International Journal of Advanced Robotic Systems

OPEN ACCESS
ARTICLE

zePPeLIN: Distributed Path Planning Using an Overhead Camera Network

Invited Feature Article

Andreagiovanni Reina[1,*], Luca Maria Gambardella[2], Marco Dorigo[1] and Gianni A. Di Caro[2]

1 IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
2 Dalle Molle Institute for Artificial Intelligence (IDSIA), Lugano, Switzerland
* Corresponding author E-mail: areina@ulb.ac.be

Received 14 Mar 2014; Accepted 29 May 2014

DOI: 10.5772/58748

Abstract We introduce zePPeLIN, a distributed system designed to address the challenges of path planning in large, cluttered, dynamic environments. The objective is to define a sequence of instructions to precisely move a ground object (e.g., a mobile robot) from an initial to a final configuration in an environment. zePPeLIN is based on a set of wirelessly networked overhead cameras. While each camera only covers a limited environment portion, the camera set fully covers the environment through the union of its fields of view. Path planning is performed in a fully distributed and cooperative way, based on potential diffusion over local Voronoi skeletons and local message exchanging. Additionally, the control of the moving object is fully distributed: it receives movement instructions from each camera when it enters that camera's field of view. The overall task is made particularly challenging by intrinsic errors in the overlap in cameras' fields of view. We study the performance of the system as a function of these errors, as well as its scalability for the size and density of the camera network. We also propose a few heuristics to improve performance and computational and communication efficiency. The reported results include both extensive simulation experiments and validation using a real camera network planning for a two-robot system.

Keywords Distributed Path Planning, Cooperative Multi-camera Network, Robot Navigation

1. Introduction

Path planning refers to the calculation of the path that an object has to follow in moving from a starting point to a given destination/configuration. This is a fundamental problem in 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, including other robots. A large number of versions of this problem, as well as algorithms for its solution, have been proposed in the last three decades (e.g., see [1–3] for overviews). In this paper, 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 no dynamic constraints on its motion (*holonomic motion*). The Piano mover's problem assumes that an agent who plans the path has as input a map of the environment and the object's model. However, in many cases of practical interest, such a map needs to be gathered on the spot, immediately before path planning, in order to reliably include the precise deployment of the furniture and the status of any dynamic obstacles (e.g., a moving robot, a closed door, a human crowd). For instance, sensing devices such as video or depth cameras placed in the environment can be conveniently used to build the required map.

Starting from this basic scenario, we specifically address the case in which the distance between the start and end points of the path is such that one single sensing device is not sufficient to reliably cover and map the area in between. In other words, we address the situation where path planning needs to be performed over a *large area*. Under this condition, a distributed collection of sensing devices is needed. Each device can only cover a limited portion of the environment and build a sub-map, but all sub-maps can be merged in order to obtain a full and consistent view of the entire area in which the object can move. In particular, we consider the scenario in which the sensing devices are *smart cameras* distributed over the area in which the object moves. Each camera has an *overhead view* of the situation on the ground, which is exploited to build a navigation map for the moving object. We assume that each camera is equipped with a *wireless interface for local communications* and that camera deployment is such that, altogether, the camera set can visually cover the full area on the ground. In practice, the camera set could be a set of fixed smart video cameras deployed by hand, it could correspond to a set of camera-equipped flying robots dynamically deployed over the area [4], or it could even be associated with a more general distributed aerial platform [5]. Regarding the object, we assume that it is holonomic and we do not impose restrictions on its shape.

By collectively exploiting their overhead view, the distributed camera system (hereafter referred to as *zePPeLIN (Path PLannIng Network)*) can effectively build the map of the environment and the model of the moving object and then use this information to collaboratively compute a detailed path. In zePPeLIN, the role of each camera of the network is to *plan the local part of the path* which is relative to its field of view, and to *communicate and coordinate* with its neighbouring cameras in order to let the system as a whole consistently compute the global path for the object, from its origin to the designated destination. More specifically, each camera communicates with its neighbouring cameras to locally merge their sub-paths and to cooperatively generate a global path (i.e., the precise sequence of roto-translations) for the ground moving object. During *plan 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 networked camera system allows us 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. Possible examples of *application scenarios* for the zePPeLIN system include: guiding the movements of a fleet of ground transportation robots in large warehouses, directing the motion of human crowds towards the escape exits of a large building in case of emergency, supporting and guiding a reconnaissance vehicle on the ground with the overhead surveillance provided from a fleet of UAVs flying in formation with overlapping views.

The description and implementation of the zePPeLIN distributed path planning system is the central contribution of this work. The core of the proposed system consists of a 2D deterministic approach based on the distributed combination of standard path planning techniques relying on potential fields and Voronoi skeletons. A set of heuristics is also proposed to overcome efficacy issues related to the adopted approaches (e.g., incurring in local minima while performing gradient descent) and to improve the computational and communication efficiency of the entire distributed process.

An additional - and equally important - contribution of the paper regards the study of the robustness of the distributed path planning process to *measurement errors of the relative positioning and alignment of the system cameras*. These errors can be seen as "intrinsic" to distributed vision-based or, in general, map-based approaches. In fact, since each camera can only compute the path segment for the sub-map associated with its limited field of view, the individual path segments need to be consistently merged, carefully taking into account how adjacent sub-maps overlap with each other, in order to generate a global feasible path. However, in practice it is not always possible to know the relative positioning and alignment of the different cameras with high accuracy. Hence, the presence of uncertainties about the size and the location of the regions where the fields of view of neighbouring cameras overlap have to be considered as inherent to the process. These uncertainties, in turn, can determine *inconsistencies* in the way the path segment locally computed by a camera according to its sub-map is linked with the path segments computed by its neighbouring cameras to produce the desired global path. These inconsistencies can result in an infeasible path in practice, which in some cases can be dealt with by locally calculating a *repairing path* once the problem arises during path execution (this is discussed in Section 9), or can lead to a *global path failure*. Figure 1 shows an example of such a situation and illustrates the disastrous consequences of the incorrect merging of two path segments due to errors in the knowledge of the relative positioning of neighbouring cameras (more technical explanations are given in Section 5). This could be a common situation when using flying robots or other aerial platforms, due to their intrinsic dynamic instability when hovering/floating in the air. However, also in the case of fixed cameras positioned by hand, measuring the precise fields of views and relative positioning of neighbouring cameras is not a trivial task and is expected to result in non negligible errors in practice.

