time, defined as the interval of time between the moment in which an intruder breaks in and the moment in which it is detected and removed. In order to do so, each robot must focus on areas in which an attack is more likely to happen. The strategy used by the patroller equally splits the available time among sweeping, a quick gross-grain reconnaissance of the environment to identify regions that are likely to have been attacked, and searching, the actual elimination of intruders, using weighted density. The result is that the sweeping part is done relatively quickly and no intruders are left in the area.

A more game theoretic approach can be found in [14]. This approach allows for the development of patrolling strategies in which the possible actions of the intruder are taken into consideration when deciding where the robot should move. Usually, the intruder can hide and observe the action of the robot before breaking into a location. This model leads to to the adoption of a leader-follower solution concept, where the leader decides to follow a strategy, and the follower acts as a best responder, given the chosen strategy. In the context of this work, the leader is the patroller and the follower is the intruder. Even though this work is mainly theoretical and refers to a single robot performing the patrolling task, it is interesting as it presents an approach to compute optimal leader-follower strategies in

environments with arbitrary topologies. The problem of findind a leader-follower equilibrium is reduced to a mathematical programming problem, and the final result this work shows is that the patroller strategy, found at such equilibrium, is the optimal strategy for the mobile robot.

2.2 Distributed patrolling strategies

This second family of stratefies eliminates the need of the central coordinator in deciding the plans the robots have to follow: each robot computes its own plan to fulfill the task.

In [4], the main duty of each robot is to observe some features of the environment by means of on-board sensors. The environment is represented through a patrol graph, with vertices being the different poses (considering both position and orientation) of targets in the environment, and a cost function denoting the expected travel time for each edge, according to the capabilities of the robots. The cost of a path is given by the sum of the costs of each edge like in [2] and [10]. It is interesting to notice the difference between this type of graph and the commonly used topological graph: in a topological graph vertices are just locations, while in the patrol graph they are poses. Moreover, vertices in the patrol graph are a subset of the map topological vertices, which means that each patrol vertex must be reached through the map topological graph; and edges in the patrol graph are mapped to paths in the map topological graph. Since cyclic paths are used, as the patrol graph is fully-connected, the main constraint on the plan is that, like in [2] and [9], two subsequent robots must keep a constant distance between them. The performance of the system is measured using the concept of vertex idleness as described in [2], measuring both the average idleness and its standard deviation since the actin execution is non-deterministic. Contrarily to [2], interaction between robots is now considered as it may have a significant impact on the navigation performances and on the mission objective. The plans synchronization mechanism is based on communication between two consecutive robots in order to maintain the minimum distance: each robot sends to the following robot the time it has reached each patrol vertex, and the latter robot decides to stay on its current position if the time it expects to spend for reaching the next patrol vertex causes the idleness of that vertex to go below the desired average idleness. In this way, the bandwidth requirements of the communication remain very low. To prove the validity of the system, both linear environments (not having loops in the topological map but allowing the patrol graph to be cyclic), and cyclic environments are taken into consideration. Moreover, as already done in [2], a comparison with the partition strategy is made. In both cases, the cyclic strategy is proven to be more

effective and robust to robot failures; furthermore, it guarantees no conflicts among robots in cyclic environments. In general the whole system is proven to be robust to small navigation problems and to robots with different navigation abilities.

In [5] an experimental comparison between different distributed patrolling strategies is presented. An evaluation of five representative patrol approaches is done in terms of their performance, the behavior resulting from interactions between teammates, and team scalability in different environments. The metric used to compare the performance is the average idleness of the graph representing the area to patrol. The results in this work help to identify which strategies enable enhanced team scalability and which are the most suitable approaches given any environment.

A completely different approach is used in [3], where the patrolling task is modeled as a reinforcement learning problem. This model is adopted with the aim of allowing automatic adaptation of the robots to the environment. The environment is represented through a graph where vertices are locations, and edges encode possible paths between vertices. The patrolling problem is modeled through a Semi-Markov Decision Process, a general version of the classical Markov Decision Problem, where actions can take a variable amount of time. Each robot has a certain probability of choosing one action, with which it can traverse one edge on the terrain abstraction. The only feedback that a robot receives is the idleness of the vertex currently being visited. The final goal is to minimize the average idleness of the graph by maximizing the idleness values of the visited vertices. In fact, if the robot's reward is the idleness of the vertices, and it acquires it only when actually visiting them, it will try to visit the vertices with the highest idleness in order to decrease the global one by aximizing its own reward. As far as communication is concerned, more than one possibilities were implemented and compared, not only between themselves, but also with a centralized architecture with no communication. The three explored schemes are: a black box system, in which robots do not know about the actions of other robots, but use a simple flag communication scheme, they leave a flag in every node they visit that contains the number of cycles they have been in that node; a white box system, in which each robot knows about the actions undertaken by all other robots; and a grey box system, in which robots can communicate their future actions or their intents only to robots that are near the current vertex. In the context of the paper teams of robot exploiting the white box system are proven to be superior than any other compared to them.

In [15] a team of mobile robots must cover an environment in which communication is restricted to a set of locations with the constraint of gaining connectivity at fixed intervals. The objective is to prove that in robotic information gathering domains, breaking connectivity can lead to paths with higher information quality. The environment is known and is discretized into a graph with vertices representing convex regions and edges representing the possibility of communicating between

such regions. In this model there is a time budget costraint, since the paths robots have to plan cannot be longer than a given time horizon. The algorithm proposed to fulfill the task is " Implicit Coordination". Robots plan their paths in turns and each robot only considers changing its own path while planning, keeping shared plans and/or state information for the team. Each robot maximizes its reward over the given time horizon, equal to the disconnection interval, and chooses a path that is connected to the planned configuration of its teammates at the end of the interval. The algorithm is re-run every fixed periodic time interval, as in a receeding-horizon framework, and successfully terminates as long as the robots have up-to-date information from the rest of the team. The system is tested for a team of robots that have to search for stationary targets and also in a non-adversarial search domain in which robots have to find a mobile target. In particular, in this last domain it is proven that the algorithm leads to better capture times with respect to continuous connectivity. The negative aspect of periodic connectivity is that, since robots are not always able to communicate, they must plan synchronously when they are connected to guarantee that each of them has the newest information available. In addition, the algorithm assumes that the time in which all robots are connected can be predicted. This leads to the need for robots to either wait for their teammates or set a specific time-out to continue planning without the entire team in case one or more robots take additional time to reach the connected configuration.

All the studies described so far provide a solid background to work on and have been of inspiration for this thesis. As in most of the works, we adopt a graph-based representation of the environment. In particular, we add the differentiation of vertices found in [8] in order to implement the same kind of communication architecture to our model. The performance metric we are using is the vertex idleness defined in [2] and used in most of the above mentioned strategies. Of course we had to adapt it to our communication-restricted patrolling framework. Also the algorithms we implement to fulfill the patrolling task, that are mostly influenced by [15] and [5], had to be rearrenged to fit in our framework.

Works	Centr./ Distri.	Comm.	Discr.	Obj.	Strategies	Advers.
[2]	Centralized	Not modeled	Topological, unidirected graph	Minimizing worst-case idleness	Cyclic strategy/ partitioning strategy	No
[9]	Centralized	Not needed	Uniform grid	Maximizing uniform visit frequency	Spanning trees to compute cyclic paths	No
[10]	Centralized	Not needed	Topological, directed graph	Periodically visit a discrete set of predefined sites	Three different algorithms compared	No
[11]	Centralized	Not modeled	Topological, directed graph	Maximizing a reward	Cyclic strategy	No
[8]	Centralized	Possible only in some regions of the environment	Topological, unidirected graph	Minimizing communication latencies	Integer linear programming /heuristic algorithm	No
[12]	Centralized	Not modeled	Grid	Minimizing the uncertainty on sites	Reactive strategy	Yes
[13]	Centralized	Not needed	Probabilistic quadtrees	Minimizing the penalty over time	Equally splitting sweeping and searching	Yes
[14]	Centralized	Not needed	Arbitrary	Eliminate the intruder	Leader-follower strategy	Yes
[4]	Distributed	Only between two consecutive robots	Patrol graph	Minimizing average idleness	Cyclic strategy	No
[5]	Distributed	Not modeled	Topological graph	Minimizing average idleness	Different algorithms compared	No
[3]	Distributed	Black box, white box, and gray box approaches compared	Topological graph	Maximizing a reward	Semi-Markov Decision Process	No
[15]	Distributed	Restricted to a set of locations	Topological, directed graph	Maximizing a reward	Implicit Coordination	No

Figure 2.1: Summary of the different features.

Chapter 3

Problem setting

The objecive of this thesis is to propose a model for an innovative communication-restricted patrolling framework, where cooperation between robots is essential, and to compare different strategies used to fulfill the patrolling task. In order to do so, we need to specify a model for the mission in which robots operate, taking inspiration from the works described in Chapter 2, and rearrange the performance metric (the *idleness*) used to investigate the strategies that solve the patrolling task.

This chapter deals with the modelling of the framework and the redefinition of the *idleness*, with a particular focus on the features proper of this new setting.

3.1 Environment definition

A team $R = \{1, \ldots, m\}$ of robots must persistently patrol some given locations of an environment while making reports to, and exploiting information provided by, a base station BS. Robots can communicate with the BS by exchanging messages through a pre-existing communication infrastructure, which is only accessible from some regions of the environment. To formalize this setting, we represent the environment (as customarily done in the literature) by means of an undirected graph $G = (V, E)$ in which vertices represent all the locations robots can visit, while edges encode physical connections between them (without considering self-loops). Each edge $(i, j) \in E$ is associated with a positive number $d(i, j)$ representing the physical distance required for moving between i and j in the environment: as typically done in robotics applications, we assume that distances satisfy the triangle inequality. Vertices V are used to represent both locations to patrol and places providing the possibility of exchanging messages with the BS. Accordingly, following the notation introduced in [8], each vertex in V is preassigned to up to two types: m, if the vertex needs to be persistently monitored (patrolled), and c, if the vertex allows to exchange messages with the BS. We denote with $V_m, V_c \subseteq V$ the two subsets

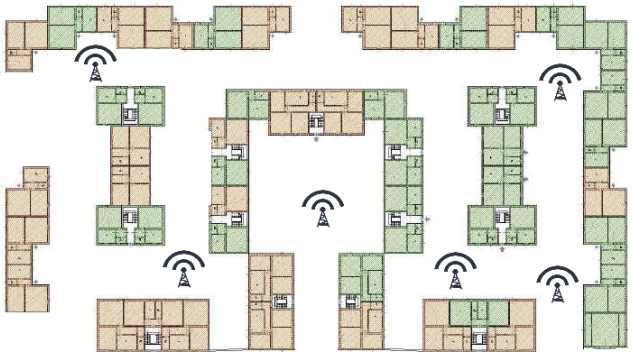


Figure 3.1: A possible communication-restricted environment in which robots have to solve the patrolling task.

of m-type and c-type vertices, respectively. (Notice that V_m and V_c may share a non-empty intersection). Robots move on the graph G by traversing the physical edges E and detect the vertex their in: once they reach a m-type vertex, they accomplish a patrolling visit on it; once they reach a c-type vertex, they are allowed to communicate with the BS. The only limit robots have to face while patrolling the environment is that they cannot move at a higher speed than the one given.

