Université Libre de Bruxelles
Faculté des Sciences Appliquées
CODE - Computers and Decision Engineering
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

A network of ceiling cameras performs distributed path planning to guide a ground robot

Andreagiovanni REINA

Promoteur:
Prof. Marco DORIGO

Rapport d'avancement de recherche
Année Académique 2011/2012

Abstract

We introduce zePPeLIN, a distributed system designed to address the challenges of path planning in large, cluttered, and dynamic environments. The objective is to define the sequence of instructions to move a ground object from an initial to a final configuration in the environment. zePPeLIN is based on a set of wirelessly networked devices, each equipped with a camera, deployed in environment. Cameras are placed at the ceiling. While each camera only covers a limited environment portion, the camera set fully covers the environment through the union of the field of views. By local message exchanging, the cameras cooperatively compute the path for the object, which gets moving instructions from each camera when it enters camera's field of view. Path planning is performed in a fully distributed way, based on potential diffusion over local Voronoi skeletons. The task is made challenging by intrinsic errors in the overlapping in cameras' field of views. We study the performance of the system vs. these errors, as well as its scalability for size and density of the camera network. We also propose a few heuristics to improve performance and computational and communication efficiency. We report about extensive simulation experiments and validation using real devices.

Acknowledgement

First of all, I would like to thank my supervisor Prof. Marco Dorigo. I am very grateful for the opportunity he gave me to work at his prestigious Swarm Intelligence Laboratory.

I would also like to thank Dr. Gianni Di Caro and Prof. Luca Gambardella for their close collaboration and their valuable help. Part of this work has been conducted under their supervision at the Artificial Intelligence Institute IDSIA in Switzerland.

The research leading to the results presented in this paper has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n° 246939.

Contents

Chapter 1

Introduction

In this work we present a distributed system for path planning. Path planning refers to the calculation of the path that an object has to follow for moving from a starting point to a given destination/configuration. This is a fundamental problem in mobile robotics, where the moving object can be the robot itself, a part of it (e.g., its robotic arms), or an object being carried by one or more robots. The calculation of a plan consists of the definition of the precise sequence of roto-translations for moving the object without hitting obstacles or other robots. A large number of different versions, algorithms, and solutions to this problem have been proposed in the last three decades (e.g., see [13, 5, 14] for overviews). In this work we focus on the version which is also informally referred to as the *Piano mover's problem*. In this case, the controllable degrees of freedom of the moving object are equal to the total degrees of freedom, meaning that the moving object (the piano) has not dynamic constraints on the motion (*holonomic motion*). The Piano mover's problem assumes that the agent who plans the path has as input the map of the environment and the object's model. However, in many practical cases, an input map including the precise deployment of furniture and the status of other typical dynamic obstacles (e.g., a closed door, or the presence of an occluding human crowd) is not immediately available. Therefore, in these cases, an up-to-date map needs to be gathered on the spot prior to path planning. However, in the case of planning over large areas, this poses a clear problem, since in practice a collection of sub-maps, each referring to a portion of the environment, needs to be gathered on the fly and possibly merged with each other in order to perform a consistent and optimized path planning over the entire area.

To tackle this class of problems and, in particular, for planning the path of a holonomic object of any shape over a large and cluttered area, we propose *zePPeLIN (Path PLannIng Network)*, an approach based on the sensorization of the environment. A set of networked smart cameras is deployed on the ceiling of the area where the object moves in order to visually cover the full area on the ground. Camera deployment can be performed in any convenient way. In the following, without losing generality, we assume the use of a *swarm of flying robots*, each equipped with a camera and a wireless communication system (see the description of the robot camera model in the next chapter). Exploiting the top view, the camera system builds the map of the environment and the model of the moving object. Each camera of the network plans the local part of the path which is relative to its field of view. Then, it communicates with its neighbor cameras to locally merge the sub-paths and to cooperatively generate a global path for the ground moving object. That is, the zePPeLIN system plans the precise sequence of roto-translations that the moving object on the ground has to perform in order to reach its given final configuration. During

path execution, each robot camera visually localizes the moving object and sends to it the required motion information as it moves into the wireless range of the robot camera itself. In this way, the use of the zePPeLIN networked camera system allows to effectively perform path calculations over large areas and can deal with dynamic changes in the environment by exploiting its parallel and distributed nature. Moreover, the system can be used to make path calculations for multiple ground-moving objects/robots at the same time.

All these advantages have however a cost: the problem is made particularly challenging by the fact that each camera has a limited field of view, so that a camera can only see a limited portion of the environment, and the fields of views of neighbor cameras are partially overlapping, with inherent uncertainties in terms of mutual alignment and size of overlapping areas (a graphical illustration of this is shown in Figure 4.1, explained in Chapter 4). The final global path is composed by partial paths which are locally calculated and linked together in the overlapping areas. Wrong alignment and overlapping information can lead the system to fail finding feasible paths. In fact, since each camera only calculates a partial path, if the information concerning the relative positioning of the overlapping area is erroneous, the linking of the local paths leads to imprecise, potentially unfeasible connections. During the phase of actual path navigation, some sub-path disconnections can be dealt by locally calculating *repairing paths*, which feasibly reconnect the sub-paths at the expenses of some additional path length (see the discussion on navigation in Chapter 6). However, in a cluttered environment disconnected sub-paths may not have a feasible repairing path. In this case the calculated path is infeasible for navigation, resulting in a global failure. Therefore, an important contribution of this work is the study of the robustness of the distributed path planning process to these errors, that are intrinsic to any distributed vision-based (or map-based) approach. Another contribution concerns the scalability of the system in terms of number of cameras and area size in the presence of these same errors. To study these issues, we performed an extensive set of experiments both in simulation and using real cameras and moving robots. In order to improve the efficiency and the accuracy of the distributed path planning process we also propose a set of heuristics and we study their impact on performance. Moreover, we consider ways to reduce the impact of camera alignment errors based on statistical reasoning.

In contrast to a centralized solution, where a single leading camera merges the visual information from all cameras, the proposed zePPeLIN approach only relies on local and partial information. The entire process is fully distributed, and the communication between cameras is only local. Such architectural choices are motivated by the objective of obtaining a system which is scalable, robust to individual camera failures, and requires minimal communication overhead. In fact, with a centralized approach, the system suffers from the problem of the *single point of failure* and in order to scale to large environments it has to cope with efficiency issues and communication bandwidth bottleneck. While these issues could be partially overcome with the use of effective ad-hoc routing algorithms for communications, the zePPeLIN solution is still expected to be more general, portable, and efficient. It is also important to remark that an alternative approach based on the ground robot building by itself the current obstacle map of the environment and performing self-localization (e.g., a typical SLAM approach) and path planning, while potentially more precise, would require a much longer computation and execution times than our approach (especially in the case of large areas), and would be more prone to path errors (e.g., while performing SLAM or moving through an area, the obstacle situation in nearby areas could have changed).

While in this introduction and in the rest of the paper we speak and makes use of

vision-based sensors to build local maps of the ground environment, we point out that the zePPeLIN distributed path planning algorithm that we propose can be used with *any* system of networked devices capable of locally mapping the environment. For instance, a networked system of *kinect*-equipped or *laser*-equipped nodes could be equally used in the zePPeLIN framework, producing equivalent (if not better) performance of a camera-based system.

The rest of the article is organized as follow. In Chapter 2, we define the problem and the reference model used for the robotic ceiling cameras. In Chapter 3, we describe the state of the art methodology that we used to implement the distributed planner. In Chapter 4, we describe the distributed path planner. In Section 4.1, we propose a set of heuristics for improving the performances of the system, that is, the quality of the solution and the efficiency of the process. Section 4.2 shows how the proposed system can deal with dynamic environments using local adaptation to changes of obstacle positions and camera failures. We present the results of the simulation experiments in Chapter 5. The experiments are designed to study the performances of the system on the main aspects of interest: robustness, efficiency and scalability. Moreover, we study how the various proposed heuristics affect the performances of the system. In Chapter 6, we describe the real robot experiments that we designed and implemented for validating the simulation results. In Chapter 7, we list other works which address similar problems with a distributed or multi-agent approach. Finally, in Chapter 8 we conclude the paper with final remarks and the promising direction that will be investigated in future works.

Chapter 2

Path planning scenario and camera network model

In zePPeLIN, we consider the following settings for the distributed path planning of a holonomic object on the ground. At the beginning, a network of robotic ceiling cameras is deployed in the environment where the object moves. As pointed out in the previous chapter, at this aim we assume the use of a swarm of flying robots each equipped with a camera. Each robot camera faces the floor, in order to acquire a top view of the local environment underneath. The deployment is such that, altogether, the robot cameras cover with their visual field the ground environment including the start and the target position of the object. Robots are equipped with an on-board wireless communication system. The deployment is performed such that the robots are locally in range with each other and *the fields of view of two neighbor robot cameras overlap*. The size of the overlapping area must be greater of or equal to the dimension of the moving object. This constraint is to allow the cameras to connect sub-paths during the distributed path planning process by exchanging the local coordinates of the moving object. Once found their position in the environment, robotic cameras attach to the ceiling and keep a static position for the whole planning process. In this way, they can save energy while keeping monitoring the area. Since the focus of this work is on planning, we do not study how the camera formation can be obtained. However, in literature a number of algorithms that perform this kind of spacial deployment in the environment can be found, and any of them could be used for this purpose (e.g., [26]).

We assume that the robotic cameras are equipped with a *relative positioning system*, which allows a robot camera to estimate, with some uncertainty, the relative position (in terms of angle and distance) of neighbor cameras. This information is used to estimate the overlapping area between the fields of view of neighbor cameras and, in turn, to locally connect the sub-paths calculated by each camera (see Chapter 4). Unfortunately, an error in the estimate of the relative position of two robot cameras results in an erroneous estimation of their overlapping fields of view, which can potentially have disastrous effects on the quality and feasibility of the calculated paths (see Figure 4.1). In this work, we study how the error in mutual positioning affects the performance of the system, also in relationship to the number of robot cameras participating to the planning process. We consider this uncertainty as being an intrinsic issue of any vision-based distributed path planning, and, as such, we consider it as an internal parameter of the problem. That is, we do not propose ways to reduce it, for instance by means of sophisticated position calibration techniques; rather we empirically study to which extent this parameter impacts on the feasibility and

the quality of the planned paths. In the simulation experiments of Chapter 5 the error on relative position and orientation is independently generated for each camera from two independent zero-mean Gaussian distributions with configurable standard deviation. In this way, we model error estimates in a rather general and, at the same time, realistic way, avoiding to make assumptions that would be specific to the hardware in use.

As reference model for the robotic ceiling camera, we consider quad-rotor based flying robots. In particular, we consider the *eye-bot* [24], a quad-rotor flying robot equipped with a pan-and-tilt camera that allows it to monitor what happens on the ground. The robots are also equipped with a Wi-Fi communication system and with a infrared range and bearing system (IrRB) [25], which is used for calculating the relative position of other eye-bots and also provide a low-bandwidth line-of-sight data communication channel. We limit the set of neighbors of an eye-bot —i.e., the communication range— to the set of neighbors detected by the IrRB system (i.e., 5 meters). An eye-bot can passively attach to the ceiling using a magnet, when a ferrous ceiling is present. In this case, the position of the robot camera is statically fixed and the robot can save energy, guaranteeing a relatively long operation time. Moreover, as typical quad-rotors can easily do, the eye-bot can also hoover on the spot it has selected. In this way the system can be effectively used also in outdoor environments (and in principle, to save energy, a robot can alternate between hoovering in the air and resting on the ground when the moving object is not in its neighborhood). Clearly, in this case the relative positioning error would fluctuate because of stability issues, introducing more errors that, however, could be smoothed out through repeated relative position exchanging using the IrRB system.

Since the focus of this work is on distributed path planning calculation, we make simplifying assumptions for some of the aspects of the scenario that are not strictly related to plan calculations. In particular, we assume that the start and target configurations are given as input. Moreover, we assume that the moving object can be moved by a system that can interpret instructions given by the zePPeLIN system (e.g., through a wireless system or a speaker). For instance, it can be a single ground robot, a set of assembled ground robots, or even humans (e.g., carrying a piano). In the real robot experiments of Chapter 6, the moving object that follows the camera instructions is a two-robot system connected together in a rigid structure, and we consider an indoor scenario where the robots can keep static positions at the ceiling.