When dealing with a parallel and distributed system such as zePPeLIN, the *scalability* of the system needs to be investigated. Therefore, another contribution of the paper is the study of scalability when the number of cameras and the area size increase, in the presence of the above-mentioned alignment errors. To study both scalability and robustness, we performed an extensive set of experiments both in simulation and using real cameras and moving robots. The results show that the system is in general robust, even if performance degrades for increasing levels of error, with a significant drop in performance only for large errors. Likewise, the scalability tests have shown good efficacy and efficiency for increasing the number and density of cameras.

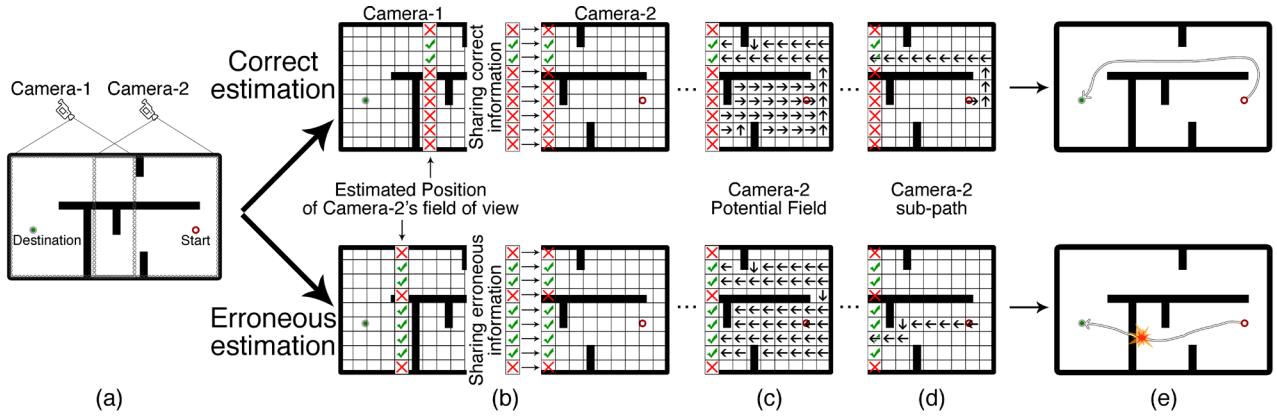


Figure 1. Illustration of the possible disastrous effect of an error in the measurement of the relative position of two neighbouring cameras. (a) The initial scenario, including the known start and destination positions, and two cameras with partially overlapping fields of view. (b) Exchange of information from Camera 1 to Camera 2 about the potential field on the edge for the correct (top) and erroneous (bottom) position estimation of Camera 2. (c) The virtual potential field computed by Camera 2 using the received information. (d) Following the virtual potential field, Camera 2 computes the local sub-path. (e) The global path resulting from the merged sub-paths: (top) successful global path computed with the correct relative position information, (bottom) the final path results in a collision due to erroneous relative position estimations.

It is important to note that, even though in this introduction, in the rest of the paper and in our previous work [6, 7] we always speak of and make use of vision-based sensors to build local maps of the ground environment, 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 used in the zePPeLIN framework, producing equivalent (if not better) performance than a camera-based system. This is to say that zePPeLIN should be seen as a general system for the distributed calculation of high resolution paths.

The rest of the article is organized as follows. In Section 2, we discuss related work both in terms of application and techniques. In Section 3, we define the problem and the reference physical model used for the overhead cameras. In Section 4, we describe the state of the art methodology that was used to implement a centralized planner based on a single global image. The centralized planner was then adapted and engineered to implement the distributed planner - the main contribution of this work - which is described in Section 5. In Section 6, 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 7 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 Section 8. The experiments are designed to study the performances of the system in relation to the main aspects of interest: robustness, efficiency and scalability. Moreover, we study how the various proposed heuristics affect the performance of the system. In Section 9, we describe the real robot experiments that we designed and implemented to validate the simulation results. Finally, in Section 10, we provide final remarks and point out the directions that will be investigated in future work.

2. Related work

Path planning is a fundamental problem in mobile robotics and, for this reason, it has been extensively studied, mainly considering single-robot approaches or multi-robot (or multi-camera approaches) based on a centralized information fusion and calculation point. An overview of the established work in this respect can be found in [1–3]. In the scenario considered in this paper, a *centralized solution* would comprise a single leading camera merging the visual information from all networked cameras. An notable example of the application of such a centralized approach can be found in the Robocup Small Size League: two overhead cameras send images of the field to a central server, which merges the images, tracks the position of the robots and the ball, and plans the robots' movements [8]. In this case, the relatively small size of the arena to be tracked allows the process to be totally centralized. In contrast, the proposed zePPeLIN approach targets potentially very large areas, and as such only relies on local and partial information. The entire process is fully distributed and the communication between cameras is only local. These architectural choices are motivated by the objective of obtaining a system which is scalable, robust to individual camera failures, and which requires minimal communication overhead. A centralized approach suffers from the *single point of failure* problem, and in order to scale to large environments it has to cope with efficiency issues and communication bandwidth bottlenecks. In fact, all cameras need to reliably forward their images to the centralized processing node, possibly using *ad hoc* multi-hop routing. The fully decentralized zePPeLIN solution does not suffer from these problems and is therefore expected to be more general, portable and efficient than centralized map-merging.