A realistic scenario in which robots have to face with such a communication-restricted environment is the one shown in Figure 3.1, in which robots must patrol the perimeter of a factory that extends in a large area in which Wi-Fi coverage is not always granted. In the example, the area of the factory are colored in green if administrative offices are present in the building, and we assume that only in those offices strong Wi-Fi coverage is granted. Therefore, also for a team of robots patrolling such environment, communication is possible only near green buildings.

3.2 Performance metric definition

In order to assess the quality of the different patrolling strategies, we now generalize classical evaluation metrics related to the notion of *idleness* [2] in the light of our extended communication-restricted framework. We recall that the *idleness* is defined as the time elapsed between two consecutive visits of a vertex. The persistent patrolling task unfolds across an arbitrary long time horizon T in which time evolves in discrete steps $\{1,\dots,T\}$. The patrolling tour undertaken by a robot r is represented by a *walk* on the graph G of the form $w_r = (v_0, v_1, \dots, v_k)$, where

v_0 denotes the robot's starting vertex, v_k denotes the last vertex visited before T, and a generic v_i denotes the i-th vertex visited by the robot. (Notice that, being w_r a walk on G, the same vertex can appear multiple times in w_r). For each walk w_r, we define a function $t_r : \{1,\dots,k\} \to \{1,\dots,T\}$ mapping the i-th visited vertex by robot r to the time step in which the visit takes place. In particular, each $t_r(i)$ is determined by the total distance traveled until the i-th vertex visited in w_r ($\sum_{j=0}^{i-1} d(j, j+1)$) and by robot r's (possibly varying) speed along that portion of the walk. Now, for any $i \in \{1,\dots,k\}$ such that $v_i \in V_m$, let $v_{\bar{c}}, \bar{c} \geq i$ be the first vertex after the i-th vertex visited s.t. $v_{\bar{c}} \in V_c$, and let $c(i) = \bar{c}$. Following this notation, we define robot r's *reporting time* of visit i to the BS (only for vertices v_i s.t. $v_i \in V_m$) as $t_r(c(i))$, assuming that this quantity will be infinite if the visit is never reported before the patrolling task ends at T. (Notice that $t_r(c(i)) = t_r(i)$ iff $v_i \in V_c \cap V_m$).

It may be possible, in our framework, that the BS receives reports containing outdated information. This happens when there exist two robots $r, r' \in R$ (not necessarily different), two walks $w_r = (v_0, v_1, \dots, v_k), w_{r'} = (v_0, v_1, \dots, v_{k'})$, and two visits $i \in \{1,\dots,k\}, j \in \{1,\dots,k'\}$ s.t. $v_i = v_j$, $t_r(i) < t_{r'}(j)$, and $t_r(c(i)) \geq t_{r'}(c(j))$. In such a case, we say that the i-th visit of robot r is *outdated*, in the sense that the corresponding report to the BS will contain outdated information. It must be noted that vertices belonging to both V_m and V_c can never generate an outdated visit. We are now ready for introducing metrics related to *communication idleness*.

We define the *instantaneous communication idleness* of vertex $v \in V_m$ at time $\bar{t} \in \{1,\dots,T\}$ as:

$$IC_v(\bar{t}) = \bar{t} - \bar{t'},$$

where $\bar{t'} \leq \bar{t}$ denotes the timestep of the last not outdated report of vertex v to the BS not after timestep $\bar{t}$. We assume that the initial situation of all the vertices to be monitored is known, and hence that $IC_v(\bar{t}) = \bar{t}$ if there exists no visit to v that has been communicated (by any robot) at time $\bar{t}$. The *instantaneous graph communication idleness* at time $\bar{t}$ is defined as the *average instantaneous communication idleness* among the vertices to be patrolled:

$$IC_G(\bar{t}) = \frac{\sum_{v \in V_m} IC_v(\bar{t})}{|V_m|}.$$

In this thesis, we are mainly interested in studying patrolling strategies allowing to minimize the *average graph communication idleness*, formally defined as:

$$\overline{IC}_G = \frac{\sum_{\bar{t} \in \{1,\dots,T\}} IC_G(\bar{t})}{T}.$$

Another interesting performance indicator we will investigate is the *worst-case graph communication idless*, WIC_G, which can be informally defined as the largest

instantaneous vertex communication idleness encountered through the patrolling task. Average and worst-case idleness for the single vertices can be defined in a similar way: given a vertex v_i its average vertex communication idleness is

$$\overline{IC}_(v_i) = \frac{\sum_{\bar{t}\in\{1,\dots,T\}} IC_(v_i)(\bar{t})}{T}$$

while its *worst-case idleness* is the largest instantaneous vertex communication idleness, related only to v_i, encountered through the patrolling task

The *instantaneous communication idlenesses* of the vertices are not computed by robots, but the BS is in charge of this calculation. This calculation is done only when a robot communicates with the BS, giving it updated information on each vertex it has visited from last communication. The main objective of the communication infrastructure, common to all strategies investigated, is to keep the values of these idlenesses always updated: every time a robot communicates the BS computes the new values of the *instantaneous communication idleness* for each vertex of the graph and send them back to the robot. In this way both the robots and the BS have updated information: robots will use it to carry on their strategy, the BS will use it to compute the *instantaneous graph idleness* and *the average graph idleness*.

With this redefinition of the notion of idleness and of the objective of the robots, another major difference between the works discusse din Chapter 2 and ours arises: while, most of the time, in those works robots head to those vertices not recently visited to solve the patrolling task, in our new framework robots move in order to reach those vertices on which we do not have recent information. It is interesting to notice that not always this two kinds of vertices coincide.

Lastly, we point out that, according to our problem definition in which robots move along walks on G, we are not considering the possibility that a robot remains in its current vertex for more than a single time step, since walks are computed on a graph in which the edge set represents physical connections between locations. This is done in order to avoid that, during the planning phase, some patrolling strategies might consider potentially attractive walks with low *average graph idleness*, but where robots prefer to remain still across the whole patrolling task. Moreover, remaining stuck at a given vertex would be pointless when the minimization of *worst-case communication idleness* as a secondary objective is also taken into account.

Chapter 4

Patrolling strategies

In this chapter we present the three different multirobot patrolling strategies that we designed for the communication-restricted setting introduced in Chapter 3 (plus a random baseline one), with the objective to minimize the *average graph communication idleness* $\overline{IC}_G$. Our main goal is to implement and study distributed strategies, but we also present an optimal centralized strategy in order to compare it with them.

In Section 4.1, we present a strategy in which the BS performs the planning phase by computing centralized optimal plans for all the robots. Then, to overcome its unavoidable scalability issues, we present two distributed strategies in which robots can query the BS about the current situation of the environment in terms of vertices idlenesses and teammates future intentions. Such distributed strategies can be thought as exploiting the available communication infrastructure to access a *blackboard* [16], residing at the BS, where agents share and update their knowledge about the unfolding of the mission. The usage of such an approach, firstly investigated in the context of classical multirobot patrolling [1], is particularly attractive in our framework, since it allows the robots to coordinate without any assumption about local transmission capabilities between two robots and about the structure of the environment to monitor (except for the presence of communication zones). However, as we show in Chapter 6, robots still need to rely on local message exchange in order to efficiently avoid each other while pursuing their own plans. The strategy described in Section 4.2 is the most similar to the centralized one. In Section 4.3, instead, we propose an alternative strategy, requiring lower computational effort and, yet, solid good performances.

4.1 Globally optimal strategy

The *globally optimal strategy* (G-OPT) is an optimal, centralized, and offline[1] strategy working as follows. At the beginning of the task, the BS computes the best set of joint robot walks $\mathbf{w}^* = (w_1^*, w_2^*, \ldots, w_m^*)$ spanning the whole task horizon T and minimizing $\overline{IC}_G$; then, it communicates the plans to the robots, which can then start to execute them. Robots may start either in a vertex $v_0 \in V_c$ or in a vertex $v_0 \in V_m$ with the difference that, when starting in a m-type vertex, the first action robots must perform is moving to a c-type vertex to receive the plan the BS has previously computed for them. Each robot receives only its own plan. In this strategy a robot knows nothing about the future vertices his temmates are going to visit and nothing about the *instantaneous communication idlenesses* of the vertices composing the graph.

As far as the computational complexity of such a planning scheme is concerned, we can notice that the evaluation of $\overline{IC}_G$ for a candidate set of joint robots' walks requires quasi-polynomial time ($O(mT)$), while the number of joint walks of length not exceeding T on G grows exponentially with the size of the input: therefore, this strategy may be suitable only for a limited number of small problem settings. Moreover, it is evident that the walks obtained by adopting this planning approach will be actually optimal with respect to the minimization of $\overline{IC}_G$ if the errors in the predictions of their execution can be safely assumed to be negligible. The pseudo-code of the planning algorithm of G-OPT is shown in Algorithm 1.

Algorithm 1 G-OPT planning algorithm

```
W ← {enumerate all joint walks of m robots on G with maximum length T}
for all w ∈ W do
    simulate the concurrent execution of w and compute the corresponding IC̄_G
end for
w* ← {the best w obtained that minimizes IC̄_G}
return w*
```

In Figure4.1, we present through a UML sequence diagram the messages that the robots and the BS need to exchange at the beginning of the mission to implement this strategy.

[1] In an offline strategy the whole problem data is known from the beginning, and the output computed when the mission starts is the solution to the entire problem.