Chapter 3

Path planning on a single environment map

The zePPeLIN system is based on the use of the partially overlapping vision maps of the environment gathered by the different robot cameras. Each robot camera performs path planning calculations on its own local map, and it tries to optimally connect its path with that of the neighbor cameras through local message exchanges. Therefore, at first, each robot camera performs a "centralized" path planning calculation based on its local map, which includes also the overlapping areas with its neighbors. We designed and implemented this centralized planner by extending current state-of-the-art deterministic 2D path planning algorithms [13, 5, 14]. The developed centralized planner is also used in the simulation experiments of Chapter 5 as baseline to assess the performance of the zePPeLIN distributed approach. In this case, the planner takes as input, as a single image, the entire view of the area where the object can move. The techniques, methods, and algorithms that we used for implementing the centralized path planner are described in the rest of this chapter.

Environment model. The environment where the object moves, the ground, is modeled as a plane discretized in a uniform 2D grid of squared cells: the *occupancy map*. Each cell of the occupancy map is labeled as *free cell* or *occluded cell*. The former are the cells that are free from obstacles, meaning that the path of the moving object can pass through the area of the free cells. The set of the free cells is termed the *free space*. The occluded cells are the cells occupied, or partially occupied, by an obstacle. Having a top view of the environment, the classification of the cells in free or occupied is performed through a 2D isometric projection of the obstacles on the ground. This way of proceeding might result in the definition of a subset of the actual free space, since the projection of some "table-like" object results in an occluded area, while in practice there could be space for passing under it (depending on the 3D geometry of the moving object).

Potential field. The centralized path planning method we use is based on a classical work in the field [13]. More specifically, we use the *numerical potential field technique*. First, a virtual potential field is computed over the occupancy map, subsequently the path of the moving object is calculated descending the gradient of the potential field. The potential field defines a force that attracts the object towards the destination, and at the same time repels the object away from obstacles. In our algorithm, the potential is a scalar function that defines the attraction/repulsion intensity for each cell. The function has a

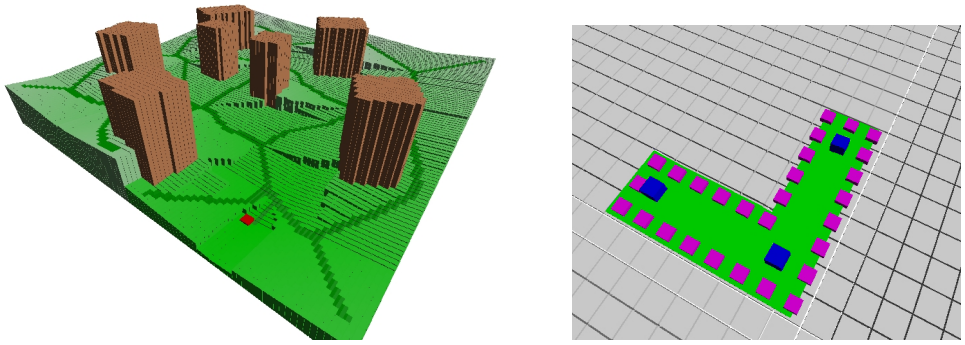
global minimum at the destination point, and maximum value on the obstacles. In all other cells the function decreases towards the destination, such that the planner can direct the object to the goal following the direction indicated by gradient descent. A 3D illustration of the potential field is provided in Figure 3.1a. The potential field is computed in three phases, described below: (i) calculation of the *skeleton*, (ii) diffusion of the potential field over the skeleton and (iii) diffusion over the remaining free cells.

Voronoi skeleton. The *skeleton* corresponds to the *Voronoi diagram* on the 2D occupancy map [28] (i.e., the set of points where the distance to the two closest objects is the same [5]). Since the planner operates on a discrete map, the skeleton is the subset of the free cells where the number of cells from the two closest occluded cells is the same. As final skeleton calculation step, the destination cell is connected to the skeleton by a shortest line path (i.e., the set of cells laying on the shortest line from the destination cell to the skeleton's closest cell).

Diffusion over the skeleton. Once the skeleton is computed over the entire map, a potential field value is assigned to each cell of the skeleton. This operation is based on a *diffusion process*. The diffusion starts from the destination cell, to which a zero value of potential is assigned. A zero value means that there is no potential force when the object lays on the destination cell. Then, the potential value is assigned to the neighbor cells of the set of the last cells to which a value has been assigned (after the first step, only the destination cell belongs to this set). The new potential value assigned to the neighbor cells is the potential value of the previous cells incremented by one. The process is iterated until the potential is diffused over all the cells of the skeleton.

Diffusion over the free space. After the skeleton cells have been assigned a potential value, the potential field is computed over the remaining free cells. The diffusion process is similar to the previous one. However, in this case the diffusion begins from the skeleton cells. In the case the importance of the skeleton needs to be enhanced —i.e., the desired path must overlap the skeleton— in the first diffusion step the increment of the potential value can be greater than one. In our algorithm we used 3 as incrementing value from the skeleton. For the subsequent diffusion steps, the incrementing value is fixed to 1. The process iterates until the potential field is diffused over all the free space connected to the destination. The value of the occluded cells is fixed to the maximum value. This means that a high repulsive force is applied to the cells occupied by obstacles (see Figure 3.1a for a graphical illustration of the process).

Path calculation. The last phase of the planning algorithm consists in the actual calculation of the path. It is computed following the gradient in the descending direction. The process begins from the starting position of the object and follows the decreasing value of the potential field. The potential field descent is computed using the least-cost finding A^* algorithm [11]. In order to apply A^*, the 2D map is seen as a graph where the cells represent the nodes and there is an edge between two nodes when two cells are adjacent (i.e., we use Minkowski's Taxicab Geometry [29]). The A^* algorithm finds the least-cost path from the given initial cell to the destination cell. It uses a distance-plus-cost heuristic



(a) (b)

Figure 3.1: Graphical representation of the potential field (a) and the moving object (b). The environment is discretized in a 2D matrix of cells. The height and the saturation of the cells represent the potential field value. Obstacles are represented by high brown cells, and the skeleton by the dark green graph. The moving object (the L-shaped green area) is represented by a set of control points (blue squares) and a set of collision points (purple squares).

function $f(x)$ which is computed on every cell x visited by the algorithm:

$$f(x) = g(x) + h(x) \tag{3.1}$$

where $g(x)$ is the *path cost* function and $h(x)$ is the *estimate* function. The former represents the distance from the starting cell to the current cell x. The latter is a heuristic estimate of the steps from the cell x to the destination. This value is the value of the potential field.

Control points. We consider a moving object that can have any arbitrary shape and whose size is greater than the cell size. Therefore, it can occupy more than one cell. We use *control points* [5] to deal with this situation. The object is described by a set of control points, an example is shown in Figure 3.1b. At each instant the object is in a precise *configuration*, which corresponds to the spatial position and orientation of the object. A configuration c consist of the coordinates of the control points, and the $g(c)$ and $h(c)$ values of Equation (3.1). The value of $g(c)$ is the cost of all the unit movements from the start configuration to c. A unit movement can be either a translation or a rotation. The translation is of one cell in one of the cardinal directions (N, E, S, W, equivalent to up, right, down, left) and has cost equal to 0.5. A rotation of θ degrees can have place centered on any of the control points, or on the center of mass of the control points, with θ a configurable parameter. The cost of a unit rotation is proportional to the number of traversed cells (in particular, it is the average number of cells traversed by all the control points). Therefore, an object with N control points has $[4 + 2 * (N + 1)]$ possible unit movements and neighbor configurations. The $h(c)$ function is defined as the average over the potential value of all the control points. To describe a configuration, an additional set of points is used, the *collision points*. These points lay on all the cells occupied by the moving object's perimeter (see Figure 3.1b). These points are not used for calculating the function $h(c)$, but are used instead for checking collisions with obstacles. In the case a

configuration has one (or more) collision points colliding with an obstacle, the configuration is not considered by the algorithm, being infeasible.

Chapter 4

The distributed path planner

The centralized planner described in the previous chapter is the basic building block of the zePPeLIN distributed path planner: it is used by each individual camera for planning limited to its local map. The distributed planner takes into consideration the whole set of local plans and define ways for their effective merging and coordination. The distributed algorithm is based on local sensing (the camera's limited field of view), local communication between neighbor cameras, and partial knowledge of the overall status of the planning process (no omniscient leader or centralized controller exists).

The distributed algorithm presented in this chapter only executes path calculation, and it terminates when a valid path is found. The implementation of the path, that is, the actual navigation in the environment of the moving object is described in Chapter 6.

The zePPeLIN distributed path planning algorithm is composed of three phases, which are described in detail below: (i) *Neighbor mapping*, (ii) *Potential field diffusion*, (iii)*Path calculation*. Figure 4.4 shows the zePPeLIN pseudo-code algorithm that is executed at a generic camera nodes; it includes both the local path planning and the navigation process.

Neighbor mapping. During this first phase, each camera estimates the overlapping area of its field of view with that of its neighbors. A robot camera n calculates the overlapping area between its field of view and the one of the its neighbor m in the following way (and repeats the same process for all its neighbors). First, n collects two pieces of information: (i) the relative position and orientation of m (measured using the range and bearing system); (ii) the size of m's field of view, received from m via wireless communication.

Then, with these two pieces of information it is able to calculate on its local 2D map the projection of the field of view of m.[1] Using the projection of the field of view of m, n calculates the overlapping area, the *open edges* and the *shared edges*. The open edges are the edges of n's field of view that lay inside the m's field of view. The closed edges are the edges of m's field of view that lay in n's field of view. A graphical representation of this process is showed in Figure 4.1.

Potential field diffusion. Calculation of the potential field is the second phase of the distributed path planning process. It is based on a diffusion process. Since each camera only sees a limited part of the environment, and the whole environment map is segmented across the networked camera system, robot cameras need to engage in a cooperative diffusion of the potential field. The process is fully distributed: starting from the cameras at the

[1] As a simplifying assumption, we consider that all camera robots are placed at the same height, avoiding, in this way, issues related to fields of view of different size.

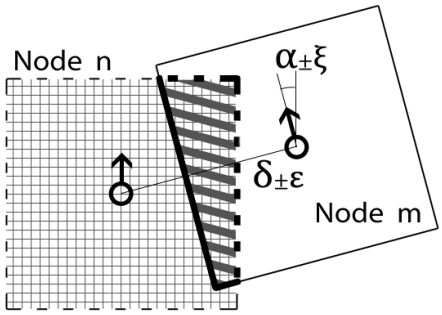
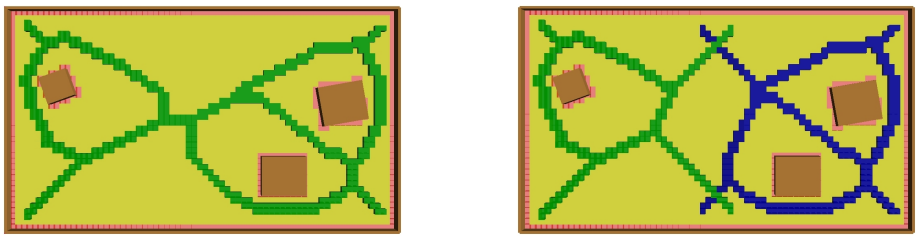


Figure 4.1: Illustration of how two neighbor camera nodes m and n define their overlapping region and assign shared and open edges. δ is the relative position (in x, y coordinates) of the robot camera node m in the reference system of n, α is the relative orientation of m with respect to the orientation of n. Using the δ and α estimates, node n builds a projection of the field of view of node m. In this way, node n defines the overlapping region (striped region), the *shared edges* (solid bold lines), and the *open edges* (dashed bold lines). δ and α are affected by the ϵ and ξ errors respectively. As discussed in Chapter 1, this results in an erroneous estimation of the overlapping area, which can potentially have negative effects on the calculated path.

goal location, each camera first computes the potential over its local map, then, it sends the potential field values of its shared edges to its neighbors, in order to allow them to continue with the potential field diffusion. From an operational point of view, the process is implemented as follows.