Another form of non-distributed approach which is an alternative to the distributed camera system that we propose consists of letting the ground robots build the occupancy map of the environment by themselves and perform self-localization and path planning. This is the typical approach taken in *SLAM* studies, which are very

popular nowadays (e.g., see [9, 10] for examples of visual SLAM approaches). However, while potentially more precise, this way of proceeding requires much longer computation and execution times than our approach (especially in the case of large areas), and is more prone to path errors (e.g., while performing SLAM or moving through an area, the obstacle situation in nearby areas could have changed). In any case, we do not consider zePPeLIN as a replacement for SLAM, but rather as an alternative based on different architectural assumptions.

In terms of *distributed path planning*, existing research works can be roughly grouped into four main threads: (i) on-board multi-robot path planning, (ii) path planning assisted by a sensor network, (iii) swarm approaches, and (iv) parallelism of computation. In the following, we briefly discuss some of the most prominent approaches in each of these threads of research, with a particular emphasis on the second one, since our work falls within it.

On-board multi-robot path planning studies propose distributed planning and navigation algorithms to enable a group of robots to move in an environment without colliding with obstacles or one another. An overview of the solutions to this type of scenario can be found in [11]. The more recent works [12–17] are based on different assumptions and problem statements, which also makes it relatively difficult to draw proper comparisons among the different approaches. In [13, 15], the authors investigate how the multi-robot system's behaviour changes when the noise in sensor measurements varies. In [12], the goal is to let the robots keep a predefined formation while they move towards a given destination. The work in [17] exploits multi-hop information sharing in a mobile multi-robot system for the calculation of paths that allow the simultaneous optimization of the travelled distance and network performance. In all these works, the algorithm runs on-board the robot, and thus the path is planned directly by the robot, which then implements it. To do so, the robots need to know the map of the environment. This information can be either given as input or else 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, unable to deal robustly with large and dynamic environments.

Path planning assisted by a sensor network has been considered in a number of works in the last decade [18–25]. 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 within the environment. Compared to an individual 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 this happens at locations distant from the moving robot position). In these works, the resulting path is commonly defined as a 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 receives the necessary instruction to proceed with the plan. The robot is in charge of planning and implementing the precise sequence of roto-translations in moving towards the next sensor and while manoeuvring between obstacles and other moving robots. The planned path consists of a sequence of sensors. Obstacles within the environment are usually not explicitly considered, assuming that there is always a valid path between two neighbouring 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 neighbourhood (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: the avoidance of dangerous areas [20, 22], collision-free trajectories for multi-robot systems [18], and the presence of different terrain surfaces (which results in different motion speeds) [24].

To the best of our knowledge, only Yao and Gupta [26] have studied the same type of problem in considering a distributed setting similar to our own. Their algorithm makes use of a sensor network with complex spatial sensing capabilities, and the generated path is detailed with a fine resolution and takes into account the size and the shape of the ground moving robot. This work is the closest to ours. However, the authors do not report any study of the impact of measurement errors for the overlapping areas, which is one of the focuses of our work and which is an intrinsic aspect of distributed sensory and planning systems. More specifically, in the *Distributed Probabilistic Roadmaps* algorithm proposed by Yao and Gupta, each sensor calculates a part of the path in the area within its sensing range and joins its sub-path with the one of its neighbours in a shared area of overlapping sensing. The resulting path is the sequence of the via-points that allow it to avoid collisions with obstacles and to safely reach the next sensor/router. In this setting, which is equivalent to ours, any *alignment error* between the local maps (it does not matter how they are built) can easily cause the generation of sub-optimal or even infeasible global paths, as has been pointed out in the Introduction. However, this critical aspect was not taken into account in [26], while we fully account for errors of different types and extensively study how they affect the overall performance of the system, running experiments both in simulation and with real robots. Moreover, since *scalability* and *robustness* are core properties of a distributed system, we perform an extensive analysis in this respect (see Section 8), while these issues have not been addressed in [26]. In terms of the approach taken, an important difference between the work in [26] and our own lies in the fact that in [26] the solution is based on *Probabilistic Roadmaps* (PRM) while our approach is *deterministic*, based on the gradient descent over the potential field. Normally, the use of probabilistic approaches is justified by the need to deal with the exponential growth of the search space. However, in a distributed path planning scenario, given the relatively small size of each local map, a deterministic

(and more precise) approach is computationally affordable and seems preferable, precisely because it leverages on a distributed architecture. The experimental results of Section 8 support our choice, showing that zePPeLIN is faster than the Distributed Probabilistic Roadmaps algorithm.

Swarm robotics systems are characterized by a large number of robots and the use of local communication and self-organized cooperation. Works in the literature exploit these aspects to perform swarm-level path planning (see survey [27]). In [28], the system assumes that a dedicated sensor network is deployed within the environment which navigates the swarm, in a similar fashion to what happens in the case of the works mentioned above based on the use of external sensor networks. In [29], the algorithm exploits the robots in the swarm to create a chain of static landmarks to direct the navigation of the other robots in the environment. In [30–32], the system does not allocate any specific resource for building a dedicated sensor network, but the swarm itself is a mobile sensor network. While moving in the environment, each robot acts as a landmark and communicates information about the localization of the local neighbours and at the same time receives information from its neighbours. In all these works, the sensors (or the robots acting as sensors) act as high level routers and direct the moving robots to the next sensor communicating only the direction, rather than planning the precise path through a cluttered environment. 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 involves distributing the planning computation on different processors in order to increase the overall computational speed of the system [33, 34]. In these works, as distinct from our own, the computing nodes have global knowledge of the environment, and the issue is how to optimize the division of computation. A different approach to parallelism is taken in [16], 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. Using this approach, the algorithm is meant to exploit perfect communication and to have a gracefully decline of performance in case of communication errors.