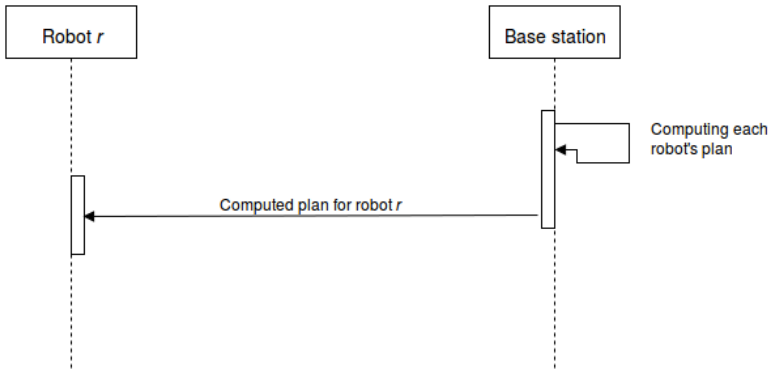


Figure 4.1: The sequence of messages exchanged at the beginning of the mission.

4.2 Individually optimal strategy

As the name of this strategy suggests, the *individually optimal strategy* (I-OPT) can be thought as the distributed and online version of G-OPT. When adopting this strategy, robots iteratively compute new portions of their own walk, $\mathbf{w} = (w_1, w_2, \ldots, w_m)$, as soon as they reach a new c-type vertex. As in the G-OPT strategy, robots may start the mission in either a c-type vertex or in a m-type. When starting in a m-type vertex, the first action robots must perform is moving to a c-type vertex to receive the information they need to compute their plan. Every time a robot reaches a c-type vertex, it queries the BS about (a) the current *instantaneous vertex communication idlenesses*, and (b) their teammates intentions, expressed in terms of portions of the walks they are currently following. Knowing the current *instantaneous communication idlenesses* of m-type vertices allows the robot to understand on which vertices the BS has not received updated information, while knowing the future intentions of its teammeates means to be aware of which vertices will be visited and reported to the BS in the near future. More specifically, let $\bar{t}$ be a generic timestep in which robot r reaches a c-type vertex, and let $\hat{w}_1, \ldots, \hat{w}_{r-1}, \hat{w}_{r+1}, \ldots, \hat{w}_m$ the portions of walks its teammates are going to followed from step $\bar{t}$. Robot r computes a new portion of walk $\hat{w}_r$ on G of estimated travel time H (the planning horizon) as the walk minimizing the average graph communication idleness between $\bar{t}$ and $\bar{t}+H$, while taking into account the walks followed by its teammates. Notice that the portions of walks followed by the teammates may have different estimated travel times, as they depend on the previous arrivals of the robots to c-type vertices. This means that, in order to be able to simulate in a truthful way the concurrent execution of its possible plans and of the walks of each teammate between $\bar{t}$ and $\bar{t}+H$, robot r should understand where each teammate is (in terms of which vertex the teammate is visiting while it

is computing its own best plan). If a teammate has not communicated a plan valid until $\bar{t}+H$, it is assumed to remain still at the last vertex of the communicated plan. A similar assumption is made in the following situation: if a teammate is supposed to reach a c-type vertex before time $\bar{t}+H$, it is assumed to be still in that vertex for the remaining travel time. If H is sufficiently small, a complete evaluation of all the possible walks of length H can for robot r be completed in reasonable time. Otherwise, a sampling strategy needs to be adopted. This planning scheme, whose pseudo-code is reported in Algorithm 2, is very similar to the *Implicit Coordination algorithm* of Hollinger and Singh [15]. The main difference is that, following I-OPT strategy and thanks to the *blackboard* mechanism, robots do not need to wait for the whole team to be simultaneously in a c-type vertex.

Algorithm 2 I-OPT planning algorithm for robot r at time $\bar{t}$

$\hat{w}_1,\ldots,\hat{w}_{r-1},\hat{w}_{r+1},\ldots,\hat{w}_m \leftarrow \{$query the BS about the walks followed by others$\}$
query the BS about instantaneous communication idlenesses of vertices V_m
$W_r \leftarrow \{$enumerate/sample walks from r's current position along $[\bar{t},\bar{t}+H]\}$
for all $w_r \in W_r$ **do**
  $\hat{\mathbf{w}} \leftarrow (\hat{w}_1,\ldots,\hat{w}_{r-1},w_r,\hat{w}_{r+1},\ldots,\hat{w}_m)$
  simulate the execution of $\hat{\mathbf{w}}$ and compute the corresponding $\overline{IC}_G$ in $[\bar{t},\bar{t}+H]$
end for
$\hat{w}_r \leftarrow \{$the best w_r obtained that minimizes $\overline{IC}_G$ in $[\bar{t},\bar{t}+H]\}$
return $\hat{w}_r$

In Figure4.2, we present, through a UML sequence diagram, the messages that the robots and the BS need to exchange every time a robot arrives to a c-type vertex in order to implement this strategy.

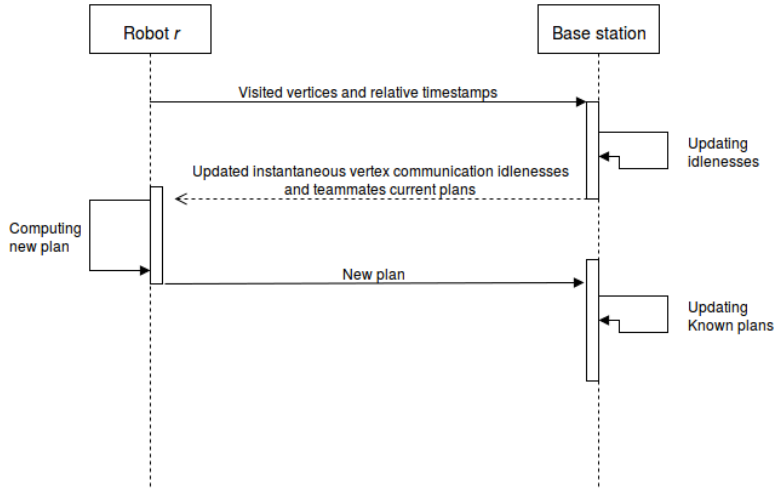


Figure 4.2: The sequence of messages exchanged in every c-type vertex.

4.3 Reactive strategy

The *reactive strategy* (RE) we present is still distributed, but does not involve long-term planning as the I-OPT one. instead, it is inspired by a classical reactive strategy empirically showing good performances when considering the classical notion of idleness (*the Cognitive Coordinated strategy* studied in [1, 2]). By adopting the RE strategy, robots iteratively choose a new m-type vertex to reach according to a heuristic in which vertices with the highest instantaneous communication latency have higher priority, and, in case the chosen vertex does not belong also to V_c, subsequently move to the closest c-type vertex in order to obtain from the BS the data needed for computing a new plan. As before, when the mission begins robots can be in either a c-type vertex or a m-type vertex. If a robot starts the patrolling task in a m-type vertex it moves towards the closest c-type vertex to receive the information it needs to compute the plan (as it happens in G-OPT and I-OPT strategies). When a robot r arrives at a c-type vertex at a generic time step $\bar{t}$, it queries the BS about (a) the *instantaneous communication idleness* of all the m-type vertices at that time instant and (b) its teammates intentions, once again expressed in terms of portions of the walks they are currently following. It then selects to reach the m-type vertex $\hat{v}$ with the highest *instantaneous communication idleness* not in the portion of walk followed by the other robots (in case there are multiple vertices with the same idleness, the robot chooses one at random). The path to $\hat{v}$ is the shortest path connecting the current position of the robot and $\hat{v}$, computed on the entire graph G. As a side-effect of this choice, additional m-type vertices can be visited. In case $\hat{v} \notin V_c$, the path is augmented with the shortest path connecting $\hat{v}$ to the closest c-type vertex. Notice that, if a robot reaches a c-type vertex before reaching $\hat{v}$ it communicates to the BS the list of vertices, and their relative timestamps, visited until that moment but it does not receive any information from the BS and does not compute a new plan. This is done in order to avoid that robots continue to move in cycles, and it is also the reason why, if $\hat{v}$ is not a c-type vertex, they need the additional path so they can reach a c-type vertex, receive updated information, and compute the new plan.The pseudo-code each robot runs for implementing this strategy is shown in Algorithm 3.

In Chapter 6, we will investigate the behavior of these strategies, in representative environments, by also comparing them against a baseline random strategy in which each robot chooses randomly a new vertex of G to reach, regardless of its current idleness and membership to V_m and/or V_c.

Algorithm 3 RE planning algorithm for robot r at time $\bar{t}$

$\hat{w}_1, \ldots, \hat{w}_{r-1}, \hat{w}_{r+1}, \ldots, \hat{w}_m \leftarrow \{$query the BS about the walks followed by others$\}$
query the BS about instantaneous communication idlenesses of vertices V_m
$\hat{v} \leftarrow \arg\max_{v \in V_m, v \notin \hat{w}_i, i \in R \setminus \{r\}} IC_v(\bar{t})$
compute $\hat{w}_r$ as the path on G from r's current position to $\hat{v}$
if $\hat{v} \notin V_c$ **then**
    let $\hat{v}_c$ be the c-type vertex closest to $\hat{v}$
    augment $\hat{w}_r$ with the shortest path on G from $\hat{v}$ to $\hat{v}_c$
end if
return $\hat{w}_r$

Chapter 5

System architecture

This chapter presents the software architecture designed to test our communication-aware framework and the patrolling strategies described in Chapter 4.

In Section 5.1 we give an overwiev of the main characteristics of the Robot Operating System (ROS), on which we build and test our framework. In Section 5.2 we present how we decided to implement the two entities participating in the patrolling task the robots and the BS and and their coordination. Since the experiments are carried out through simulations, not having the possibility of testing the system on real robots, a Communication Server (CS) is implemented in order to take care of the simulation of the communication channel by guaranteeing that messages are delivered at the right time to the rightful receiver.