Each camera calculates the local *skeleton*, which corresponds to the *Voronoi diagram* on its 2D occupancy map (see Chapter 3). The environment's skeleton resulting from the sum of all local skeletons differs from the one which would be calculated in a centralized way using a single global map (Figure 4.2). Differences are due to the fact that, during skeleton calculation, the frontiers of a (local) map need to be considered as obstacles. Therefore, at the corners of each map the local skeleton shows bifurcations that do not find their counterpart in the centralized skeleton. Differences in skeleton turn into differences in the resulting path. This effect can be clearly observed from the results of the experiments with perfect alignment between cameras (see Chapter 5): the paths planned with the distributed process and the paths planned by the centralized planner have different length. An example of the paths generated by the global, single map planner and by zePPeLIN's distributed planner is shown in Figure 4.3.



(a) (b)

Figure 4.2: Comparison of the skeletons.

Once the skeleton is generated, cameras *cooperatively diffuse the potential*, where the value of the potential field of each cell represents the number of steps from the destination to the current cell. The cameras that have the *goal configuration* in their field of view start the process. They first diffuse the potential field on their local map. The diffusion starting point is the center of mass of the control points of the goal configuration. Once the diffusion in the local map is completed, they send to the neighbors the value of the potential field on the shared edges. In this way, a robot camera that receives this information can continue the diffusion starting from the values of the received edge cells. On reception, a camera copies the received values in the cells of its local map, and then executes the diffusion process on its local map starting from these received points. Each camera updates the potential value of a cell only if the new potential value is lower than the best value it had calculated so far (i.e., a lower number of cells toward the destination is found). This process is iteratively executed in a distributed way among all the cameras in the network.

Cameras that have the *final destination configuration* in their field of view behave in a slightly different way compared to the others. In fact, they need to precisely calculate the path that defines both the final position and orientation of the object. For this, the use of the information regarding the center of mass of the control points is not sufficient to guarantee the correct final orientation. Therefore, these camera calculate C different potential fields, one for each of the C control points. Each potential field is then diffused using as final destination point one of the control points. During path calculation, these cameras compute the $h(c)$ value averaging the potential field value of all the control points. However, differently from the other cameras, for each control point the value which is used is that of the corresponding potential field. In this way, during the third phase (see below), the cameras having the final destination in their view map can define the set of roto-translations of the object taking in consideration also its desired final orientation. Since only these cameras need to use multiple potential fields, this way of proceeding does not have a major impact in terms of computation and communications, yet it is able to guarantee the correctness of the path.

In this same respect, it is important to remark that, in order to *minimize communication overhead*, with the aim of favoring scalability, portability, and of minimizing energy consumption, cameras communicate among them only the value of the skeleton cells laying on the shared edges. In this way, communication messages only contain the values of a few cells, resulting in a low number of small packets of just a few bytes.

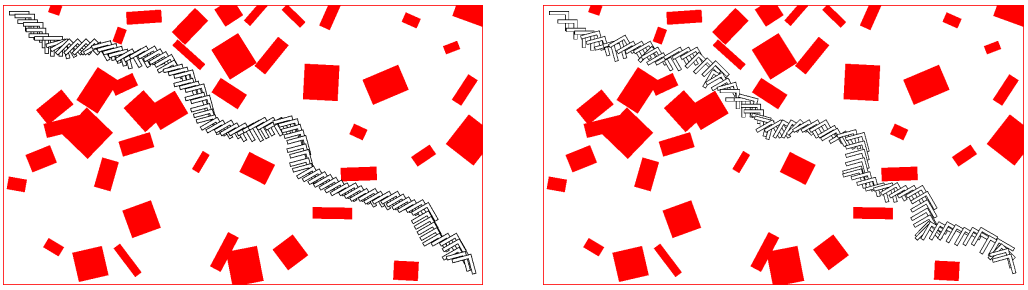


Figure 4.3: Final paths calculated by the global planner (Left) and by the zePPeLIN system (Right) in the same environment.

Path calculation. The third phase of the process consists in the actual calculation of the path, i.e., the definition of the sequence of roto-translations that connect the start and the final configurations. Each camera calculates the part of the path which is relative to its local map, and then sends a message to one of its neighbors for letting it continue the planning. The process is implemented in the following way.

As in the previous phase, the camera that sees the moving object (that is, the start configuration), hereafter indicated with s, begins the third phase. Camera s calculates on its local map the partial path of the object, ending on an open edge. Then, s randomly selects one of the neighbors with which it shares the open edge. Since each camera knows the projections of its neighbors' field of views (see Neighbor mapping of Phase 1), s can translate the coordinates of the moving object from its frame of reference to the neighbor's frame of reference. Finally, s sends these object coordinates to the neighbor, which then begins the local planning using as starting configuration the object positioned at the received coordinates.

The process iterates among all cameras until one of the cameras calculates the partial path that reaches the final configuration. When this happens, this camera broadcasts to all its neighbors a *success* message, which is flooded in a multi-hop fashion to all the other cameras in the network. In this case, the camera network has cooperatively found a path that connects the start and the end configuration positions. The information about the complete path is not stored in any specific camera, but instead it is fully distributed in the network: each camera has the knowledge of only the partial path relative to its field of view. Once the path has been defined, the camera s triggers path execution by communicating to the object the local information which is necessary to begin the actual path navigation.

During the phase of path calculation, the camera can be in *two possible states*. In one state, it performs the actual calculation of the partial path in its field of view, in the the other state it is waiting for a message from neighbor cameras. There are 4 possible messages that a camera m can receive from its neighbor n, which are listed below together with the actions that the reception of one of these messages triggers:

- *Start Path*: This message contains the coordinates of the control points of the moving object in m's frame of reference. After receiving from a neighbor n a Start Path message, m starts the path calculation using the received coordinates as start configuration. These coordinates represent the final configuration of the partial path as calculated by n.

- *Local Failure*: This message is sent in response to a Start Path message when one of the following two possible situations occurs: n has not found a valid path in its field of view, or the start configuration that has been included in the Start Path message is *not-valid*. A configuration c is defined as not-valid when one or more of the following conditions hold: c lays over an obstacle, c has already been evaluated by n during a previous path calculation, c can be connected to a previous partial path calculated by n. The first operation that a camera n does after receiving a Start Path message precisely consists in checking whether one of these conditions is satisfied. In case the configuration c is not valid, then n replies to m with a Local Failure message. After receiving this message, m sends a new Start Path message to a different neighbor laying on the same open edge on which the configuration c is positioned. If no alternative neighbors are present, camera m resumes the local path calculation from the status where it was before sending the configuration c.
  Through this process, when a camera does not find a local path, the system imple-

ments a *backtracking* strategy: the control is given back to the previous node which searches for alternative solutions.

- *Goal Found*: This message notifies the successful completion of the path planning process. The camera that calculates the final part of the path (reaching the final configuration) locally broadcasts to all its neighbors the Goal Found message, which is then flooded into the camera network through multi-hop message relay.

- *Global Failure*: This message notifies a camera of a system-level failure for the path planning process. The system issues a global failure when all the configurations have been explored but the robot camera network is not able to find, in a distributed fashion, any path that can feasibly connect the assigned start and end configurations. This means that a feasible solution does not exist given the characteristics of the calculated potential field and the selected search parameters (e.g., cell discretization, minimal rotation angle). A Global Failure message is generated first by the start camera, according to the following process. When a camera explores all the configurations in its field of view but has found no path that reaches the goal or exits from an open edge, the camera asserts a local failure, as described above, and sends to the previous camera in the path a Local Failure message. Since its first selection for path continuation was aborted, also this camera can now possibly incur in an analogous local failure while trying to find an alternative path continuation. This potential sequence of local failures can let the system backtrack from camera to camera along the path built so far, until it reaches the first camera of the sequence, the one that started the planning phase and that sees the start position. At this point no further backtracking is possible. Therefore, if also the start camera locally fails it means that the entire planning process has failed. In this case the Global Failure message is generated by the start camera and flooded in a multi-hop fashion throughout the camera network.

4.1 Heuristics

In the following two sections we describe a set of heuristics that we designed to address the issue of getting trapped in local minima during the path search process, as well as issues related to the computational and communication efficiency of the system, with the aim of improving its overall scalability.

4.1.1 Heuristics to avoid local minima

During potential diffusion, the single-map path planner presented in Chapter 3 can potentially get trapped in *local minima*. This is due to the fact that potential diffusion does not explicitly consider the dimensions of the moving object, while path calculation does it. Therefore, when the shortest route towards the final configuration includes a narrow passage, the potential field can diffuse through the passage and assign low potential values to the corresponding area. During the path calculation phase, the search algorithm explores the solution space expanding the search tree towards the areas with assigned low values for the potential field. In this way, the exploration process gets naturally attracted to the direction of the narrow passage and tries to establish a path through it. However, while the skeleton and the potential field were able to pass through the narrow passage, the

Algorithm 4.0.1 Local path planning and navigation algorithm executed at a camera node

```
1: M ← CalculateOccupancyMap()                                   /* Detect obstacles and build local map */
2: for all n ∈ Neighbors do
3:     CommunicateMyFoVSize(n)
4:     fov_n ← GetNeighborFoVSize(n)
5:     δ_n ← DetectNeighborPosition(n)
6:     α_n ← DetectNeighborOrientation(n)
7:     o ← OverlappingArea(n, δ_n, α_n, fov_n)
8: end for
9: CalulateSkeleton(M)
10: if (isVisible(Destination)) then                             /* Camera with final configuration in the FoV */
11:     DiffusePotential(M, Destination)                         /* Potential field diffusion with Destination as starting point */
12:     SendPotentialFieldMessageToNeighbors(M, N)               /* Send the potential field on the shared edges */
13: end if
14: while pot_new ← NewPotentialFieldMessagesReceived() do       /* Received updated potential field values pot_new from the neighbors */
15:     Diffuse potential(M, pot_new)                            /* Potential field diffusion with pot_new as starting points */
16:     SendPotentialFieldMessageToNeighbors(M, N)
17: end while
18: while true do
19:     WaitForMessageFromNeighbors()
20:     i ← ReadMessage(n_i)                                     /* n_i is the sender of message i */
21:     switch (i)
22:     case StartMessage:
23:         s_i ← ReadMessageContent(i)                          /* s_i = starting configuration received with message i */
24:         if (isValidStart(s_i)) then
25:             if (P ← CalculateLocalPath(s_i, M, o)) then      /* The algorithm has found a local path P */
26:                 if (P[lastPosition] = Destination) then
27:                     LocalBroadcastGoalFound(N)
28:                     go to: 56
29:                 else
30:                     s_j, n_j ← IdentifyNextNeighborInPath(P, N)   /* s_j = last position of the path P converted in the reference system of neighbor n_j */
31:                     SendStartPathMessage(s_j, n_j)
32:                 end if
33:                 PATHS ← StoreLocalPathInfo(P, n_i, n_j, openset)   /* openset = current status of the search */
34:             else
35:                 if (isTheStartPositionVisible() AND isEmpty(PATHS)) then
36:                     LocalBroadcastGlobalFailure(N)
37:                     exit
38:                 else
39:                     SendLocalFailureMessage(n_i)
40:                 end if
41:             end if
42:         else
43:             SendLocalFailureMessage(n_i)
44:         end if
45:     case LocalFailure:
46:         openset ← RestorePlanningStatus(PATHS[lastPosition])
47:         go to: 25
48:     case GoalFound:
49:         RelayMessage(i, N)
50:         go to: 56
51:     case GlobalFailure:
52:         RelayMessage(i, N)
53:         exit
54:     end switch
55: end while
56: k ← 1
57: if (isTheStartPositionVisible()) then
58:     go to: 62
59: end if
60: while NavigationCompleted do
61:     WaitForContinueNavigationMessage()
62:     NavigateTheRobot(PATHS[k])
63:     SendLocalNavigationCoontrolMessages(PATHS[k].next)
64:     k ← k + 1
65: end while
```

Figure 4.4: zePPeLIN pseudo-code: path planning process at a generic camera node.