3. 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. Initially, a network of cameras is deployed within the environment where the object moves (e.g., using a swarm of flying robots, each equipped with a camera). Each camera faces the ground, meaning that we assume that each camera acquires a top view of the local environment underneath. We assume that the deployment is such that, altogether, the camera set covers - with the union of all visual fields - the ground environment, including the start and the target positions of the object. Each camera device is equipped with an on-board wireless communication system. The cameras are positioned so that they are locally within range of one another and *the fields of view of two neighbouring cameras overlap*. The size of the overlapping area must be greater than or equal to the dimensions 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. The cameras do not change their position for the entire planning process.

We assume that the cameras are equipped with a *relative (or global) positioning system* which allows them to estimate, with some uncertainty, the relative positions of neighbouring cameras in terms of angle and distance. For instance, this information could be derived through the use of GPS or the use of an on-board range and bearing system such as the one described in [35]. The relative positioning information is used to estimate the overlapping area between the fields of view of neighbouring cameras and, in turn, to locally connect the sub-paths calculated by each camera (see Section 5).

Unfortunately, an error in the estimate of the relative positions of two cameras results in an erroneous estimation of their overlapping fields of view (see Figure 3, as explained in Section 5), which can potentially have a disastrous effect on the quality and feasibility of the calculated paths. Therefore, we study how the error in mutual positioning affects the performance of the system, also in relation to the number of cameras participating in the planning process. We consider this uncertainty to be 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 what extent this parameter impacts upon the feasibility and quality of the planned paths. In the simulation experiments of Section 8, the error in 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 making assumptions that would be specific to the hardware in use.

Since the focus of this work is on distributed path planning calculation, we make simplifying assumptions for some aspects of the scenario that are not strictly related to path 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 might 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 Section 9, 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 cameras are fixed to the ceiling.

4. Path planning using a single environment map

In this section, we describe a *centralized* version of the path planner that we propose: we consider the case in which there is only one camera that has the entire environment and the moving object included within its field of view. The distributed path planning algorithm described in Section 5 is based on this centralized algorithm, but it extends it with various components aimed at effectively addressing the unique challenges of a distributed environment, in which multiple camera nodes have to tightly interact and cooperate with each other. The core contribution of this work precisely consists of the architecture and the algorithmic parts of the distributed planner, while the design of the centralized planner is mostly based on well-established techniques and approaches for *deterministic* 2D path planning [1–3].

The choice of a deterministic planner, which is reflected in the distributed version, is related to the limited size of the 2D local map, and in turn the search space, that each camera node individually has to deal with. In the case of very large search spaces, a probabilistic approach would have been preferred. The validity of our choice is confirmed by the results reported in Section 8, showing that it allows the generation of optimal sub-paths at the camera level without compromising the global performance. Moreover, the computational results show that, in terms of time, zePPeLIN outperforms the mentioned method for distributed path planning from Yao and Gupta [26], which is based on a probabilistic planning algorithm.

In the rest of this section, we describe the techniques, methods and algorithms used for implementing the centralized path planner.

4.1. Occupancy map of the environment

The ground environment where the object moves is modelled as a plane discretized in a uniform 2D grid of squared cells making-up the *occupancy map*. Each cell of the occupancy map is labelled as either a *free cell* or an *occluded cell*. The former are those 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 free cells is termed the *free space*. The occluded cells are those cells which are occupied, or partially occupied, by an obstacle. Having an overhead view of the environment, the classification of the cells as free or occupied cells is performed through a 2D isometric projection of the 3D obstacles on the ground. This might result in the inconsistent classification of some cells, depending upon how accurate the machine vision algorithm used for the occupancy map is. For instance, the projection of a "table-like" object may result in an occluded area, while there could in fact be space for passing under it (depending upon the 3D geometry of the moving object).

4.2. Potential field: calculation and diffusion

The centralized path planning method we use is the *numerical potential field technique* [1]. First, a virtual potential field is computed over the occupancy map, then the path of the moving object is calculated by 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 field is a scalar function that defines, for each cell, the attraction/repulsion intensity value. The function has a global minimum at the destination point and a maximum value on the obstacles. In all other cells, the function decreases in the direction of the destination, such that the planner can direct the object to the goal following the direction indicated by the gradient descent. A 3D illustration of the potential field is provided in Figure 2. The potential field is computed in three phases, described below: (i) the calculation of the *skeleton*, (ii) the diffusion of the potential field over the skeleton, and (iii) the diffusion over the remaining free cells.

(i) Voronoi skeleton. The *skeleton* corresponds to the *Voronoi diagram* on the 2D occupancy map [36] (i.e., the set of points where the distance to the two closest obstacles is the same [2]). 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. Once a skeleton has been calculated, the destination cell is connected to the skeleton by a shortest line path (i.e., the set of cells on the shortest line from the destination cell to the skeleton's closest cell).

(ii) Diffusion over the skeleton. Once the skeleton is computed for 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 lies on the destination cell. Next, the potential value is assigned to the neighbouring 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 neighbouring 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.

(iii) 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 process. However, in this case diffusion begins from the skeleton cells and increases towards the obstacles. The increment at the first diffusion step can be greater than one so as to increase the importance of the skeleton with respect to the other free cells. A greater difference between two adjacent cells results in a higher attraction force from the higher potential cell to the lower potential cell. Therefore, increasing the difference from the skeleton to other free cells favours paths that overlap the skeleton (i.e., paths safer with respect to collisions). In our algorithm, to favour paths overlapping the skeleton we use three as the incrementing value from the skeleton. The process iterates until the potential field is diffused over the entirety of the free space connected to the destination. The potential field 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 2 for a graphical illustration of the process).