5.1 ROS: Robot Operating System

ROS is a distributed, flexible framework for writing robot software that offers a message exchanging interface providing inter-process communication, behaving like a middleware. As described in [17], the main components of the ROS middleware are nodes, messages, topics, and services. Nodes are processes that perform computation. As ROS is designed to be modular, a system is typically composed by several nodes. Nodes communicate with each other by passing messages in two different ways: through topics or through services. In both cases, the communication pattern used by ROS is a publish-subscribe one. The difference between the two ways of handling the message exchange is the following: when dealing with topics, a node sends a message by publishing it to the topic of interest, while a node that is interested in receiving the same kind of data subscribes to the same topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. On the contrary, services are defined by a pair of strictly typed messagges, one for the request and

one for the answer, and only one node can advertise a service. The introduction of the service mechanism overcomes the problem of synchronous transactions, not taken care by the publish-subscribe paradigm. The consequence of this architecture is that, once a node subscribes to a topic, it sees all the messages arriving on that topic, also those not directly meant for it. For instance, while simulating multirobot systems on a single computer without an ad hoc communication infrastructure, a robot can potentially "cheat", by subscribing to topics used by other robots too to send or receive messages. This allows the robot to always have a complete picture of what is going on during the mission, making the coordination problem meaningless and the simulation far from facing the issues specific of multirobot systems. In order for ROS nodes to be able to communicate, a ROS network must have its *roscore* running. The roscore is a collection of nodes and programs that are prerequisites of a ROS-based system, it includes a *ROS master*, that provides naming and registration services to the rest of the nodes and allows them to locate each other, a *ROS parameter server*, initialized by the master and that nodes used to retrieve and store parameter values, and a *rosout* logging node.

Although ROS is widely used in robotics, its involvement in multirobot systems is relatively recent. The publish-subscribe paradigm is not well suited for this kind of systems and there are features that are proper of multirobot systems that still need to be better implemented in ROS. There are a lot of examples of works done in order to be able to implement a solid multirobot system using ROS, one of them being [18], where concepts, problems, and possible solutions for ROS multi master systems are presented. These systems are built from two or more ROS networks, each with its own roscore. The idea of [18] is to implement multirobot systems by assigning to each robot its own ROS sub-system. In this configuration, two types of messages are present, one for the infra-robot communications, and one for inter-robot communications. In both cases, the way communication and message exchange is handled is exactly like the one described above, according to the publish-subscribe paradigm. The implementation of this system is done through *multimaster fkie* framework. Here, two different types of ROS master exist:(a) the *master discovery*, that detects other ROS masters, makes them aware of its presence, and keeps track of any changes occurring in the different sub-systems, (b) the *master sync*, which updates topics and services with the information provided by the *master discovery*. This configuration is valid when there is the possibility of having multiple physical robots with independent processors. Unfortunately, there are still problems when dealing with one computer running simulations of a multirobot system, since robots are not completely isolated and each one of them is still able to see everything that is happening during the mission.

An alternative way to deal with the problem is to use the *namespaces*. Namespaces an important mechanism in ROS for providing encapsulation. Each ROS

node is defined within a namespace, which it may share with many other nodes. In general, nodes can create resources (e.g messages or services) within their namespace and they can access resources within or above their own namespace. Connections can be made between resources and nodes in different namespaces. This encapsulation isolates different portions of the system from accidentally grabbing the wrong named resource.

5.2 Nodes implementation

As previously stated in Section 5.1, the ROS nodes composing our system are organized in robots and BS. Each of them performs different computations and has a different goal. We decided to implement a node for the BS and a node for each robot so as to have a controller for each of them composing the team, as we are dealing with multirobot systems. In this way each robot is completely independent from the others.

The novelty of our framework is the communication infrastructure that allows robots and BS to communicate in a smooth way. The publish-subscribe communication paradigm is not suited for multirobot systems since it does not discriminate senders and receivers but lets everyone see every message that is being exchanged. As we are mostly interested in simulation, we propose the introduction of a Communication Server CS, a node in charge of sorting the messages to the rightful receivers. The introduction of such a node can be avoided when dealing with multiple, physical robots, as all its functions are performed by the physical communication network. The CS, as a module in the robot nodes, will be in charge of transparently manage the communication between the communication module of the robots and the available network model (e.g., wifi, infrared, radio signals, etc). In Figure 5.1 the two different architectures are portrayed. The structure of the CS that we are using in our simulations, as the name suggests, is that of a server. More precisely, the CS is the only one advertises 4 different services:

- the "permission" service, whose clients are the robots of the team that call it in order to request the permission to communicate, once they are in a c-type vertex. If the permission is granted, the whole communication procedure starts;

- the "talking" service, whose clients are the robots. It is called by them after having received the permission to start the communication procedure, and through it they communicate the visited vertices and their relative timestamps to the BS;

- the "BS talking" service, whose client is the BS, is called to inform the robots

on the idlenesses of the nodes, and the plans of the robots that the BS knows at the time it calls the service. The "BS talking" service is the answer to the previously called, by the robots, "talking" service;

- the "new plan" service, whose clients are the robots, is called to communicate the newly computed plan to the BS. It is the last service invoqued during the communication procedure.

Consistently using the CS as the third party in the communication procedure guarantees that messages are never intercepted by robots that are not directly involved in the ongoing communication, as communication do not rely on topics in which robots and BS are all publishers and subscribers. Instead, it uses services advertised only by the CS. Notice that the CS always knows where robots are in the environment and where the c-type vertice are thanks to the simulator in which we run our framework. We use the stage Simulator [19] but the same considerations can be made for other existing simulators (e.g, Gazebo). The CS continuosly reads the positions of the robots by subscribing to the topic published by Stage, but it does not know about the robots intentions or about the values of the node idlenesses at any time. This is because its only objective is to carry out the communication procedure when requested by a robot and it doesn't need to save useless information for itself. The natural consequence is that, every time a robot needs updated information, the CS has to implement the whole procedure involving the BS too. Another important feature of the simulator is that it provides a global clock, in which time is expressed in seconds, which robots read to assign to the visited nodes the useful timestamps.

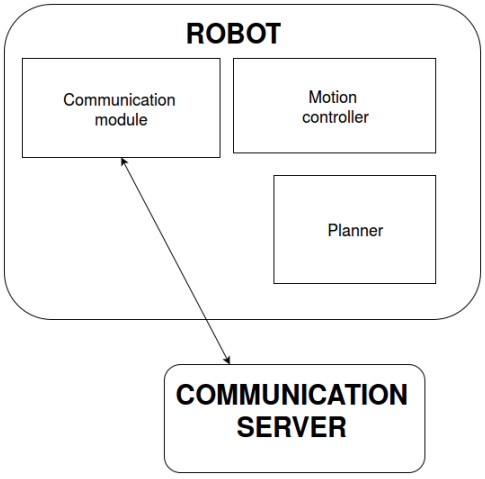
Robots are the nodes actully fulfilling the patrolling task. The different modules composing the robot nodes are (see Figure 5.1:

- the module handling the actual motion of the robots in the environment, better discussed in Chapter 6;

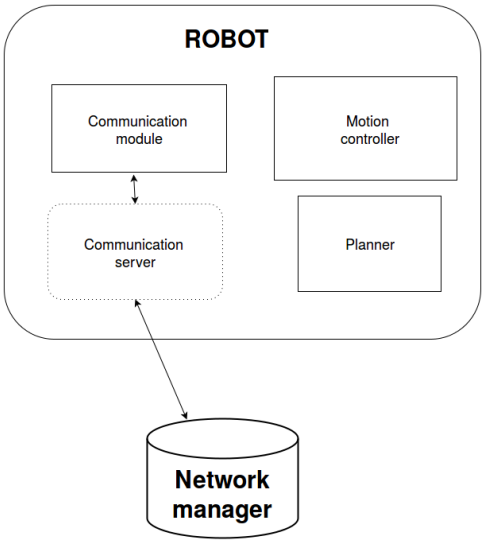
- the planner module, in charge of computing the optimal plans each time the robot receives updated information from the BS;

- the module supervising the communication with the CS.

The final objective of the planner is minimizing the average graph communication idleness $\overline{IC}_G$. Once again, the plans can be computed only in c-type vertices. Whenever a robot reaches a c-type vertex it immediately calls the the "permission" service, to be sure to be in a node that allows communication. As soon as the permission arrives from the CS it calls the "talking" service, with which it informs, through the CS, the BS on the vertices it has visited and the time it has visited

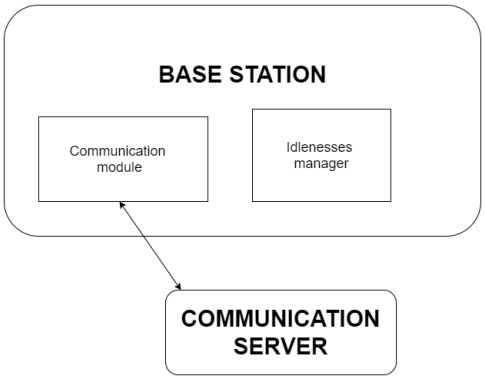


(a) Simulated robot



(b) Real robot

Figure 5.1: The two robot architectures. Here we show the different uses of the CS: in the simulated robot it is an external node, used as a means to communicate with the BS, in the real robot it becomes an internal module taking care of connecting the robot to the communication network.



(a) Base station

Figure 5.2: The base station architecture.

them. It then waits for the updated instantaneous node idlenesses and the plans of its teammates. Once the BS sends them to it, the robot computes its plan, following one of the strategies described in Chapter 4, and sends it to the BS through the "new plan" service. At this point the robot can start and execute the plan while the BS updates its knowledge on the current plan that the robot is following.

The last node composing our system is the BS. It is in charge of keeping track of the plans of the robots in the team, and of the idlenesses of the vertices, during the mission. Every time a robot reaches a c-type vertex and starts the communication procedure, the BS receives (from the CS) the list of vertices visited by the robot with their relative timestamps (one referring to the time the robot has visited the vertex, and the other one referring to the time at which it started the communication procedure). The BS checks if the information on the vertices contained in the message are outdated, as described in Section 3.1, and updates the instantaneous node idlenesses $IC_v(\bar{t})$, where $\bar{t}$ is the time in which the communication is happening, for all the m-type vertices of the graph. It has also to handle the calculation of the istantaneous graph communication idleness $IC_G(\bar{t})$, and of the average graph communication idleness $\overline{IC}_G$. Once it has finished to compute all the values of the idlenesses it calls the "BS talking" service, communicating to the robot that communicated to be in a c-type vertex the updated $IC_v(\bar{t})$, for all m-type vertices, and the plans of the other robots known at time $\bar{t}$. Finally, it waits for the last step of the procedure: receiving the future plan of the robot that has started the communication procedure. Each time the BS computes the idleness values, it writes the values in log files, recording how the situation is evolving. Overall, the modules that form the BS, shown in Figure 5.2, are the one in charge of managing the communication with the CS and performing the updates of the idlenesses.