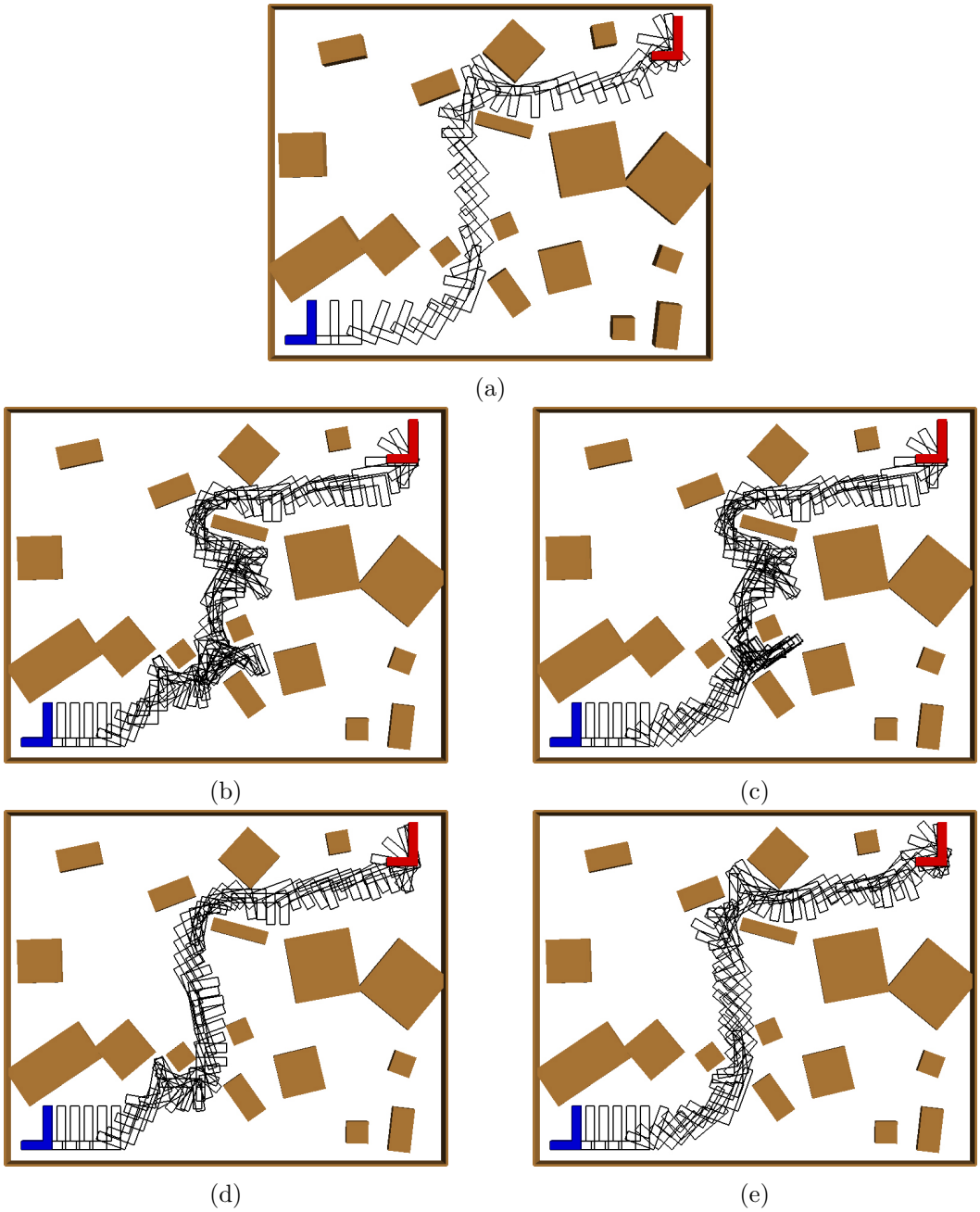


Figure 4.5: Comparison of paths. (a) Global planner. (b) zePPeLIN planner without heuristics. (c) Planner using skeleton pruning (with the parameter w_{sp} set to as the smallest dimension of the object). (d) Planner using narrow passage detection. (e) Planner using all the heuristics.

moving object has a shape and a dimension which might prevent its crossing. If this is the case, the process gets stuck exploring an area that has low potential values but which is, at the same time, too narrow for letting the moving object pass through. In other terms, the search process gets trapped in a local minimum. The algorithm deals with this issue by locally backtracking and exploring different alternative paths.

However, getting stuck in local minima and backtracking can have a significant negative impact on computational efficiency. Therefore, we propose two heuristics for minimizing the probability that this happens: *skeleton pruning* and *narrow passage detection*. The two heuristics can be implemented independently of each other and act in different phases of the process. The illustration of the effect of the two heuristics, compared to the path resulting without the application of the heuristics, is shown in Figure 4.5.

Skeleton pruning. The aim of this heuristic is to prune the skeleton during potential field diffusion in order to block passages narrower than a predefined width w_{sp}. In the experiments, we set w_{sp} to the width of the smallest dimension of the moving object. However, since width is not the only parameter defining whether an object can cross a passage or not (e.g., it depends also on the object's morphology), the heuristic does not guarantee the removal of all local minima (see Figure 4.6). On the other hand, setting w_{sp} higher than the smallest object dimension, or, more in general, too high, might result in the pruning of the majority of (or all) feasible paths.

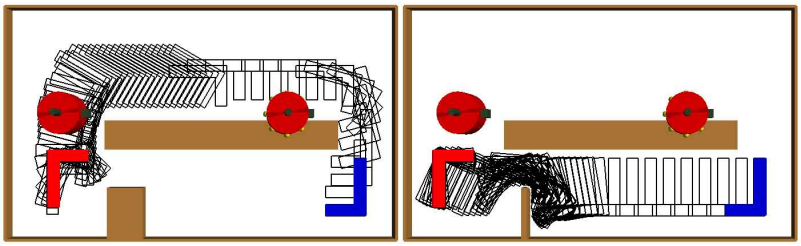


Figure 4.6: Width is not the only parameter defining whether an object can cross a passage or not. In the two figures, the narrow passages of the two similar scenarios have the same width. However the object with L shape can pass through the passage of the right figure, while cannot for the one of the left figure.

Narrow passage detection. This heuristic is executed *during path calculation*, when a camera detects a local minimum due to the presence of a too narrow passage. A narrow passage is detected when the following set of criteria is verified. Let x be the last visited configuration and b the visited configuration with the lowest value of $h(.)$. The heuristic checks the following conditions:

- $[h(x) > (h(b) + 10)]$: the search algorithm is exploring configurations with h value of one order of magnitude higher than the lowest visited. This means that the search algorithm is not exploring configurations that reduce the distance from the destination.

- $[\forall c_x \notin skeleton]$ All the control points c_x of the configuration x occupy cells that are not skeleton cells. Normally, the algorithm tends to calculate paths that follow the skeleton. If the last visited configuration x is not on the skeleton, it is sign of anomaly (i.e., a possible local minimum).

- $[dist(x, b) > size(O)]$ $dist(x, b)$ is the Euclidean distance between x and b, and $size(O)$ is the size of the largest dimension of the moving object O. This check aims to detect if the search algorithm has started to perform backtracking, or in

other words, if the algorithm is exploring different alternative paths that are distant from configuration b, which the closest to the destination till now.

- $[\exists c_{b^I} \notin freeSpace]$ Exists a control point c_{b^I} of the configuration b^I that collides with obstacles (it is not in the $freeSpace$). Configuration b^I is calculated starting from b and performing a 1-step translation in the direction of potential descent, i.e. in the direction to where the potential decreases. This means that the algorithm stopped to visit configurations with lower h value because an obstacle occludes the way.

When all the above criteria are verified, the camera has detected a narrow passage. At this point, the camera places a virtual obstacle over the cells belonging to the passage. The center $center_{np}$ of the passage is identified as the cell of b^I with lowest potential value. The algorithm places a virtual obstacle on $center_{np}$ and on all the cells in the range of $size(O)/2$ cells from $center_{np}$. Then, it triggers a new distributed potential field diffusion step that avoids in this way the passage, and, therefore, of being trapped in the associated local minimum.

4.1.2 Heuristics to improve efficiency

The following two heuristics, *blocked cells* and *loop avoidance*, aim, respectively, to reduce communications and improve path quality.

Blocked cells. The goal of this heuristic is to reduce communications between neighbors and to improve the speed of path calculation. When a robot camera n is not able to find a local solution (i.e., it has visited all possible configurations that are reachable from the starting point but no feasible solution exists), it sends a *Local Failure* message to the previous camera m in the path (see Chapter 4). Camera m then resumes path calculation from the last state reached before sending its path information to n. When m resumes its local path calculation, the search algorithm will explore new different configurations, selecting the best ones with respect to the value of $g(x)$. It is however possible that these new selected configurations are close in position to the configuration that was sent before, and which caused the generation of the Local failure message from n. That is, the algorithm might try to connect m's local path to the same neighbor camera n, sending each time a Start Message with attached a slightly different configuration. If n has no feasible exit configurations, this repeated process will result in the continual generation of Local failures at n, and in the consequent waste of time and generation of multiple messages between the two camera nodes.

The Blocked cells heuristic, aims to avoid these situations. The camera m that receives a Local Failure message from neighbor n labels the cells relative to the communicated (failed) configuration as *blocked cells*. In its future attempts, m does not try to send further Start Path messages with configurations laying on the blocked cells. In this way, after the first few trials, camera m can rapidly focus on totally different new areas, possibly considering different neighbors to proceed with path construction. The adoption of the heuristic has the drawback that feasible paths might get removed from the search process following a local failure.

Loop avoidance. If $(n_1, n_2, \ldots, n_k)$ is the sequence of cameras associated to the computed path, and $n = n_i = n_j$, for any $1 \leq i, j \leq k,\ i \neq j$, a *loop* is said to be present in the path if the two configurations entering n at steps i and j can be connected together within

n's local area. In this case, the sub-path between n_i and n_j can be safely removed. From an operational point of view this is performed in the following way. Given a path, a camera n checks whether a loop is present or not by controlling its local components of the path. If a loop is detected, n sends a *loop* message to its previous camera m in the path. After receiving the loop message, m deletes its local path and forwards the loop message to its preceding camera. The process is iterated until the loop message reaches again camera n, such that the loop is completely removed from the path. In order to avoid the re-creation of the loop, n perturbs its local potential field in the area where the previous local path ended. Then n resumes the path planning process.

4.2 Adaptation to changes in the environment

An important advantage of our distributed system is that it can locally and quickly detect and adapt to a change in the environment that might happen at any place and any time (e.g., a change in obstacles' position, or the appearing/disappearing of an obstacle). For instance, a planning system based on maps built by the moving object/robot itself (e.g., using SLAM techniques) cannot effectively cope with these situations since the sensory range is locally limited. On the other hand, a centralized system, even in presence of a small change, would correct the Voronoi skeleton, repeat the potential field diffusion and restart the path planning, which is a set of operations that altogether might require considerable resources and time. In contrast, in our distributed architecture, the system can effectively reduce re-initialization costs and time by replanning only a limited part of the path, through a process of *local adaptation to changes*.

In the case of detection of a change that blocks the current path, the camera n that controls the area where the change has happened tries to locally plan an alternative path by generating a new local potential field, with the constraint of maintaining fixed the original entrance and exit configurations. If the camera n succeeds in finding an alternative path, it does not inform its neighbors of the local change, and uses the new local path. Otherwise, n notifies the destination node (through multi-hop wireless communication) to trigger a new potential field diffusion process. The destination node decides either to repair the path (from n to the destination) or to recalculate the entire path (using as start configuration the current position of the navigating object). The decision for either alternative is taken in relation to the position of n in the sequence of nodes along the path. E.g., a local repair is issued when n is close to the final destination. In this case, if the repair process happens while the object is already performing path navigation, it can continue the navigation towards n, and when it will arrive at n a new path continuation toward the final destination will be already available.

4.2.1 Fault tolerance

Another possible change that can occur in the environment consists in the failure of one or more cameras (e.g., electrical failure, battery depletion). The zePPeLIN's fully distributed architecture can guarantee good levels of *fault tolerance*. In fact, in case a robot camera included in the path fails and stops to work, neighbor cameras can rapidly detect the problem (cameras keep sending each other short keep-alive messages) and trigger a repair process similar to the one described in the previous section. In this way, the network is able to find a new feasible path, if one exists, even in case of multiple camera failures. This capability is also observed in the experiments of Section 5.3, where for large errors in

relative camera positioning, systems with higher density of cameras (i.e., greater number of cameras in the same environment) show a a better ability to find feasible solutions compared to systems with lower densities. This is is due to the fact that when some cameras fail in practice (due to the large error), other cameras can take their place in the process.

Chapter 5

Results of simulation experiments

We studied the properties of the system through an extensive set of simulation experiments. The experiments have been designed with the objective of studying the following characteristics of the system: (i) the robustness to the alignment errors (i.e., wrong estimations of the neighbor's relative position and orientation), (ii) the effect of the heuristics on performance, (iii) the scalability in larger environments (keeping the density of cameras constant), and (iv) the scalability for increasing density of cameras (keeping the environment size constant), which corresponds to scalability of resources.