4.3. Path calculation based on the potential field

The last phase of the planning algorithm consists of the actual calculation of the path. It is computed following the gradient in the descending direction of the potential field. The process begins from the starting position of the object and follows the decreasing value of the potential field, computed using the A^* algorithm [37]. In order to apply A^*, the 2D map is seen as a graph where the cells represent the nodes and where there is an edge between two nodes when two cells are adjacent (i.e., we use Minkowski's Taxicab Geometry [38]). The A^* algorithm finds the least-cost path from a given initial cell to the destination cell. It uses a distance-plus-cost heuristic function $f(x)$ that is computed on every cell x visited by the algorithm:

$$f(x) = g(x) + h(x) \quad (1)$$

where $g(x)$ is the *path cost* function and $h(x)$ is the *estimate* function. In our case, $g(x)$ represents the distance from the starting cell to the current cell x. $h(x)$ is a heuristic estimate of the number of steps from cell x to the destination, which is derived from the value of the potential field.

Since we consider moving objects that can have any arbitrary shape and which can occupy more than one cell, the value of the functions g and h for the generic extended object (i.e., whose object size is greater than the cell size) needs to be calculated based on the individual values associated with the spanned cells. This is realized by using the notion of control points introduced below.

4.4. Path calculation for extended objects: use of control points

At each instant, the object is in a precise *configuration*, corresponding to the spatial position and orientation of the object and extending over multiple cells. In order to calculate appropriate g and h values for the object, this is described by a set of *control points* [2]. An example is shown in Figure 2. A configuration c consists of the coordinates of the control points and the associated $g(c)$ and $h(c)$ values of Equation (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. A translation involves one cell in one of the cardinal directions (N, E, S and W, equivalent to up, right, down and left) and has a cost equal to 0.5. A rotation of θ degrees can have the pivot centred on any of the control points or on the centre of mass of the control points, with θ as 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 neighbour configurations. The $h(c)$ function is defined as the average over the potential value of all the control points. To fully describe a configuration, an additional set of points is used, namely the *collision points*. These points lie on all the cells occupied by the moving object's perimeter (see Figure 2). These points are not used to calculate the function $h(c)$ but are instead used to check for collisions with obstacles. In the case where a configuration has one (or more) collision points colliding with an obstacle, the configuration is not considered by the algorithm, being thus infeasible.

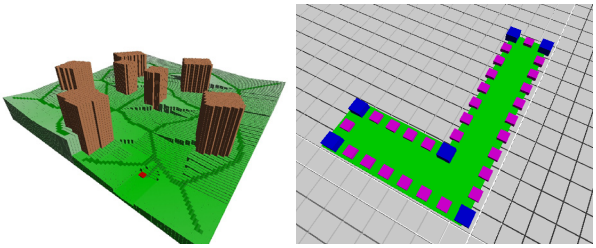


Figure 2. Graphical representation of the potential field (left) and the moving object (right). The environment is discretized in a 2D matrix of cells. The height and the saturation of the cells represent the potential field value. Obstacles are represented by high brown cells, and the skeleton by the 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).

5. The distributed path planner

The centralized planner described in the previous section 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 defines 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 neighbouring cameras, and partial knowledge of the overall status of the planning process (i.e., no omniscient leader or centralized controller exists).

The distributed algorithm presented in this section only executes the 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 Section 9.

The zePPeLIN distributed path planning algorithm is composed of three phases, described in detail below: (1) *neighbour mapping*, (2) *potential field diffusion*, and (3) *path calculation*. The zePPeLIN pseudo-code algorithm that is executed at a generic camera node is given in Algorithm 5.1.

5.1. Phase 1: Neighbour mapping at the individual nodes

During this first phase, each camera estimates the overlapping area of its field of view with that of its neighbours. A camera n calculates the overlapping area between its field of view and that of its neighbour m in the following way (and repeats the same process for all its neighbours). First, n collects two pieces of information: (i) the relative position and orientation of m, and (ii) the size of m's field of view, received from m via wireless communication.

Next, with these two pieces of information, it is able to calculate the projection of the field of view of m on its local 2D map.[1] Using the projection of the field of view of m,

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

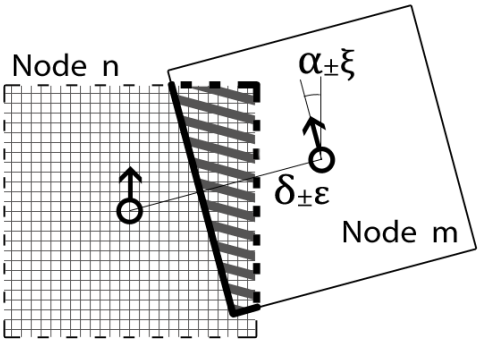


Figure 3. Illustration of how two neighbouring camera nodes m and n define their overlapping region and assign shared and open edges. δ is the relative position (in x, y coordinates) of the camera node m in the reference system of n and α is the relative orientation of m with respect to the orientation 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 Section 1, this results in an erroneous estimation of the overlapping area, which can potentially have a negative effect on the calculated path.

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 lie inside m's field of view. The shared edges are the edges of m's field of view that lie in n's field of view. A graphical representation of this process is shown in Figure 3. In the figure, only two cameras are shown for ease of comprehension; in practice, more than two cameras' fields of view can overlap.

5.2. Phase 2: Potential field diffusion across maps

The 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 since the entire environment map is segmented across the networked system, the cameras need to engage in a cooperative diffusion of the potential field. The process is fully distributed: starting from the cameras at the goal location, each camera first computes the potential over its local map; next, it sends the potential field values of its shared edges to its neighbours 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* that corresponds to the *Voronoi diagram* on its 2D occupancy map (see Section 4). 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. Any differences are due to the fact that, during the calculation of the skeleton, the frontiers of a (local) map need to be considered as obstacles. Therefore, at the corners of each map the local skeleton exhibits bifurcations that do not see any counterpart in the centralized skeleton (see Figure 4). Differences in the 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 Section 8): the paths planned with the distributed process and the paths

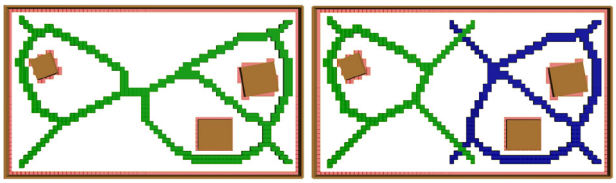


Figure 4. Comparison between the skeleton calculated in a centralized way using a single global map (left) and the sum of two local skeletons (right)

planned by the centralized planner have different lengths. An example of the paths generated by the global, single map planner and by zePPeLIN's distributed planner is shown in Figure 5.