In Figure 5.3, an example of a successful communication procedure is pre-

sented as a UML sequence diagram. Here the whole sequence of exchanged messages is described:

1. the robot that has arrived in a *c*-type vertex asks the CS for the permission to communicate;
2. the CS checks wether the robot is actually in a *c*-type vertex, and grants the permission to the robot;
3. the robot sends the list of the visited vertices and their relative timestamps to the CS;
4. the CS forwards it to the BS and informs the robot that the communication has been successful;
5. the BS updates the values of the instantaneous node communication idlenesses, of the instantaneous graph communication idleness, and of the average graph communication idleness. Once the udpating phase is completed, the BS sends the new instantaneous node communication idlenesses, and the plans of the other robots of the team that it knows, back to the CS;
6. the CS forwards the received information to the robot and informs the BS that the communication has been successful;
7. the robot computes its new plan, based on the information just received, and sends it to the CS;
8. the CS forwards the new plan to the BS and informs the robot that the communication has been successful;
9. the BS updates its knowledge on the plan that is being followed by the robot, while the robot starts to execute it.

Last picture, Figure 5.4, presents a case of unsuccessful communication procedure. Note that what is described can also happen at any of the steps in which the CS sends the message of "successful communication" to the robot (or to the BS). The sequence of exchanged messages in this cases is the following:

1. the robot that has arrived in a *c*-type vertex asks the CS for the permission to communicate;
2. the CS checks wether the robot is actually in a *c*-type vertex, it realizes that the *c*-type vertex the robot is in does not provide the communication link anymore, possibly because of some breakdowns of the network;
3. the robot receives the denial and looks for the closest *c*-type vertex where it can start the procedure again and moves towards it.

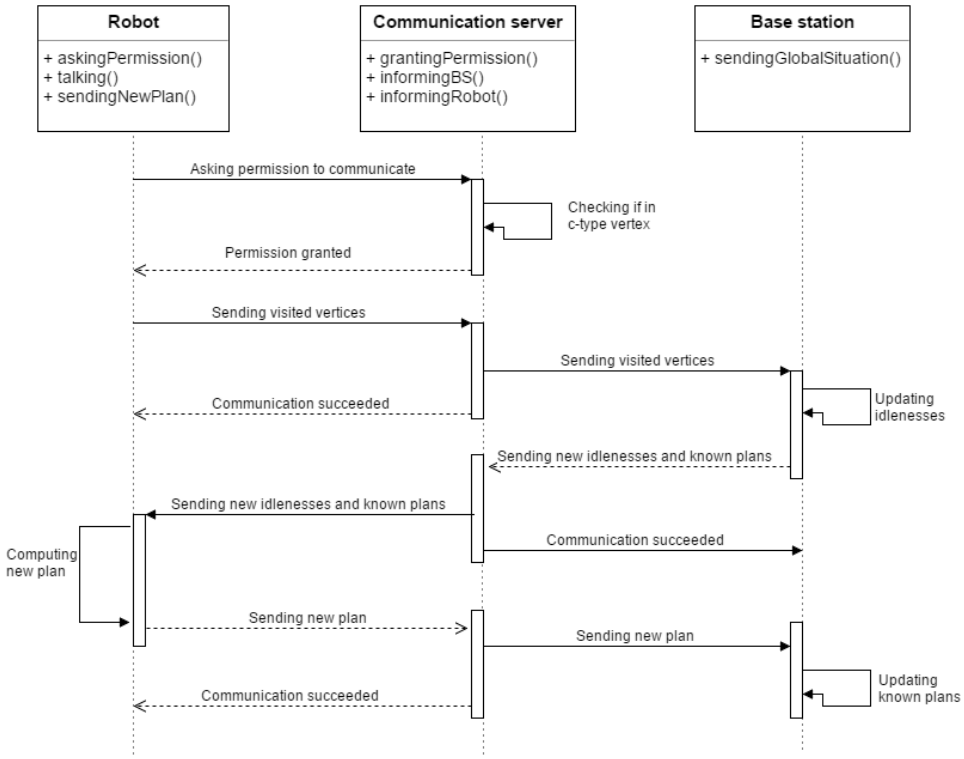


Figure 5.3: Successful communication procedure.

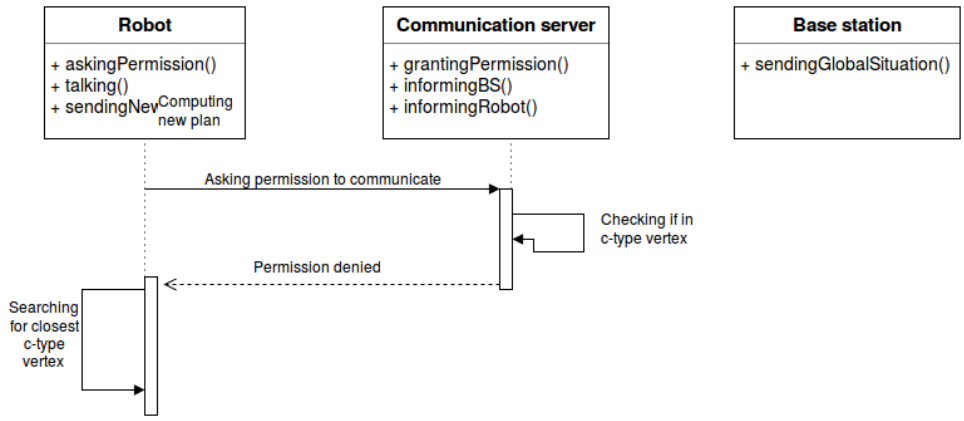


Figure 5.4: Failed communication procedure.

Chapter 6

Experimental evaluation

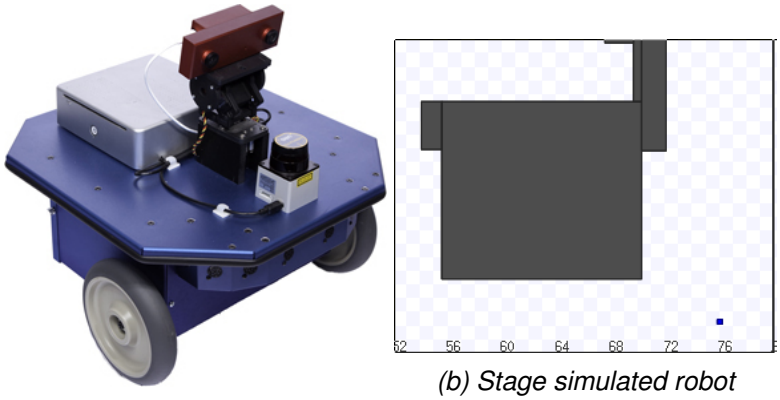
After having formulated the model, presented the strategies we implemented to fulfill the patrolling task, and developed the software architecture able to carry them out, we tested our framework in different scenarios. This chapter presents the experiments used to test the validity of our communication-restricted framework, firstly discussing how they are organized, and then discussing in details of the obtained results.

6.1 Experiments setup

As already said in Chapter 5, we implement our multirobot system within the ROS framework. To simulate the patrolling mission we use Stage [19], a 2D simulator that provides a virtual world, represented by a bitmap that it takes as input, populated by mobile robots, identified by a numerical id and equipped with sensors and actuators, together with various objects, if needed. The bitmap of the environment that Stage uses is a simple matrix of 0s and 1s, with the 0s representing areas of the environment that are free from obstacles, and 1s representing the areas in which one or more obstacles are present. The robots we simulate in the experiments are erratic-like robots. The Erratic is a full-featured, compact, and powerful mobile robot platform built by Videre [20]. It is provided with a ROS software, along with driver, navigation, and simulation support, and it is equipped with sonar sensors, with a detection range of 3 meters, translated by Stage in 30 pixels. Robots move in the environment at a maximum speed of 10px/second, slowing down during turns or in proximity of other robots. In Figure 6.1 we show the Erratic robot platform and the way it is represented in Stage's simulations.

We test our system in three different environments:

- The Leonardo campus of the Politecnico di Milano (Poli – approx. size 200×150m, robots' maximum speed 2.5m/s). It is characterized by the



(a) The real Erratic robot

(b) Stage simulated robot

Figure 6.1: The Erratic robot platform and its Stage representation

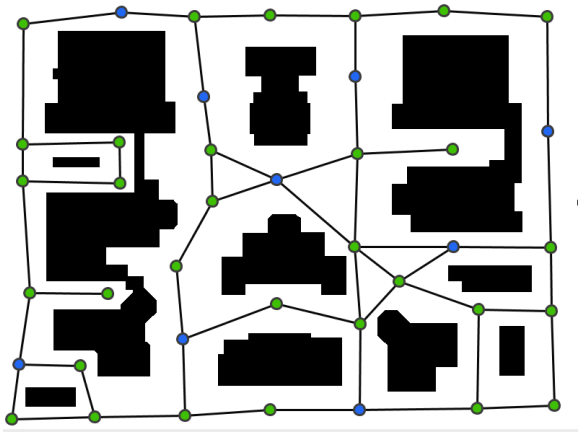
presence of large obstacles, buildings, that the robots need to avoid while moving in the free environment. The map reproduced by Stage is 800x600 pixels;

- the "Acapulco convention center" from the Radish repository [21] (Open – approx. size 80×60m, robots' maximum speed 1m/s). It is characterized by a large open area and few obstacles. In this case too the map is 800x600 pixels;

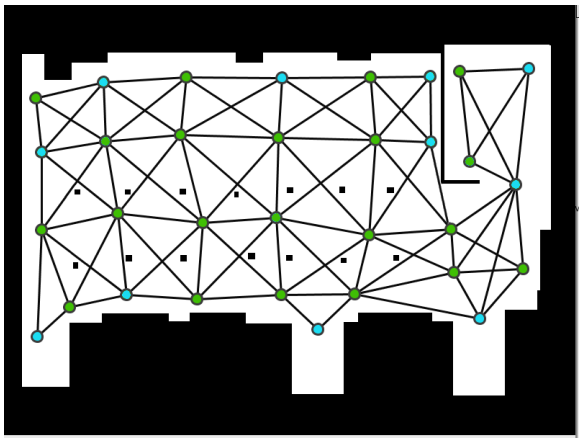
- the Manhattan grid environment, an imaginary, regular grid-block environment, already used in [2, 5] in the context of classical multirobot patrolling. Also for this last environment the map is 800x600 pixels.