Performance metrics. The two main performance metrics we used for system evaluation are the *success ratio* and the *relative path length*.

The success ratio is the percentage of successful runs over the total number of executed runs. The success of a run is determined with a post-evaluation of the resulting final path. As we described in the previous chapters, due to alignment errors the final path might be composed of disconnected sub-paths (Figure 4.1). The evaluation of the path consists in locally connecting the sub-paths to each other and in verifying its feasibility. This is done running the planning algorithm with the starting and final configurations being respectively the last and the first configurations of two consecutive sub-paths. This evaluation permits to verify whether a solution has disconnected sub-paths that can be feasibly reconnected during the navigation phase or not. If this is not the case, the disconnection results in an infeasible path, and therefore in a global failure. The reconnection attempt is performed in a confined subregion, which is bounded by the field of view of the two cameras that have calculated the two sub-paths, and by a circular region with radius proportional to the amount of the alignment error (higher error, higher radius). This spatial constriction aims to find a local reconnection path which only includes a few roto-translations (i.e., a very short sub-path), but not the definition of a completely new and alternative path which connects the two disconnected sub-paths with a large sequence of movements and a long trajectory.According to this validation procedure, a path calculated by zePPeLIN is classified as *success, invalid*, or *failure*. Success means that the system has found a solution and all the sub-paths can be feasibly connected. Invalid means that the system has found a solution but the sub-paths cannot be connected. Failure means that the system has failed to find a potentially valid solution.

While the success rate in producing feasible paths is the first metric to assess the effectiveness of the zePPeLIN system, the length of the feasible paths also needs to be considered to assess its performance. Therefore, we considered as additional metric the ratio between the length of the calculated path (including reconnection paths) and the

length of the shortest path. The shortest path is calculated on the global map by a centralized planner without any Voronoi skeleton. That is, the algorithm does not try to stay as far as possible from the obstacles. In this case, we do not care of calculating a *safe* path which keeps a safe distance from the obstacles; rather the objective is to have the shortest possible path to be used as baseline reference. In the result plots showing relative path lengths we indicate also the length of the path calculated by the centralized global planner with an algorithm identical to the one used in the distributed planner (i.e., using the Voronoi skeleton), but using a single global map.

General experimental setup. The experiments in simulation have been run with a dedicated multi-process simulator developed for this study. The input for one simulation experiment is a set of three files including: environment description, camera network formation, and parametric properties. The environment description contains the set of obstacles on the ground with their positions and dimensions. The camera network formation file contains the list of all camera sensors, their positions, and their IP address. Parametric properties refer to the values of all the parameters that characterize the system (e.g., the discretization step, the unitary rotation step, the active heuristics, the standard deviation of the error in positional measurements, the communication range). A startup process reads the environment description and generates the image of the ground environment. Then, it reads the camera network file and launches an independent process for each camera passing as input: the portion of the ground image relative to the camera field of view, and the set of neighbors (relative position and IP address). The set of neighbors is calculated in relation to the position of the camera and its range of communication. The environment description also includes the start and final configurations of the moving object. All individual camera processes communicate with each other and collaboratively plan the path.

The experiments have been run on a machine with 2 AMD Opteron 6128 (8 cores each, 2 GHz, 2x 12 MB L2/L3 cache) and 16 GB RAM. The discretization of the environment for the purpose of defining unit movements amounts to 15 cells per meter, that is, cells with size of 6.7 cm. As unitary rotation we used $\theta = 15°$. The moving object has an "L" shape with two segments of same length equal to 50 cm. In the planning process, the object is modeled by three control points as shown in Figure 3.1b. The simulation experiments are based on automatically generated maps, which differ in the placement of the obstacles. The maps are generated by placing every square meter a rectangular obstacle with probability 0.5. Obstacle placement is performed by randomly selecting its position, orientation, and size (sampled between 20 cm and 1 m). The start and the final configurations are kept fixed for all the maps, respectively in the top-left corner and in the bottom-right corner.

In all the result plots, the performance of the system is evaluated against different levels of error. Since the error is stochastically generated, we execute multiple runs for every error level in order to improve the statistical significance of the results. For each error level we run 10 to 20 runs, depending on the experiment.

5.1 Robustness to relative position errors

In this section we present the results of simulation experiments designed to study the variation of the performance as a function of different levels of alignment error. We model the errors on relative position and orientation between cameras as two independent zero-mean Gaussian distributions with configurable standard deviation. In the experiments, we

vary the standard deviation of the error on one measure, while keeping the error for the other measure fixed to zero. In Figures 5.1 and 5.2, plots (a) and (b) shows the results for increasing angle errors, with the position error set to zero. The results for the reverse setup are shown in plots (c) and (d).

Experimental setup. For the experiments of Figures 5.1 and 5.2 we used 18 different maps of size 12×7 m^2 covered by a sensor network of 25 cameras, which have been deployed in the environment in a grid formation of 5×5. Each camera has a field of view of 3×2 m^2 of the ground, and the network has a topology such that each camera's field of view has a rectangular overlapping area with the neighbor's field of view of width equal to 75 cm.

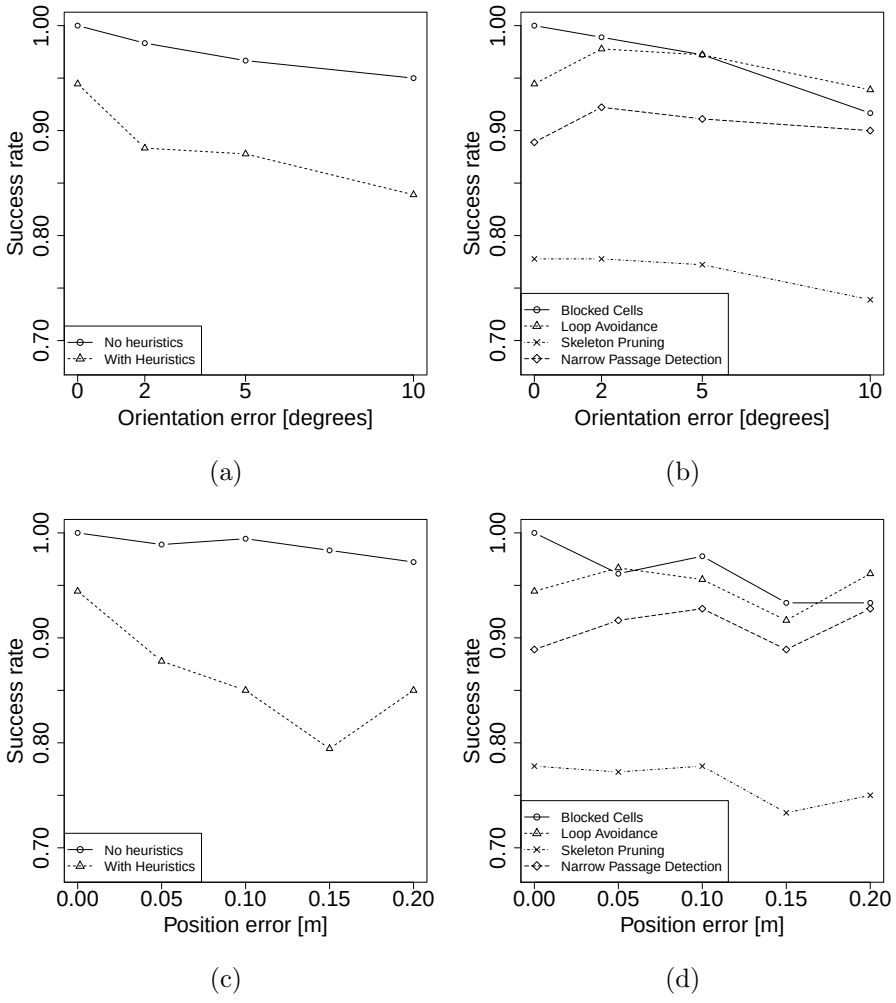


Figure 5.1: Success rate of the system with respect to alignment errors: comparison of different planners. (a) Planners with and without heuristics varying the error in the relative orientation. (b) Planners with the 4 heuristics active independently varying the error in the relative orientation. (c) Planners with and without heuristics varying the error in the relative position. (d) Planners with the 4 heuristics active independently varying the error in the relative position.

Results for robustness. Figure 5.1 shows that the success rate decreases with the increase of the alignment error. This is due to the fact that erroneous information of the overlapping area prevents the connection of partial paths during the planning phase.

Figure 5.2 shows the results for the relative path length metric. Also in this case the performances decreases for high level of alignment error: the length of the final path is longer, thus less efficient (in terms of time and energy consumption). This is partially due to the connection paths, since the final path is the sum of the calculated sub-paths and the connection paths. When the error is high, the possible disconnection between paths is larger and consequently also the connection paths are longer. This increases the length of the final path.

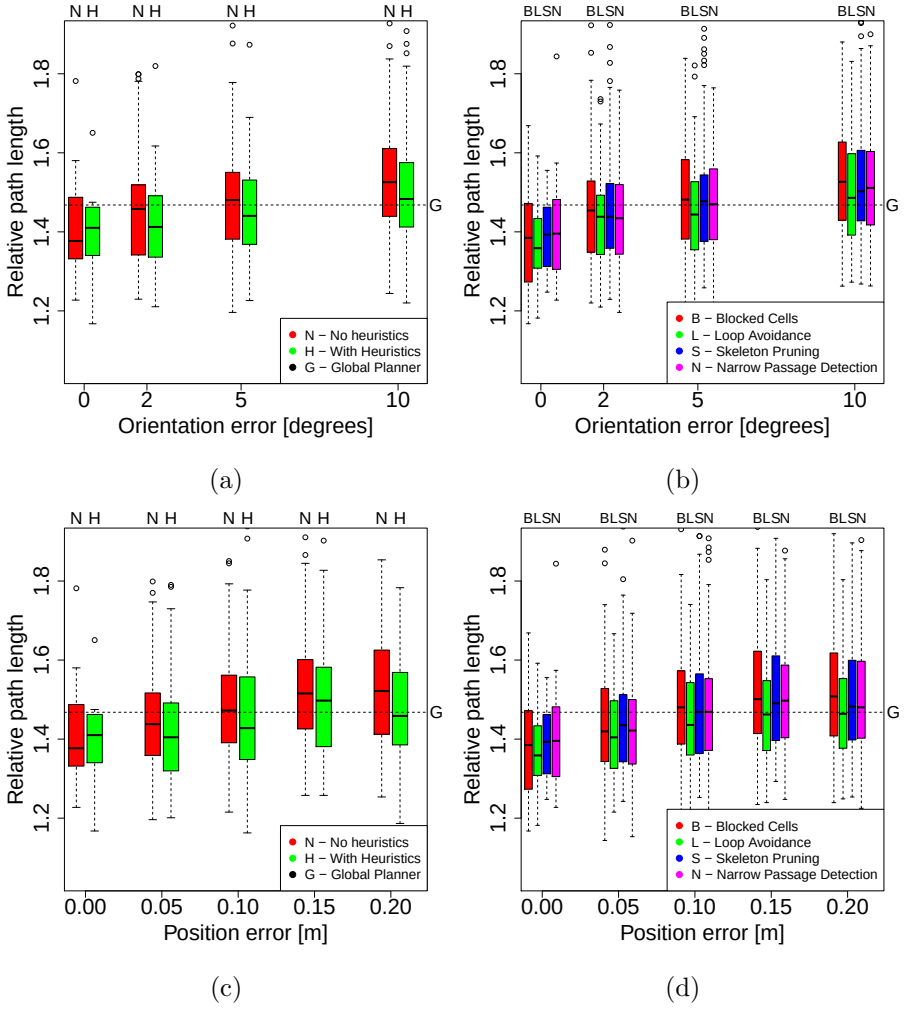


Figure 5.2: Relative path length for various levels of alignment error: comparison of different planners. (a) Planners with and without heuristics varying the error in the relative orientation. (b) Planners with the 4 heuristics active independently varying the error in the relative orientation. (c) Planners with and without heuristics varying the error in the relative position. (d) Planners with the 4 heuristics active independently varying the error in the relative position.