Once the skeleton is generated, camera nodes *cooperatively diffuse the potential*, whereby 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 begin the process. They first diffuse the potential field on their local map. The diffusion starting point is the centre of mass of the control points of the goal configuration. Once the diffusion in the local map is completed, they send to their neighbours the value of the potential field on the shared edges. In this way, a camera device that receives this information can continue the diffusion, beginning from the values of the received edge cells. Upon receipt, 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 thus far (i.e., if a lower number of cells in the direction of the destination is found). This process is iteratively executed in a distributed way among all the cameras in the network. At the end of the diffusion process, the potential field in the overlapping areas of neighbouring cameras will be similar but not necessarily the same.

Cameras that have the *final destination configuration* within 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 centre of mass of the control points is not sufficient to guarantee the correct final orientation. Therefore, these cameras calculate C different potential fields, one for each of the C control points. Each potential field is then diffused using as the final destination point one of the control points. During path calculation, these cameras compute the $h(c)$ value by averaging the potential field values 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 Phase 3 (see below), the cameras which have the final destination in their view map can define the set of roto-translations of the object while also taking into consideration its desired final orientation. Since only these cameras need to use multiple potential fields, this does not have a major impact in terms of computation and communication costs, yet it is able to guarantee the correctness of the path.

It is also important to remark that, in order to *minimize the communication overhead*, cameras exchange only the value of the skeleton cells lying 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.

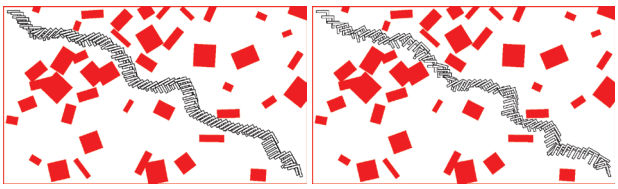


Figure 5. Final paths calculated by the global planner (left) and by the zePPeLIN system (right) in the same environment

5.3. Phase 3: Path calculation from start to destination

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 neighbours to let it continue the planning. The process is implemented in the following way.

As in the previous phase, the camera of the node that sees the moving object (i.e., 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. Next, s randomly selects one of the neighbours with which it shares the open edge. Since each camera knows the projections of its neighbours' fields of view (see the neighbour mapping of Phase 1), s can translate the coordinates of the moving object from its frame of reference to the neighbour's frame of reference. Finally, s sends these object coordinates to the neighbour, which then begins the local planning using as a starting configuration the object positioned at the received coordinates.

The process iterates among all the cameras until one of the cameras calculates the partial path that reaches the final configuration. When this occurs, the camera broadcasts to all its neighbours 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 across the network: each camera has 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 for 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 within its field of view; in the other state, it waits for a message from its neighbouring cameras. There are four possible messages that a camera m can receive from a neighbour 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 neighbour n a Start Path message, m starts the path calculation using the received coordinates as a 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: (i) n has not found a valid path in its field of view, or (ii) 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: (i) c lies over an obstacle, (ii) c has already been evaluated by n during a previous path calculation, and (iii) c can be connected to a previous partial path calculated by n. The first operation that a camera n performs after receiving a Start Path message 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 neighbour lying on the same open edge on which the configuration c is positioned. If no alternative neighbours are present, camera m resumes the local path calculation from the status at which it was before sending the configuration c.
  Throughout this process, whenever a camera fails to find a local path, the system implements 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 neighbours 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 of the path planning process. The system issues a Global Failure message when all the configurations have been explored but the camera network is unable 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 and minimal rotation angle). A Global Failure message is generated, first of all, by the start camera, according to the following process. When a camera explores all the configurations within its field of view but has found no path that reaches the goal or else 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, it is possible that this camera may now also incur

an analogous local failure while trying to find an alternative path continuation. This potential sequence of local failures can cause the system to backtrack from camera to camera along the path built thus far, until it reaches the first camera of the sequence, the one that started the planning phase and which sees the start position. At this point, no further backtracking is possible. Therefore, if the start camera also fails locally, this 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.

5.3.1. Sequential vs. parallel path calculation

With the described architecture, the path calculation phase is executed *sequentially*, with the control passing from camera to camera. A small variation of the architecture would, however, permit the utilization of *parallel execution* (in a similar fashion to the way in which parallelism is realized in [26]). Each camera can pre-calculate and store P local possible paths, each associated with a different, randomly-selected starting configuration. Later on, when a camera receives the Start Path message with the actual start configuration c, it can connect c to the closest among the start configurations of the P paths. Such local connections would be similar to the *repairing path* executed during the navigation phase (see Section 9). In this way, the cameras could exploit parallel execution, though at a cost of potentially producing less accurate solutions due to the potential sub-optimality of connecting c considering only the P available paths. Since the time to execute the sequential path calculation was, in general, quite short for the considered experimental settings (see Section 8), we did not study the effective computational gain that could be obtained with the described parallel implementation, leaving this for future work.

6. Heuristics for improving efficacy and efficiency

In the following two subsections, 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.