The discretization of the environments is done through topological graphs, shown in Figure 6.2. The graphs are obtained by choosing randomly some points, sufficiently far from the obstacles, and manually pruning some of them in order to obtain a complete topological representation of each environment. Once these points have been generated, edges are build between two vertices on the same line of sight, and c-type vertices are chosen manually. In particular, the Poli environment is discretized into a graph with 40 m-type vertices, 9 of which also belong to V_c; the Open environment is discretized into a graph with 32 vertices, 11 of which only belong to V_c and the remaining 21 only belong to V_m; and the Manhattan grid environment is discretized into a graph with 20 vertices, 4 of which only belong to V_c, 12 only to V_m, and the remaining 4 to both V_m and V_c.

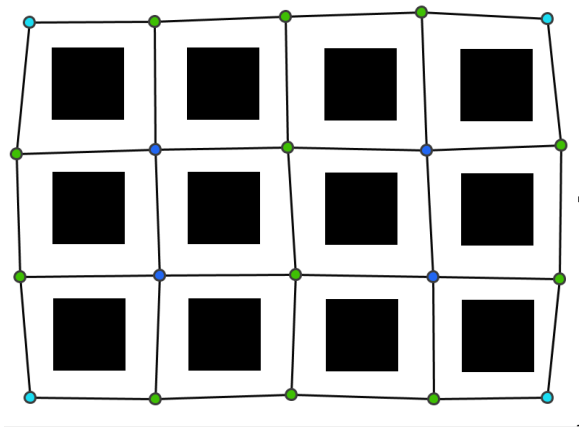
In our experiments, we assume that the robots have a complete knowledge of the environment, and that they are ablo to perfectly localize themselves in it. Moreover, they get to know the positions of the other robots of the team as soon as they enter their scan range. in fact, while patrolling the environenment, we assume that



(a) Poli



(b) Open



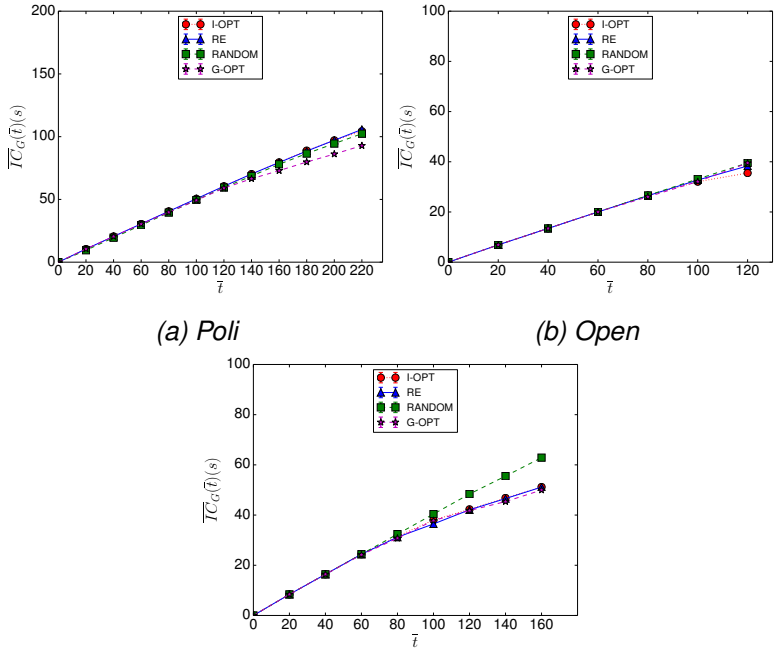
(c) Manhattan grid

Figure 6.2: The three environments. Green, blue, and light-blue vertices define the sets $V_m \setminus V_c$, $V_m \cap V_c$, and $V_c \setminus V_m$, respectively.

the sonar sensors of the robots are always active so that they are able to perceive moving obstacles when they are dangerously near. In our case, such moving obstacles are the other members of the team. In order to compute an obstacle-free path to move towards the targets in the environment, each robot implements the algorithm A* on an Occupancy Grid Map, in which each cell corresponds to a square 5x5 pixels of the input bitmap. In the Occupancy Grid Map the obstacles are inflated, each of them covering more cells than the ones covered in the original bitmap. More specifically, we add a square of 3x3 pixels around each obstacle. This is done to be sure that robots do not move too close to the obstacles, with the risk of accidentally crashing against them. When executing the A* algorithm, also the teammates are treated as obstacles. This means that, in the cell corresponding to their positions in the Occupancy Grid Map, at the moment of the A* path calculation, an additional set of obstacle cells is added to be sure that A* does not choose a path that will lead to robot-robot collisions. As robots are continuosly moving in the environment, it may happen that a robot computes the A* path before realizing that another robot is along it. In order to avoid collisions in this case, we implement the following algorithm: when two (or more) robots enter each other's scan range, the robot with the lower id stops and waits for the one with the highest id to compute a new path and to move away from the possible collision area before starting to move again towards its target.

In each experiment, runs are repeated 5 times, in order to take into consideration the fact that the outcome of the mission is influenced by the order of arrivals of messages in ROS, which is not deterministic. In each run, we simulate T $\bar{3}$0 minutes of patrolling mission, , except where indicated otherwise, as we noticed, in some preliminary experiments, that the *average idleness* values tend to stabilize after such an amount of time. Since the Stage simulator is asynchronous and its ability to coordinate the messages needed to make the robots move in the world tends to worsen as the number of messages increases (notice that the messages related to the motion of the robot never stop to be sent between the simulator and the robots), we had to interrupt and discard the outcome of 5% of the runs, as some messages were lost and robots crushed against each other. All the simulations are performed on a laptop equipped with an Intel P7450 processor and 2 GB of RAM.

In the following section we present all the valid experiments we carried out and the obtained results, in terms of performance metric. We display the results within a confidence interval of 95%, plotting it in the graphs used to comment them.



(a) Poli *(b) Open*

(c) Manhattan grid

Figure 6.3: Comparison of G-OPT, I-OPT, RE, and RANDOM through a short mission horizon with 2 robots

6.2 Experiments and results

6.2.1 Comparison between G-OPT and distributed strategies

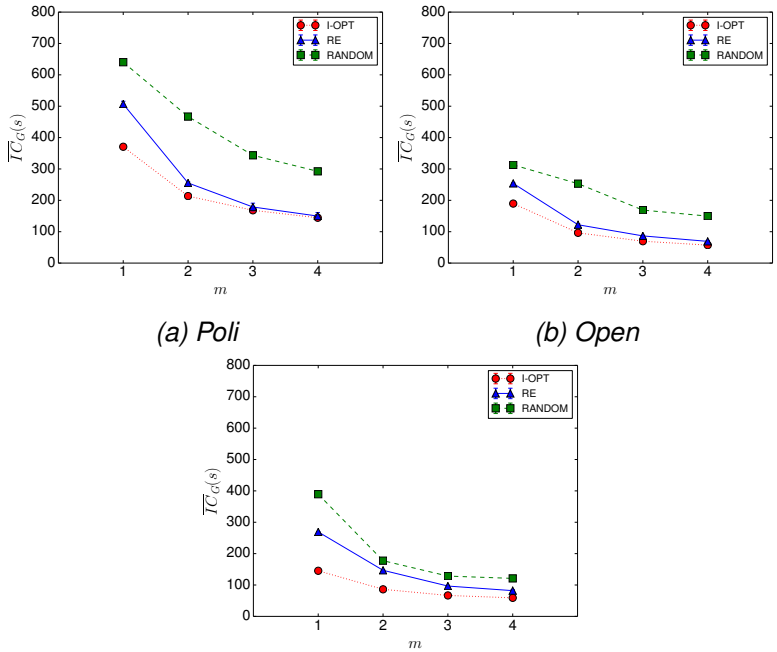
The first set of results that we present derive from the comparison between the centralized strategy, G-OPT, and the distributed strategies, I-OPT and RE, with the addition of the random baseline (RANDOM). Since G-OPT strategy is run offline at the beginning of the mission and computes the plan for all the robots of the team, and for the whole duration of the mission, the computational load is substancial. In order to make the BS able to obtain the plans in a reasonable amount of time, we decided to limit the team size to 2 robots and the duration of the mission to the H (planning horizon used in the I-OPT strategy) plus the time needed to compute the plans. Notice that, due to the different topology of the environments, H must be setted differently for each of them, to make it coherent with the time robots may use in reality: for the Poli environment it is equal to 200 seconds, for the Open environment it is equal to 100 seconds, for the Manhattan grid environment it is equel to 150 seconds. The reason why we have to lower it for the both the Open and Manhattan grid environments is that the graph representing them is more connected, providing a larger number of plans to be compared.

From the results shown in Figure 6.3, we notice that the behaviours of the strategies depend on the topology of the environments but, I-OPT and RE are able to perform as well as G-OPT on short mission horizons, and justify their employment on longer horizons and environments in which G-OPT is not a viable choice. Notice that in the Open environment I-OPT behaves slightly better than G-OPT. This is due to several factors that come into play when performing realistic simulations, such as robot interfering with each other while executing their path, and a non-perfect simulation of the concurrent walks execution. It must be noted that the results shown in this subsection are not particularly meaningful, as they only deal with the initial seconds of the mission in which the random baseline too performs as well as the other strategies.

6.2.2 Variable number of robots

This set of experiments aims at testing the I-OPT strategy, the RE strategy, and the random baseline (RANDOM) for all the environments and for a variable number of robots m. More precisely, we investigate the behaviour of the strategies making m vary from 1 to 4. In this case, we fix the planning horizon H. Specifically, we set H to 150 seconds for the Poli environment; we set H to 100 seconds for the Open environment; and we set H to 150 seconds for the Manhattan grid environment.