Results for the impact of the heuristics. In Figures 5.1 and 5.2 we plot the results for the different planners we propose. More specifically, we show the results of the planner with and without heuristics (on the left side) and the planners with each of the heuristic active independently (on the right side). The planner indicated in the figure as the 'planner with heuristics' included three heuristics: blocked cells, narrow passage detection, and loop avoidance. We excluded skeleton pruning because of its low success ratio performance.

A first consideration is the positive effect of the heuristics on the quality of the solutions. The planner with heuristics is able to calculate on average shorter paths than the planner without heuristics (Figure 5.2a and 5.2c).

However, this beneficial effect has the disadvantage of a reduction of the success rate (Figure 5.1). With the increase of the error level, the success rate decreases more rapidly for the planner with heuristics than for that without heuristics. As described in Section 4.1, when the planner gets stuck in a local minimum, it executes a backtracking strategy which is resource demanding. The planning process in a local minima free scenario does not get stuck and rapidly finds a solution. The heuristics reduce the effect of local minima and improve the efficiency of the process (time and messages) and of the solution (path length). However as drawback they reduce the solution space and feasible paths might get removed from the search process. The effect of each heuristic is discussed in separate paragraphs in the following.

The *skeleton pruning* heuristic aims to remove the local minima. It acts during the phase of skeleton generation: it closes all the passages narrower than the object size. However, the object has an L shape and it can thus overcome narrow passages with tight maneuvers between the obstacles. Therefore, in some environments this heuristic closes valid paths towards the destination. In this way, it does not let the algorithm to find any valid solution. For this reason, the skeleton pruning heuristic noticeably reduces the success rate (Figure 5.1b and 5.1d). As an advantage, this heuristics lets the planner operate in a scenario free of local minima, so that the process can rapidly converge to the solution without getting trapped. The effect is a more efficient planning process which avoids backtracking, and produces shorter paths (Figure 5.2b and 5.2d), has a quicker execution (Table 5.1), and generates a lower number of messages (Table 5.2).

The *narrow passage detection* heuristic is designed to remove local minima and their negative effects. It acts during the path calculation phase, and aims to block a passage only when the planner has realized that it is effectively not feasible. This heuristic improves efficacy, in terms of path quality (Figure 5.2b and 5.2d) and efficiency, regarding execution time (Table 5.1), producing at the same time less reduction of the success ratio compared to skeleton pruning, the other heuristic for local minima. A side effect of this heuristic is the higher number of messages it generates (Table 5.2). This is due to the fact that, when a camera detects a local minimum, it triggers a new potential field diffusion phase for the whole network.

The *blocked cells* heuristics aims to reduce the exchange of messages between neighbors. Table 5.2 shows that the heuristic succeeds in this purpose. However, similarly to the other heuristics, it has a lower success rate.

The *loop avoidance* heuristic aims to remove loops in the final resulting paths (see Section 4.1). That is, it aims to improve the quality of the resulting paths. Its effect is visible in the plots of Figure 5.2b and 5.2d: the when the loop avoidance heuristics is active, the system can calculate the paths with the shortest length.

Pos. Err.	Orient. Err.	Normal	BC	LA	SkP	NPD	All Heur.
0	0	6.79	8.08	6.91	2.91	2.89	2.52
0	2	6.49	9.95	5.81	3.27	4.79	4.33
0	5	9.61	14.29	7.47	2.74	6.23	5.34
0	10	12.3	15.96	7.89	4.08	8.92	9.23
0.05	0	6.27	7.04	4.95	1.86	4.97	4.5
0.1	0	6.61	9.85	5.84	1.94	5.9	7.25
0.15	0	15.91	12.37	11.66	4.73	9.45	15.29
0.2	0	10.45	16.63	11.6	3.08	6.63	11.42

Table 5.1: Normal: without heuristics. BC: Blocked cells. LA : Loop avoidance. SkP: Skeleton pruning. NPD: Narrow passage detection. All Heuristics: BC + LA + NPD. This table shows the execution time (in seconds) of the planners with and without heuristics with various levels of error. The value is the median of the distribution.

Position Error	Orientation Error	Normal	BC	LA	SkP	NPD	All Heur.
0	0	88.38	85.36	85.48	83.16	97.18	86.52
0	2	86.36	85	85.22	84.82	160.58	87.64
0	5	86.86	85.04	85.28	85.44	131.28	89.44
0	10	88.52	85.26	86.08	85.88	164.9	167.36
0.05	0	86.88	85.54	85.72	85.08	90.84	89.88
0.1	0	87.44	86.02	86.3	85.5	138.76	171.06
0.15	0	91.28	86	87.12	87.5	168.08	171
0.2	0	89.92	86.16	87.08	86.8	145.68	174.54

Table 5.2: Number of messages per camera for the planners with and without heuristics with various levels of error. The reported value is the median of the distribution. Normal: without heuristics; BC: Blocked cells; LA : Loop avoidance; SkP: Skeleton pruning; NPD: Narrow passage detection; All Heuristics: BC + LA + NPD.

5.2 Scalability to environment size

In this section we present the results of simulation experiments designed to study how the system scales its performance when environment size increases. We consider environments whose size is respectively two and three times larger than the size of a baseline environment.

Experimental setup. The experiments have been run considering 30 different maps with the following sizes: 10 maps of 12×7 m^2 (=84 m^2), 10 maps of 12×14 m^2 (=168 m^2), and 10 maps 12×21 m^2 (=252 m^2). Since environment size increases, also the camera network has to be increased correspondingly to cover the entire area. In order to keep a minimum overlapping area of the fields of view of 75 cm, the network scales to 5×11 cameras for the environments of double size, and to 5×17 cameras for the triple size cases.

Results for efficacy and efficiency. Figure 5.3 shows the success ratio and the relative path length for different environment sizes. Both the effectiveness (success ratio) and the efficiency (relative path length) remain constant for larger environments. These results are indicator of scalability for increasing environment size. Success rate values oscillate in a range between 100% and 93%. The plots in Figure 5.3a and 5.3b are jagged due to the strong heterogeneity of the scenarios and the random generation of errors. However, it is worth to notice, that the percentage of success remains over 93% also for high levels of error.

Boxplots in Figure 5.3c and 5.3d show the results in terms of relative path length. Also for this metric the results confirm the scalability of the approach. While for environments of 84 m^2 and 252 m^2 the relative path length values are very similar, for environment of 168 m^2 the values are slightly higher (about 5%-10% more) and with higher variance. As discussed above, this difference is due to the strong heterogeneity of the scenarios, which are randomly generated.

Figure 5.4 shows the *efficiency* of the system in terms of communication overhead. We study how the number of messages varies scaling up the environment and the network. In order to perform a fair comparison of the efficiency performance in the different environments, the reported values are normalized with respect to the average number of neighbors, which varies for the different topologies. In fact, the cameras on the edges of the network have a lower number of neighbors than the cameras in the middle. In our scenario, the cameras on the corner have 2 neighbors, the cameras on the edge have 3, and the cameras in the middle have 4. To make the measure as independent as possible from this side effect, the reported values for the number of communicated messages are calculated as follow:

$$Messages = \frac{1}{kN} \sum_{n}^{N} p_n, \tag{5.1}$$

where p_n is the number of messages received by camera n, N is the total number of cameras, and k is the average number of neighbors in the network (for example in the case of a network 5×5, $k = 3.2$).

The number of messages increases with the network size. The cameras communicate the most of the messages during the phase of potential field diffusion. This is due to the architecture of the system which waits for a fixed number of equal messages. When a camera receives from its neighbors the same potential field message for more than 21 times, the potential is considered as definitive, and the camera terminates the diffusion phase. When the network size increases, the diffusion process takes more time, and as

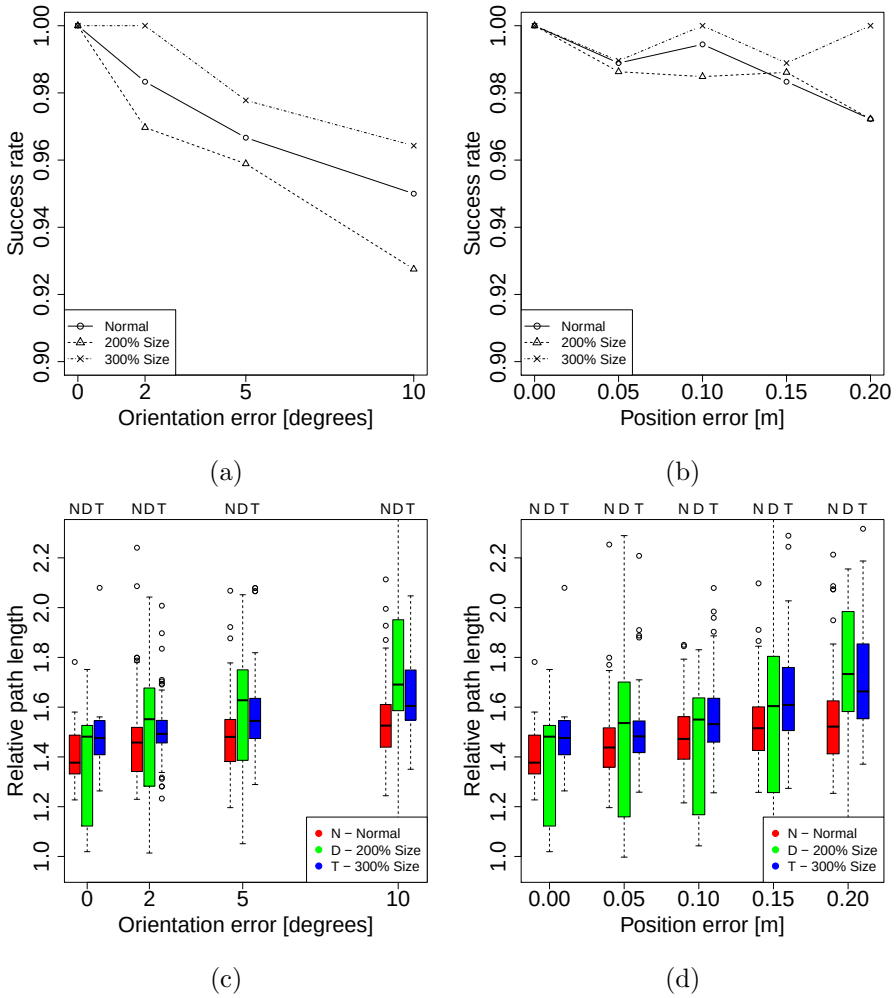
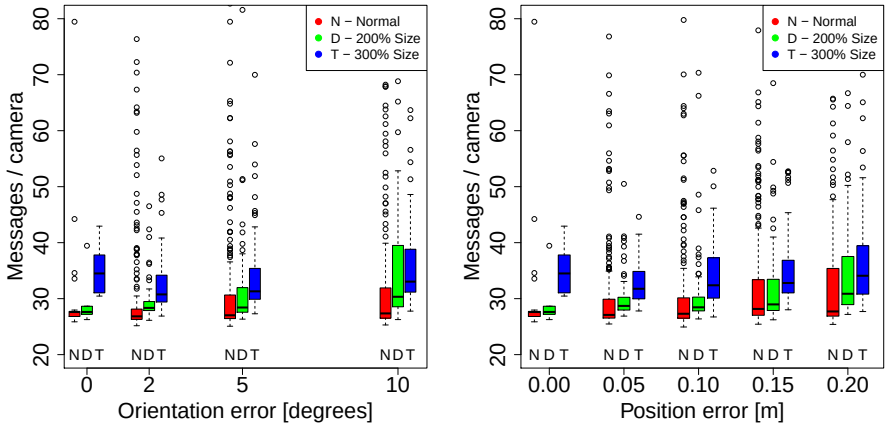


Figure 5.3: Scalability for environments with increasing size. (a) Success rate varying the error in the relative orientation. (b) Success rate varying the error in the relative position. (c) Relative path length varying the error in the relative orientation. (d) Relative path length varying the error in the relative position.

a consequence the number of messages (often repeated) increases. In our design, the bandwidth for messages was not an hard constraints, thus the message repetition solution has been implemented for simplicity. However, in case it is needed, communication during the diffusion phase can be optimized avoiding to communicate repeatedly the same message and designing the phase transition with another parameter (e.g., a temporal threshold from the last received message). In fact, the number of messages communicated during the other phases is very limited and remains constant for all the studied scenarios in a number ranging from 3 to 10 messages per camera.