6.1. Heuristics to avoid local minima

During potential diffusion, the single-map path planner presented in Section 4 can potentially become trapped in *local minima*. This is due to the fact that potential diffusion does not explicitly consider the dimensions of the moving object, while during path calculation the object dimension is taken into consideration. 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 by expanding the search tree towards those areas with assigned low values for the potential field. In this way, the exploration process is naturally attracted in the direction of the narrow passage and tries to establish a path through it. However, while the skeleton and the potential field might be able to pass through the narrow passage, the moving object itself has a shape and dimensions which might prevent its crossing. If this is the case, the process becomes stuck in exploring an area that has low potential values but which is, at the same time, too narrow to allow the moving object to pass through. In other words, the search process becomes 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: (i) *skeleton pruning*, and (ii) *narrow passage detection*. The two heuristics can be implemented independently of each other and also act during different phases of the process. The illustration of the effect of the two heuristics, in comparison to the path resulting without the application of the heuristics, is shown in Figure 6. In Figure 6c, only the smallest passages are blocked by the skeleton pruning, while some wider local minima are not detected. Alternatively, in Figure 6d, the narrow passage detection is able to find the most computationally expensive local minima, but also in this case the resulting path is affected by one undetected local minimum. Instead, in Figure 6e, both the heuristics are active and the resulting path is very close to that calculated by the centralized planner.

6.1.1. 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 safety 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 or not an object can cross a passage (e.g., it depends also on the object's shape), the heuristic does not guarantee the removal of all local minima (see Figure 7). On the other hand, setting w_{sp} higher than the smallest object dimension (or, more generally, too high) might result in the pruning of the majority of (or all) feasible paths.

6.1.2. Narrow passage detection

This heuristic is executed *during path calculation*, when a camera detects a local minimum due to the presence of a narrow passage. A narrow passage is detected when the following conditions, each one indicating an anomaly related to the presence of a narrow passage, are all verified. Let x be the last visited configuration and b be the visited configuration with the lowest value of $h(.)$. The heuristic checks the conditions described below.

- $[h(x) > (h(b) + M)]$: the search algorithm is exploring configurations with a h value which is significantly higher than the lowest h value associated with the configurations visited thus far (M=10 in the experiments). As a result, the search algorithm is not exploring configurations which reduce the distance from the destination, which means that the local

Algorithm 5.1 zePPeLIN pseudo-code: local path planning and navigation algorithm executed at a generic camera node

```
1: M ← CalculateOccupancyMap()                          {* Detect obstacles and build local map *}
2: for all n ∈ Neighbours do
3:     CommunicateMyFoVSize(n)
4:     fov_n ← GetNeighbourFoVSize(n)
5:     δ_n ← DetectNeighbourPosition(n)
6:     α_n ← DetectNeighbourOrientation(n)
7:     o ← OverlappingArea(n, δ_n, α_n, fov_n)
8: end for
9: CalculateSkeleton(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:     SendPotentialFieldMessageToNeighbours(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 neighbours *}
15:     Diffuse potential(M, pot_new)                   {* Potential field diffusion with pot_new as starting points *}
16:     SendPotentialFieldMessageToNeighbours(M, N)
17: end while
18: while true do
19:     WaitForMessageFromNeighbours()
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 ← IdentifyNextNeighbourInPath(P, N)   {* s_j = last position of path P converted in the reference system of neighbour 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:     SendLocalNavigationControlMessages(PATHS[k].next)
64:     k ← k + 1
65: end while
```

planner has started backtracking. This is possibly due to the presence of a narrow passage.

- $[\forall c_x \notin skeleton]$: all the control points c_x of the current configuration x are on cells that are not skeleton cells. While the skeleton is able to pass through the narrow passage, the moving object has a shape and dimensions that might prevent its passage. Normally, the algorithm tends to calculate paths that follow the skeleton. However, if the last visited configuration x is not on the skeleton, it is sign of anomaly, indicating a possible local minimum.
- $[dist(x, b) > size(O)]$: the Euclidean distance $dist(x, b)$ between x and b is greater than the size $size(O)$ of the largest dimension of the moving object O. This check aims to detect whether or not the search algorithm has started to perform backtracking. That is, it detects if the algorithm is exploring different alternative paths that are distant from configuration b, which is so far the closest to the destination.