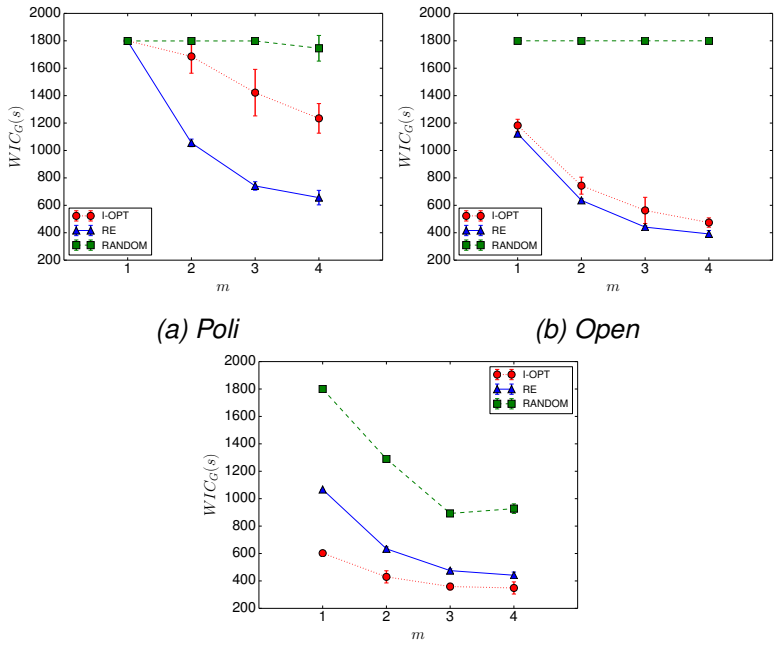
In Figure 6.4 we show the *average idleness* values obtained on the three environments. It is immediately noted that, in all the environments and for all the values of m, RE outperforms RANDOM, and it is in turn outperformed by I-OPT: these results reflect the increased planning complexity and the knowledge exploited by the three strategies. Moreover, increasing the value of m leads to an advantage for all the strategies, but the performance gain becomes always smaller: this means that, in presence of resource constraints, one can obtain good performances with relatively small team sizes. It is also particularly interesting to notice that, when taking into consideration the the *worst-case idleness* values (Figure 6.5), I-OPT is outperformed by RE in Poli and Open. This is not surprising, as RE is designed to lead the robot always toward the vertex currently displaying the highest idleness and so greedily minimizing the worst-case idleness, while I-OPT plans without considering worst-case idleness values. The same result is not visible in the Manhattan grid environment. This is due to the position of c-type vertices in the map: they are all relatively close to each other, and this leads to frequent robot-robot interference, especially in the RE strategy where the objective of the robots is to visit the vertex with highest idleness and report immediately to the BS. Using this strategy it is more frequent for robots to move simultaneously to a c-type vertex. Again, larger team sizes lead to lower worst-case idleness values in all the environments and for all the strategies, with the only exception of RANDOM in Open,



(a) Poli *(b) Open*

(c) Manhattan grid

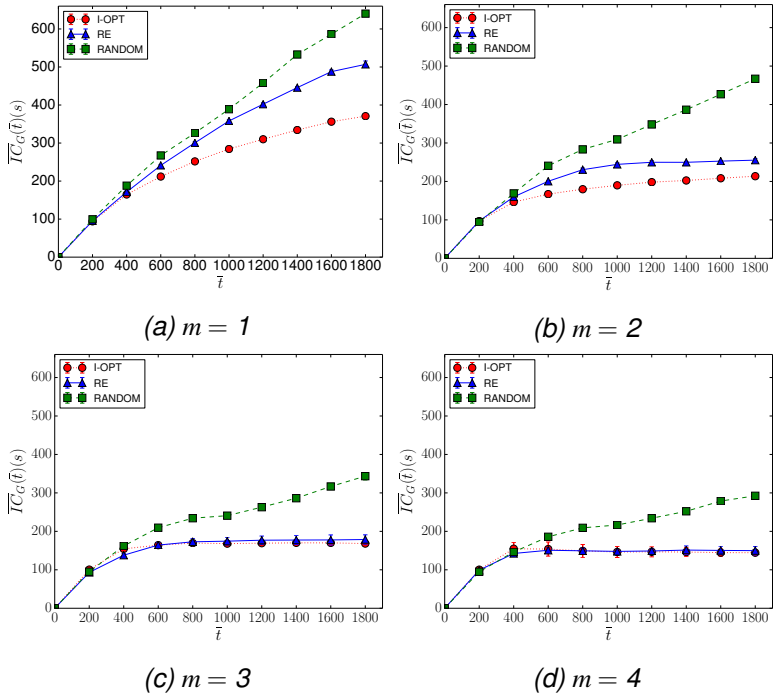
Figure 6.4: Average idleness for different team sizes m in all environments



(a) Poli *(b) Open*

(c) Manhattan grid

Figure 6.5: Worst-case idleness for different team sizes m in all environments



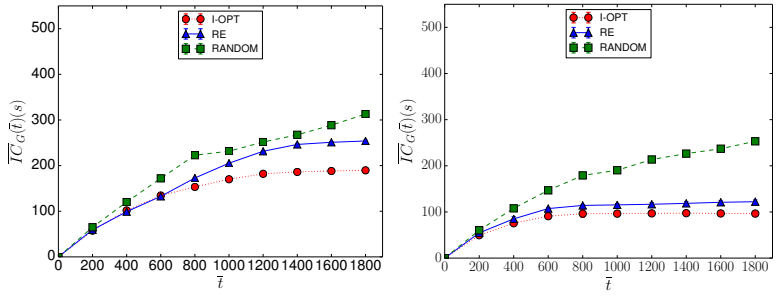
(a) $m = 1$ *(b) $m = 2$*

(c) $m = 3$ *(d) $m = 4$*

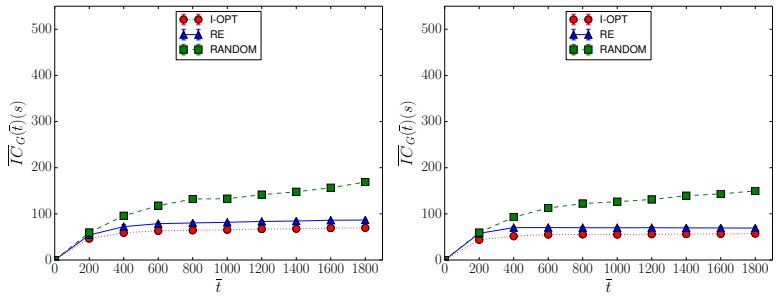
Figure 6.6: Idleness evolution Poli for different team sizes m

where there is always at least one vertex whose situation is never reported to the BS.

In the "Evolution" graphs (Figures 6.6, 6.7, and 6.8) we show the entire evolution of the *average graph communication idleness* from the beginning of the mission to its end. As expected, the more robots are participating in the mission the sooner the *average graph communication idleness* stabilizes and the lower is the stabilization value. An interesting fact to point out is that, even if strategy I-OPT always guarantees the best results, the ones achieved by the RE strategy are not very far from them. It must be reminded that the topology of the graph representing the environments plays a fundamental role in determining the performance of the strategies. In fact, the values of the *average graph communication idleness* are significantly lower in the Open environment and in the Manhattan grid environment because of two main reasons: the graphs representing both Open and Manhattan grid are more connected, having always at least two edges starting and ending in each vertex; and both graphs are characterized by the presence of c-type vertices not contributing to the increase of the idleness, as they are not m-type.

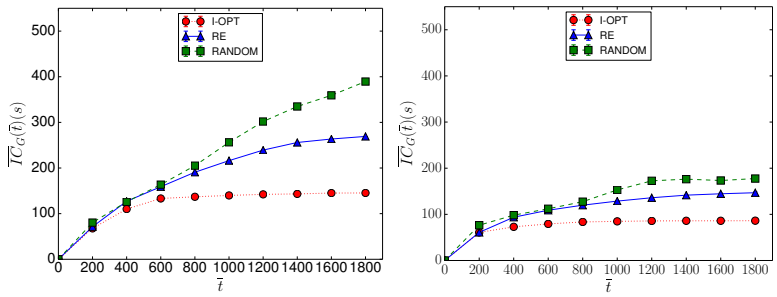


(a) $m = 1$ *(b)* $m = 2$

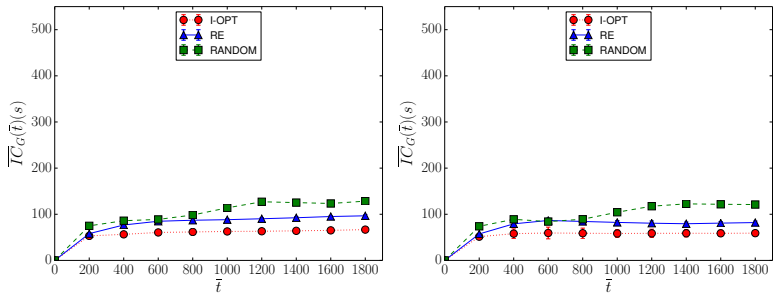


(c) $m = 3$ *(d)* $m = 4$

Figure 6.7: Idleness evolution Open for different team sizes m



(a) $m = 1$ *(b)* $m = 2$



(c) $m = 3$ *(d)* $m = 4$

Figure 6.8: Idleness evolution Manhattan grid for different team sizes m

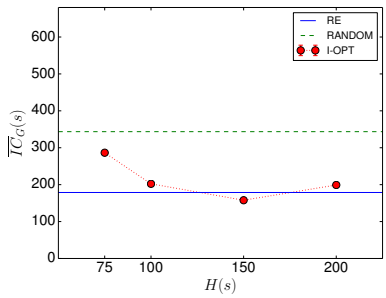


Figure 6.9: Average idleness for different H

6.2.3 Variable planning horizon

With this set of experiments we investigate how varying H leads to different performances. This is done only for the I-OPT strategy, being the only one influenced by H. In this case, we fix the environment, running the experiments only for Poli. The other fixed parameter is the number of robots, set to 3. The different values of H that we use are: 75, 100, 150 and 200 seconds.

The results shown in Figure 6.9 demonstrate that, using a planning horizon lower than 150 seconds leads to poor results, as the computable path are not long enough to reach all vertices, but also augmenting it to 200 seconds does not improve the overall performance. Instead, it leads to an increasing of the computational time, used to calculate each plan, which causes the worsening of the performance since less time is available to patrol the environment.