5.3 Scalability of resources

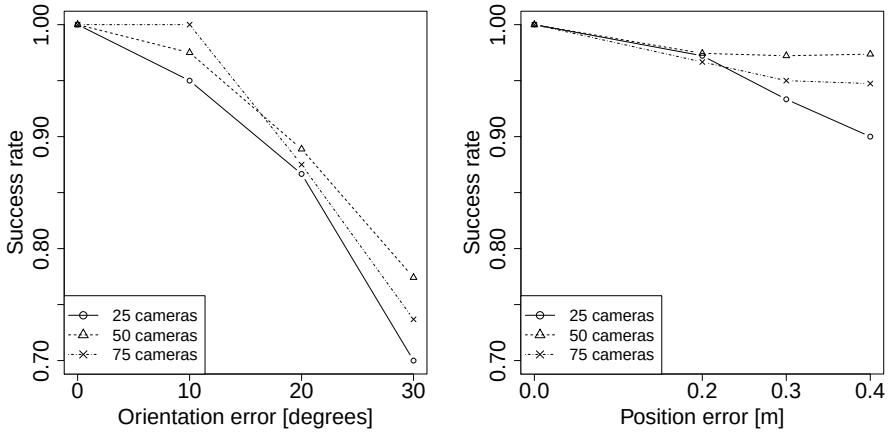
The set of experiments of this section shows how the performance of the system varies when increasing the number of cameras, while maintaining constant the size of the environment.



(a) (b)

Figure 5.4: Scalability for environment of larger size: efficiency measures. (a) Messages per camera varying the error in the relative orientation. (b) Messages per camera varying the error in the relative position.

Experimental setup. The experiment setup is the same as in Section 5.1. We ran experiments over 10 maps, and for every map we increased the density of the camera network. The initial network is composed of a grid of 5×5 cameras, which is the minimum number to fully cover the entire environment. We then increased the density by adding, in a first test case 25 cameras (+100% = double density), and in a second test case 50 cameras (+200% = triple density). In both cases, the cameras were assigned with randomly selected position and orientation. For each of the three setups we ran experiments varying the level of error on position and orientation and executing 10 runs for each error level. This means that every point in the plots of Figure 5.5 corresponds to the average of $10 \cdot 10 = 100$ runs.



(a) (b)

Figure 5.5: Scalability for higher number of cameras keeping the environment size constant. (a) Success rate varying the error in relative orientation. (a) Success rate varying the error in relative position.

Results. For high levels of error, the performance of zePPeLIN decreases: Figure 5.5 shows the success rate as a function of alignment errors. These errors, in some cases, prevent connections between sub-paths in the overlapping area; therefore for high levels of errors the process more often fails to find a valid path. This negative effect can be reduced by increasing the number of cameras in the sensor network (Figure 5.5). With the standard sensor network formation 5×5, the overlapping area has a width of 75 cm and the moving object has a length of 50 cm. This formation has a limited margin of error, which allows the system to cope effectively only with low levels of error (success rate over 95% (first dot of plots in Figure 5.5). A more dense network has wider overlapping areas, which is aspect that allows the system to effectively plan valid paths more often and even for high levels of errors.

The system improves its performances in response to an increase of the resources in the system (i.e., the number of cameras in the sensor network). In zePPeLIN, the increase of resources is eased by a distributed, scalable and flexible architecture, which is designed to allow the user to add new cameras without any need to modify, update or setup the algorithm.

Chapter 6

Real robot experiments

We completed the experimental evaluation of zePPeLIN with a set of experiments with real cameras and ground robots. In these experiments, zePPeLIN executes all planning and navigation phases: it detects moving robot position, it calculates the path from the current position to the given destination, and provides the instruction to navigate the ground robots in the environment. The setup adopted for the real world experiments and the results are described in the rest of the chapter.

The environment. The ground robot moves in an area of 33 m^2 where the gray floor is occluded by red obstacles. This color configuration has been selected for easing the obstacle detection since it is not the main focus of the work. A sample image of the arena is shown in Figure 6.1 (Left).

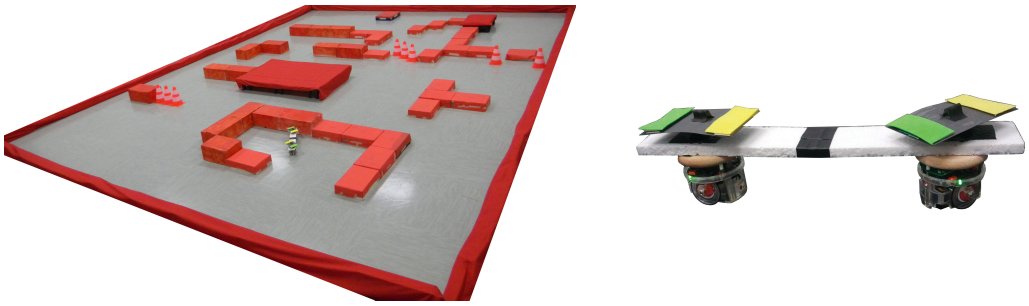


Figure 6.1: Scenario for real robot experiments. (Left) An example arena. The moving object has to move in the arena avoiding collision with the red obstacles. (Right) The moving object that follows camera instructions, which is implemented as a set of 2 e-puck robots interconnected by a rigid structure.

The camera network. The robotic camera network is implemented with a set of 4 cameras, fixed at the ceiling, pointing to the ground and connected to different computers. Each camera is controlled by an independent process, which cooperates and communicates wireless with the other processes via UPD sockets. The cameras are placed at an height of 3 m and have a field of view of 4.56×3.06 meters with an image resolution of 640×480. In our experimental setup we used normal cameras, that cannot autonomously estimate their relative position with respect to the neighbor cameras. For overcoming this issue, for each camera an input configuration file specifies the IP addresses and the relative positions of

its neighbors. The cameras are deployed in a rectangular formation, such as each camera has an overlapping field of view with two neighbor cameras. All the cameras have the same orientation, relative distances of 3 m (with the neighbor on the x axis) and 2.38 m (with the neighbor on the y axis), and a communication range limited to 3.5 m. The error on the measures of the relative distances and orientation is of the order of 0.1 m and 10° respectively.

The moving robot. The holonomic object moving on the ground is implemented as a set of two non-holonomic robots, the *e-puck* [17], interconnected by a rigid structure (see Figure 6.1, right). Each robot can freely rotate on place, while straight direction movements are constrained by the rigid structure and must be executed in coordination between the two robot. In this way, the two robots form an object with a relatively large shape, which is able to rotate and move in any direction. As discussed in Chapter 3, the path planning algorithm models a moving object by using control points. In this case, the control points corresponds to the center of the two e-pucks. Each of the two e-pucks has a colored patch applied on its top, in order to allow the cameras to track their positions and orientations.

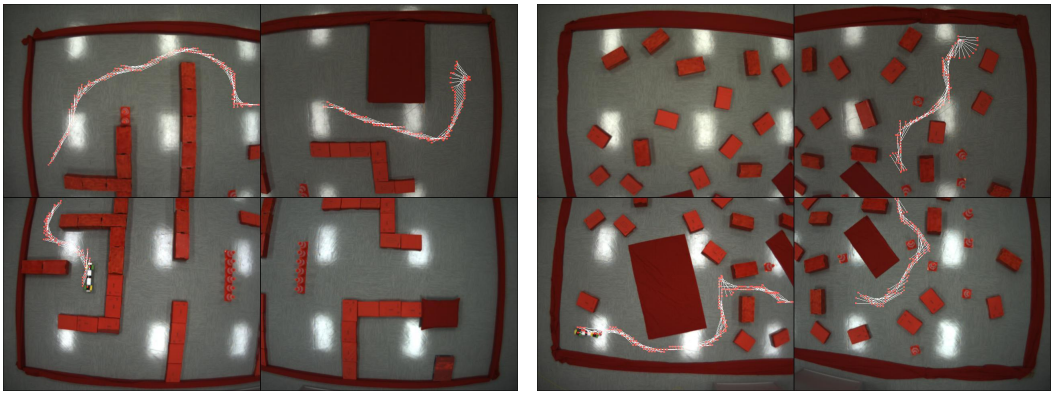
The path planning process. Each camera is controlled by an independent process. Trough a graphical interface, the user connects to a desired camera and specifies the final position and orientation of the object. The camera system autonomously detects the obstacles and the current position of the moving robot in the environment. With this information, the system calculates the path from the start to the final configuration in a distributed way, where each camera operates only on its local field of view. The resulting path is the ordered sequence of local configurations (i.e., an ordered sequence of roto-translations), where each camera holds the sequence relative to its field of view.

The navigation in the environment. Once the path is defined, the system starts the navigation phase. The navigation control process c_s connected to the camera s that has the moving robot in its field of view starts the navigation phase. Using the camera, process c_s tracks the current position of the two e-pucks and sends them via Bluetooth two independent messages with the information of the relative movement to be performed, that can be either a rotation in place or a translation. Each e-puck, when it completes the required movement, sends a notification message back to c_s. Iteratively, using the current position of the two robots and the next configuration to be reached, as specified by the locally planned path, the control process calculates and sends the relative movement that each robot has to perform next. In this way the camera-robot system operates in a closed loop, such that it is able to correct possible path implementation errors.

When process c_s navigates the robots to the last position of the its local path p_s, it communicates with the navigation control process c_n of the next camera n in the path in order to let it take control and continue robot navigation. It might happen that the e-puck system completely enters the field of view of n before the execution of path p_s is completed (e.g., because of little errors in navigation, or because of a relatively large overlapping between the fields of view). When this happens, since process c_n has better information than c_s regarding its local environment, it is appropriate that n takes control. Therefore, n tries to plan a local path p_{conn} which connects the current position of the robots to the closest point of the calculated path p_n. If a connection is found, it means that n can locally navigate the robots to a configuration in p_n and continue the navigation on p_n as expected.

In this case, the process n sends a messages to process c_s requesting the control. On the reception of this message, c_s interrupts the navigation and hands the control to c_n.

The process described for cameras s and n is then iterated between all cameras involved in the planned path, until the moving robots reach the desired destination. A few sample videos showing path calculation and path navigation are available at the supplementary page[1]. The videos are shown three times faster than reality. In the videos, when a camera passes the navigation control to the next camera, a connection path is calculated, which is drawn in yellow. A camera icon shows which camera is taking control. Every time the control is passed from one camera to the next one the two robots rotate in place before starting to move. This is due to the following reason. In order to make the tracking process simple and scalable, the same color patch is put on the top of the two e-pucks, such that there is no need to create new custom patches if new robots are added to the multi-robot structure. However, this way of proceeding has the drawback that the tracking system has no way to to distinguish between the two e-pucks, which is needed because each e-puck has to perform different movements and the control process has to send different messages to each of them. Therefore, before starting to send instructions, the control process sends, in sequence, a message to each one of the known addresses of the two e-pucks, asking to perform a rotation in place. This allows the control process to associate each robot to its wireless address.



(a) Structured environment. (b) Randomly displaced obstacles.

Figure 6.2: Composite screen-shot of the four fields of view of the four cameras in two different experimental setups. The planned path (in red the control points and in white the stylized object) is shown. In this example, the information of the final path is segmented in three partial paths distributed among the cameras.

Experimental results. We performed experiments on 20 different maps: in 14 of them the environment is relatively well structured, with the obstacles placed to form straight walls (see Figure 6.2a for an example), while in the remaining 6 the obstacles are deployed completely randomly (see Figure 6.2b for a sample scenario). In all the experiments, the system has been able to calculate valid paths no longer than 150% of the optimal shortest path, and 115% of the path calculated by the global planner. Figures 6.2a and 6.2b show the composite screen-shot of the 4 fields of view. Planning time is relatively low, for

[1]The videos are available at `http://iridia.ulb.ac.be/supp/IridiaSupp2012-013/index.html`

example the planning time for the paths shown in Figures 6.2a and 6.2b is respectively of 4.6 s 7.2 s.

From the mentioned sample videos, it is possible to appreciate that the robots implementing the instructions received from the overhead camera are able to follow the calculated path with good precision. The system is able to effectively correct local actuation errors through the continuous tracking of the positions and recalculation of the next relative movements to communicate.