6.2.4 Variable number of c-type vertices

This set of experiments involves changing the number of c-type vertices in the graphs representing the environments. With this kind of experiments we want to analyze how adding or removing c-type vertices, namely making the problem more or less simple respectively, influences the overall performance of I-OPT and RE strategies (also comparing them to the random baseline). The different percentage of c-type vertices analyzed are: $|V_c|/|V| = 0.05$, $|V_c|/|V| = 0.10$, $|V_c|/|V| = 0.20$, and $|V_c|/|V| = 0.40$. In this case too, we fix the environment in which running the simulations to the Poli environment, where each c-type vertex is also an m-type vertex. Also H and the number of robots are fixed and equal to 150 seconds and 3, respectively.

The results (see Figure 6.10) show clearly that the more c-type vertices are present in the graph, the better the performance of all strategies. It is particularly interesting to notice that, when the number of c-type vertices approaches the 50% of the overall number of vertices, following the I-OPT strategy leads to consider-

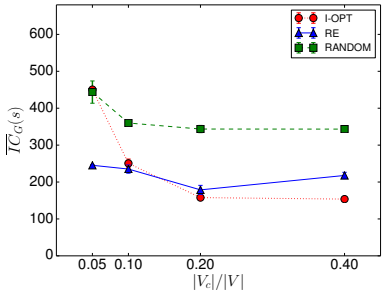
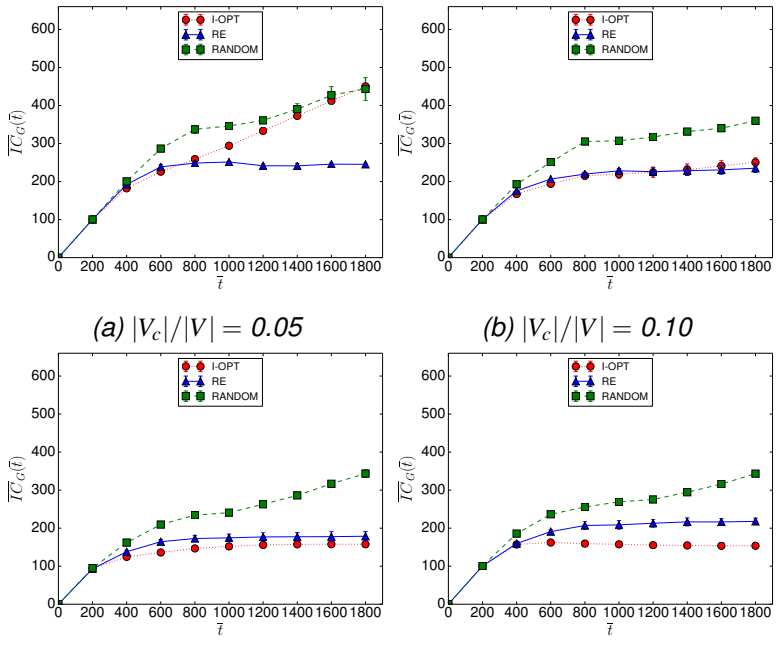


Figure 6.10: Average idleness for different % of c-type vertices



(a) $|V_c|/|V| = 0.05$ *(b) $|V_c|/|V| = 0.10$*

(c) $|V_c|/|V| = 0.20$ *(d) $|V_c|/|V| = 0.40$*

Figure 6.11: Idleness evolution varying number of c-type vertices

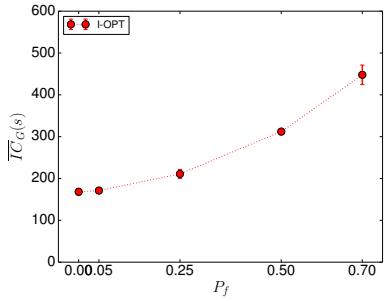


Figure 6.12: Average idleness for different probability of communication failure

ably better performances than the RE strategy. This is even more visible in the graphs describing the evolution of the *average communication idleness* during the entire duration of the mission, in Figure 6.11. Here, for $|V_c|/|V| = 0.40$, the stabilization value is reached much sooner from the I-OPT startegy than from the others, and is significantly lower. The main reason why RE strategy does not improve together with I-OPT is that, since more c-type vertices exist, it is more rare that the obtained walk need to be augmented to reach one of them, implying that less m-type vertices are visited as a side-effect. The last important thing that we can notice is that for $|V_c|/|V| = 0.05$ I-OPT behaves as badly as RANDOM. The reason is that, in this specific case, there are only two c-type vertices, placed at two opposite corners of the map: therefore, a planning horizon of 150 seconds always leaves out from the plans some m-type vertices that are too far from them.

6.2.5 Variable probability of communication failure

The last set of experiments deals with a realistic case in which we simulate the temporary unavailability of a communication link with the BS. Faults are assumed to happen independently from each other as follows: once a robot reaches a c-type vertex, with probability p_f the expected communication link will not be present (due, e.g., to a temporary network congestion), while with probaibility $1 - p_f$ the robot will be able to communicate with the BS as before. To overcome these unexpected events, we examine the behavior of a simple recovery procedure in which the robots decide to move towards the closest c-type vertex. We focus on the best strategy for $\overline{IG}_G$ in the Poli environment, as the recovery procedure is independent of the chosen patrolling strategy, and examine how the patrolling performance $\overline{IG}_G$ degrades for varying values of p_f.

The results reported in Figure 6.12 show that such a simple backup strategy is effectively able to mitigate the impact of communication failures up to 25%, while for higher values of p_f the performance worsens less gracefully, but not dramatically.

Chapter 7

Conclusions

This thesis has focused on investigating different multirobot patrolling strategies, in which robots must persistently patrol an environment and transmit to a base station (BS) the status of some locations of interest, given that communication is possible only in some "communication zones". In this thesis, we explored both a centralized and a distributed approach, in which each robot autonomously computes its plans according to different amount of knowledge about the plans followed by others, with the objective of minimizing the *average graph communication idleness*. More specifically, after having formulated the multirobot patrolling problem in our communication-restricted framework and having extended the classical *idleness* evaluation metrics to fit in it, we designed two *ad hoc* distributed strategies that allow the fulfillment of the patrolling task. In both strategies, the robots do not rely on the BS for computing the joint plan to successfully carry out the mission, but each of them computes its own plan, given some information received from the BS. To evaluate the designed strategies, we implement a centralized patrolling strategy in which the BS takes care of computing the best joint plan at the beginning of the mission and then sends to each robot the plan it has to follow.

The experimental evaluation has been performed using ROS/Stage simulations. These experiments were executed in three environments, with different topological characteristics, and making important parameters change. We explored how the performance, in terms of *average graph communication idleness*, is influenced by the variation of the number of robots in the team, of the tatal percentage of c-type vertices in the graph, of the planning horizon H, and of the probability p_f of failing a communication. The results obtained are consistent with what we expected: they show that the distributed strategies performs as good as, and under certain conditions even better than, the centralized one, as they lighten the computational load for the calculation of the joint plan by dividing it among the robots in the team. Moreover, we noticed that better results are achieved when increasing the number

of robots and/or the number of c-type vertices over the total number of vertices of the graph.

Since several issues relative to our communication-restricted framework have not been explored yet, possible future works may investigate new distributed strategies in order to find which is the best one for this kind of setting. This future study may also include a theoretical evaluation of *cyclyc strategies*, as they have been investigated in most of the existing literature on multirobot patrolling. Finally, it would be necessary to validate our approach also on real robots.

Bibliography

[1] J.D. Zucker A. Machado, G. Ramalho and A. Drogoul. Multi-agent patrolling: an empirical analysis of alternative architectures. In *Proc. MABS*, pages 155–170, 2002.

[2] Y. Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proc. IAT*, pages 302–308, 2004.

[3] V. Corruble H. Santana, G. Ramalho and B. Ratitch. Multi-agent patrolling with reinforcement learning. In *Proc. AAMAS*, pages 1122–1129, 2004.

[4] L. Marchetti L. Iocchi and D. Nardi. Multi-robot patrolling with coordinated behaviours in realistic environments. In *Proc. IROS*, pages 2796–2801, 2011.

[5] D. Portugal and R.P. Rocha. Multi-robot patrolling algorithms: examining performance and scalability. *Adv. Robotics*, 27(5):325–336, 2013.

[6] M. Tortonesi, C. Stefanelli, E. Benvegnu, K. Ford, N. Suri, and M. Linderman. Multiple-uav coordination and communications in tactical edge networks. *IEEE Commun Mag*, 50(10):48–55, 2012.

[7] S Ochoa and R Santos. Human-centric wireless sensor networks to improve information availability during urban search and rescue activities. *Inform Fusion*, 22:71–84, 2015.

[8] N. Basilico J. Banfi and F. Amigoni. Minimizing communication latency in multirobot situation-aware patrolling. In *Proc. IROS*, pages 616–622, 2015.

[9] N. Agmon Y. Elmaliach and G. Kaminka. Multi-robot area patrol under frequency constraints. *Ann Math Artif Intel*, 57(3-4):293–320, 2009.

[10] E. Stump and N. Michael. Multi-robot persistent surveillance planning as a vehicle routing problem. In *Proc. CASE*, pages 569–575, 2011.

[11] M. Schwager J. Yu and D. Rus. Correlated orienteering problem and its application to informative path planning for persistent monitoring tasks. In *Proc. IROS*, pages 342–349, 2014.

[12] E. Plaku A. Wallar and D.A. Sofge. Reactive motion planning for unmanned aerial surveillance of risk-sensitive areas. *T-ASE*, 12(3):969–980, 2015.

[13] N. Basilico and S. Carpin. Online patrolling using hierarchical spatial representations. In *Proc. ICRA*, pages 2163–2169, 2012.

[14] N. Gatti N. Basilico and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proc. AAMAS*, pages 57–64, 2009.

[15] G. Hollinger and S. Singh. Multirobot coordination with periodic connectivity: theory and experiments. *T-RO*, 28(4):969–980, 2012.

[16] D.D. Corkill. Blackboard systems. *AI Expert*, 9(6):40–47, 1991.

[17] K. Conley J. Faust T. Foote J. Leibs E. Berger R. Wheeler Q. Morgan, B. Gerkey and A. Ng. Ros: an open-source robot operating system. In *Proc. ICRA*, volume 3, pages 1–6, 2009.

[18] S.H. Juan and F.H. Cotarelo. Multi-master ros systems, 2015. http://http://www.iri.upc.edu/.

[19] R.T. Vaughan. Massively multi-robot simulations in stage. *Swarm Intell*, 2(2–4):189–208, 2008.

[20] H. Rev. *ERA Mobile Robot User Manual*, 2009.

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