Chapter 7

Related work

Path planning is a fundamental problem in mobile robotics and for this reason it has been extensively studied, mainly considering centralized and single-robot approaches. An overview of the established work in this respect can be found in [13, 5, 14]. However, recently, there is increasing interest for distributed approaches in robotics, and this applies also to path planning. Work on distributed path planning can be roughly grouped in four main research threads: (i) on-board multi-robot path planning, (ii) path planning assisted by a sensor network, (iii) swarm approaches, and (iv) parallelism of computation. Our work belongs to the second thread. in the following we briefly discuss some of the most prominent approaches for all the considered threads of research, with a particular emphasis on the second one.

On-board multi-robot path planning studies [9, 2, 18, 21] propose distributed planning and navigation algorithms for letting a group of robots moving in the environment without colliding with the obstacles and with each other. Each work is usually based on different assumptions and problem statements, which also makes relatively difficult to draw proper comparisons among the different approaches. In [9, 18] the authors investigate how the multi-robot system behaves varying the noise in sensor measurements. In [23] the goal is to keep a predefined formation of the group of robots while moving towards a given destination. In all these works the algorithm runs on-board of the robot, thus the path is planned directly by the robot which then implements it. Robots need to know the map of the environment. This information can be either given as input or the robots can locally sense and explore the environment and generate a map on the fly (e.g., using a SLAM technique). Both approaches are however not able to deal robustly with large and dynamic environments, which, on the other hand, are the class of environments we target with our system.

Path planning assisted by a sensor network has been considered in a number of works in the last decade [16, 1, 15, 6, 3, 20, 30, 7]. In the considered scenarios, the moving robot is not required to be equipped with sophisticated devices and powerful CPUs in order to perceive the environment, build a representation of it (i.e., a map), and calculate a motion plan. Instead, the robot is guided by a distributed sensor network deployed in the environment. Compared to the single robot, the sensor network can exploit a better and distributed point of view, can rely on a much wider coverage of the environment, and can quickly react to changes in the environment or in the network (even if these happens at locations distant from the current position of the moving robot). In these works based on the use of an external sensor networks, the resulting path is commonly defined as the sequence of sensors that the moving robot has to visit. Sensors act as routers and compute

only the high level plan (i.e., the direction towards the next sensor). When entering the communication range of a sensor, a robot gets the necessary instruction to proceed with the plan. The robot is in charge of planning and implement the precise sequence of roto-translations for moving toward the next router while maneuvering between the obstacles and other moving robots. The planned path consists of a sequence of nodes. Obstacles in the environment are usually not explicitly considered, assuming that there is always a valid path between two neighbor sensors and the moving robot is able to effectively sense the environment and compute local motion planning. The path planning problem is therefore reduced to compute the shortest path on a graph, where the sensors are the nodes of the graph and some notion of neighborhood (e.g., wireless range) defines that two nodes are connected by an arc that can be navigated by the robots. In this same context, some works have studied the path planning problem considering additional constraints such as: avoidance of dangerous areas [15, 3], collision-free trajectories for multi-robot systems [16], presence of different terrain surfaces (which results in different motion speeds) [30]. The algorithm presented in [31] makes use of a sensor network with complex spatial sensing capabilities. The authors of the paper propose the Distributed Probabilistic Roadmaps algorithm where each sensor calculates a part of the path in the area in its sensing range and joins its sub-path with the one of its neighbors in a shared area of overlapping sensing. The resulting path is the sequence of the via-points that allow to avoid collisions with obstacles and to safely reach the next sensor/router. This study presents strong similarities with our work: the generated path is sufficiently detailed and takes in account the size and the shape of the moving robot. However, the authors do not report any study on the impact of measurement errors for the overlapping areas, which is one of the focus of our work being this a sort of intrinsic aspect of distributed sensory and planning systems.

Swarm robotics systems are characterized by large number of robots and the use of local communications and self-organized cooperation. Works in literature exploit these aspects to perform swarm-level path planning. In [22] the system assumes that a dedicated sensor network is deployed in the environment which navigates the swarm, similarly as it happens in the case of the works mentioned above based on the use of external sensor networks. In [19] the algorithm exploits the same robots in the swarm for creating a chain of static landmarks to navigate the other robots in the environment. In [10, 4, 8] the system does not allocate (and fix) any specific resource for building a dedicated sensor network, but the swarm itself is a mobile sensor network. Each robot while moving in the environment acts as landmark communicating information for the localization of the local neighbors and at the same time receives information from the neighbors. In all these works, both the sensors and/or the robots do not calculate the precise path through a cluttered environment, but act as high level routers which direct the moving robots to the next sensor/robot communicating only the direction. The final path consists of the sequence of sensors to be visited, which is very different from our work where the resulting path is the precise sequence of roto-translations to be performed to move through a potentially highly cluttered environment.

Parallelism of computation consists in distributing the planning computation on different processors for increasing the overall computational speed of the system [12, 27]. In these works, differently from ours, the computing nodes have a global knowledge of the environment, and the issue is how to optimize the division of computation. A different approach to parallelism is taken in [21], where the authors cast a multi-robot system as a distributed multi-computer, with each robot calculating a (possibly different) solution based on a rapidly-exploring random tree technique which is shared with the other robots

according to communication conditions. Using this approach, the algorithm is meant to exploit perfect communication and have gracefully performance decline otherwise.

Chapter 8

Conclusions and future work

We have proposed zePPeLIN, a novel system for distributed path planning in large, cluttered, and dynamic areas. The system is based on a fully distributed architecture, in which a swarm of flying robots is deployed in the environment and forms a distributed camera network. Each robot camera only has a partial, top view of the ground environment where the object needs to move from the initial to the final configuration. The system of robots solves the path planning problem cooperatively, through local calculations of potential field and Voronoi skeleton, and wireless message exchanges. A number of heuristics have been proposed to enhance both system's efficiency and effectiveness. The system has also a built-in component to deal with dynamic changes (e.g., appearance/disappearance of an obstacle, failure of a camera).

In simulation, we performed an extensive analysis of the performance and of the heuristics, and we also validate the system through a series of experiments using real camera devices and moving robots. Compared to systems with single cameras or centralized computations, our fully distributed approach is more scalable, flexible, and robust. However, it introduces efficiency issues and sensory errors. In particular, we studied the impact on performance of the alignment errors between the fields of view of neighbor cameras; a type of error which we consider as intrinsic to any distributed system of local, partially overlapping, maps.

The experimental results show that, as expected, the system performance degrades with the increase of the alignment error. However, only for very large errors, the drop in performance becomes significant. Errors on distance and angle, results in slightly different performance drops, with the error on the angle having a larger impact in this respect. The impact of the heuristics on performance is, in general, of improving quality (path length) and efficiency (computation time, communication overhead), but reducing, at the same time the, the success rate in finding feasible paths. The system shows relatively good scalability in terms of number and density of cameras.

Since in principle the zePPeLIN distributed algorithm can be used with any networked system of devices, given that each device is capable to build a local map of the environment in its 'field of view' (e.g., a kinect), as a future work we intend to explore the use of different devices instead of video cameras. Moreover, we intend to extend the system adding general mechanisms to manage, and possibly reduce, the overlapping errors. Additional tests will consider the inclusion of dynamic obstacles (e.g., including humans or moving robots) and larger environments.

Bibliography

[1] M. Batalin, M. Hattig, and G. Sukhatme. Mobile robot navigation using a sensor network. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 636–642. IEEE Computer Society Press, Los Alamitos, CA, April 2004.

[2] S. Bhattacharya, M. Likhachev, and V. Kumar. Multi-agent path planning with multiple tasks and distance constraints. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE Computer Society Press, Los Alamitos, CA, 3-8 May 2010.

[3] C. Buragohain, D. Agrawal, and S. Suri. Distributed navigation algorithms for sensor networks. In *Proceedings of IEEE INFOCOM*. IEEE Computer Society Press, Los Alamitos, CA, 2006.

[4] A. Campo, Á. Gutiérrez, S. Nouyan, C. Pinciroli, V. Longchamp, S. Garnier, and M. Dorigo. Artificial pheromone for path selection by a foraging swarm of robots. *Biological Cybernetics*, 103(5):339–352, 2010.

[5] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, 2005.

[6] P. Corke, R. Peterson, and D. Rus. Localization and navigation assisted by cooperating networked sensors and robots. *The International Journal of Robotics Research*, 24(9):771–786, 2005.

[7] F. Ducatelle, G. A. Di Caro, C. Pinciroli, and L. Gambardella. Self-organised cooperation between robotic swarms. *Swarm Intelligence*, 5(2):73–96, 2011.

[8] F. Ducatelle, G. A. Di Caro, C. Pinciroli, F. Mondada, and L. Gambardella. Communication assisted navigation in robotic swarms: self-organization and cooperation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4981–4988. IEEE Computer Society Press, Los Alamitos, CA, 2011.

[9] A. Fridman, S. Weber, V. Kumar, and M. Kam. Distributed path planning for connectivity under uncertainty by ant colony optimization. In *Proceedings of the American Control Conference*, pages 1952 –1958. IEEE Computer Society Press, Los Alamitos, CA, 2008.

[10] Á. Gutiérrez, A. Campo, F. Monasterio-Huelin, L. Magdalena, and M. Dorigo. Collective decision-making based on social odometry. *Neural Computing and Applications*, 19(6):807–823, 2010.

[11] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[12] D. Henrich. Fast motion planning by parallel processing – Review. *Journal of Intelligent and Robotic Systems*, 20:45–69, 1997.

[13] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic, 1991.

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

[15] Q. Li and D. Rus. Navigation protocols in sensor networks. *ACM Transactions on Sensor Networks*, 1:3–35, 2005.

[16] R. Luna, A. Oyama, and K. E. Bekris. Network-guided multi-robot path planning for resource-constrained planetary rovers. In *Proceedings of the 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*. IEEE Computer Society Press, Los Alamitos, CA, 2010.

[17] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, volume 1, pages 59–65. IPCB: Instituto Politecnico de Castelo Branco, Portugal, 2009.

[18] Y. Mostofi. Decentralized communication-aware motion planning in mobile networks: An information-gain approach. *Journal of Intelligent and Robotic Systems*, 56(1-2):233–256, 2009.

[19] S. Nouyan, A. Campo, and M. Dorigo. Path formation in a robot swarm. *Swarm Intelligence*, 2(1):1–23, 2008.

[20] K. O'Hara, V. Bigio, S. Whitt, D. Walker, and T. Balch. Evaluation of a large scale pervasive embedded network for robot path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE Computer Society Press, Los Alamitos, CA, 2006.

[21] M. Otte and N. Correll. Any-Com multi-robot path-planning: Multi-robot coordination under communication constraints. In *Proceedings of the International Symposium on Experimental Robotics*. 2010.

[22] D. Payton, M. Daily, R. Estowski, M. Howard, and C. Lee. Pheromone robotics. *Autonomous Robots*, 11(3):319–324, 2001.

[23] G. A. Pereira, A. K. Das, V. Kumar, and M. Campos. Decentralized motion planning for multiple robots subject to sensing and communication constraints. In *Proceedings of the Second Multi-Robot Systems Workshop*, pages 267–278. 2003.

[24] J. Roberts, T. Stirling, J.-C. Zufferey, and D. Floreano. Quadrotor using minimal sensing for autonomous indoor flight. In *Proceedings of the European Micro Air Vehicle Conference and Flight Competition (EMAV)*. 2007.

[25] J. Roberts, T. Stirling, J.-C. Zufferey, and D. Floreano. 2.5D infrared range and bearing system for collective robotics. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Computer Society Press, Los Alamitos, CA, 2009.

[26] T. Stirling, S. Wischmann, and D. Floreano. Energy-efficient indoor search by swarms of simulated flying robots without global information. *Swarm Intelligence*, 4(2):117–143, 2010.

[27] B. Taati, M. Greenspan, and K. Gupta. A dynamic load-balancing parallel search for enumerative robot path planning. *Journal of Intelligent and Robotic Systems*, 47:55–85, 2006.

[28] O. Takahashi and R.J. Shilling. Motion planning in a plane using generalized Voronoi diagrams. *IEEE Transactions on Robotics and Automation*, 5(2):143–150, 1989.

[29] A. C. Thompson. *Minkowski Geometry*. Cambridge University Press, Dalhousie University, Nova Scotia, Canada, 1996.

[30] C. M. Vigorito. Distributed path planning for mobile robots using a swarm of interacting reinforcement learners. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, New York, NY, USA, 2007.

[31] Z. Yao and K. Gupta. Distributed roadmaps for robot navigation in sensor networks. *IEEE Transactions on Robotics*, 27(5):997–1004, 2011.