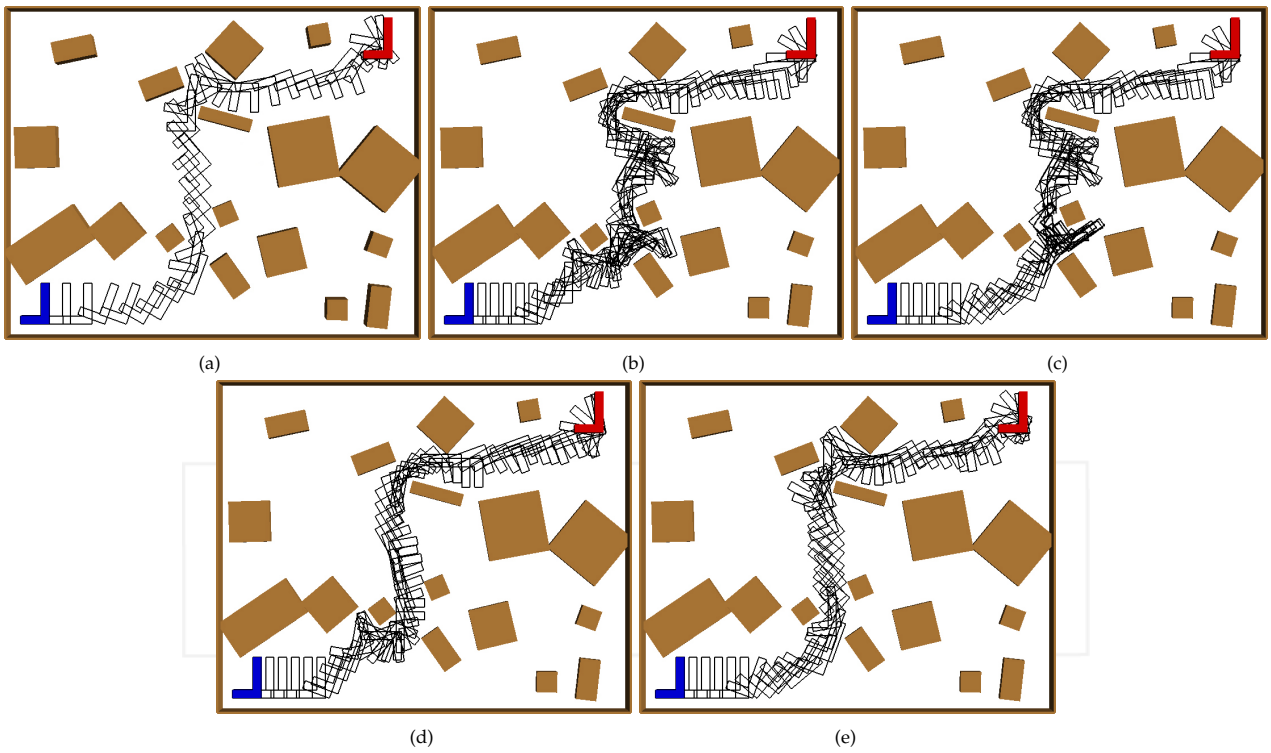


Figure 6. Comparison of paths in grid of 4×3 cameras. (a) Global planner. (b) zePPeLIN planner without heuristics. (c) Planner using skeleton pruning (with the parameter w_{sp} set as the smallest dimension of the object). (d) Planner using narrow passage detection. (e) Planner using all the heuristics.

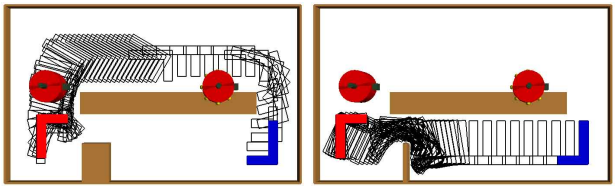


Figure 7. Width is not the only parameter defining whether or not an object can cross a passage. In the two figures, the narrow passages of the two similar scenarios have the same width. However, the object with L shape can pass through the passage of the right figure, while this cannot be said for that of the left figure.

- $[\exists\ c_{b^I} \notin freeSpace]$: there exists a control point c_{b^I} of the configuration b^I that collides with some obstacle (i.e., it is not in the *freeSpace* set). Configuration b^I is calculated by starting from b and performing a one-step translation in the direction of the descent of the potential (i.e. in the direction in which the potential decreases). This means that the algorithm has stopped visiting configurations with a lower h value because an obstacle is occluding the way. In other words, it is likely that a narrow passage lies in the direction of the shortest path.

Since each of the above criteria is a strong indicator of the possible presence of a narrow passage, when all the criteria are met together the camera node can robustly assess the presence of a narrow passage. It deals with this evidence by placing a *virtual obstacle* over the cells belonging to the passage. This way of proceeding has a potentially high cost, since the access to a part of the environment gets cut out by the virtual obstacle. This is the reason why all the above conditions are required to be met before stating the presence of a narrow passage, thus aiming to minimize the probability of the issuing of a false positive.

The centre $center_{np}$ of the passage is identified as the cell of b^I with the lowest potential value. The algorithm places a virtual obstacle on $center_{np}$ and on all the cells within the range of $size(O)/2$ cells from $center_{np}$. Next, it triggers a new distributed potential field diffusion step that avoids - in this way - the passage and, therefore, being trapped in the associated local minimum.

6.2. Heuristics to improve efficiency

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

6.2.1. Blocked cells

The goal of this heuristic is to reduce communications between neighbours and to improve the speed of path calculation. When a camera n is unable to find a local solution (i.e., when 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 Section 5). 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 different, new 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 neighbour camera n, each time sending a Start Message with a slightly different configuration attached. If n has no feasible exit configurations, this repeated process will result in the continual generation of local failures at n, consequently wasting time and generating multiple messages between the two camera nodes.

The blocked cells heuristic aims to avoid such situations. The camera m that receives a Local Failure message from a neighbour 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 lying on the blocked cells. In this way, after the first few attempts, camera m can rapidly focus on totally different new areas, possibly considering different neighbours to proceed with path construction. The adoption of the heuristic has the drawback that feasible paths might be removed from the search process following a local failure.

6.2.2. Loop avoidance

If $(n_1, n_2, \ldots, n_k)$ is the sequence of cameras associated with the computed path and $n = n_i = n_j$, then 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 or not a loop is present by controlling its local components of the path. That is, camera n checks whether the entering configurations of its two local paths can be connected together within the area within n's field of view. If this is the case, a loop is detected and 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. This process is iterated until the loop message again reaches 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. In particular, n increases the potential field value of that area up to 80% of the maximum value (i.e., the value assigned to obstacles) so that the search algorithm will not generate the same local path and will instead explore different configurations in new unexplored areas. Finally, n resumes the path planning process.

7. Adaptation to changes in the environment

An important advantage of our distributed system in comparison to single robot or centralized multi-camera systems is that it can effectively cope with dynamic environments. The system can locally and quickly detect and adapt to a change in the environment that might happen at any place and any time (e.g., changes in obstacles' positions or the appearance/disappearance of an obstacle). For instance, a planning system based on maps built by a single moving object/robot (e.g., using SLAM techniques) could not effectively cope with these situations, since the sensory range would necessarily be locally limited. On the other hand, a centralized system using multiple cameras, even in the presence of a small change, would correct the Voronoi skeleton, repeat the potential field diffusion and restart the path planning, comprising a set of operations that deal with the problem but which, taken altogether, might require considerable resources and time. In contrast, in our distributed architecture, the system can effectively reduce re-initialization costs and time by re-planning only a limited part of the path through a process of *local adaptation to changes*.

In a dynamic environment, in case a change blocks the current path, the camera n that controls the area where the change has occurred 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 can safely use the new local path without the need for informing its neighbours of the local change. 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 a start configuration the current position of the navigating object). The decision as to 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, such that when it arrives at n a new path continuation towards the final destination will already be available.

7.1. Fault tolerance

Another possible change that can occur in the environment involves 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 camera included in the path fails and stops working, neighbouring cameras can rapidly detect the problem (the cameras keep sending short keep-alive messages to each other) and trigger a repair process similar to that 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 Subsection 8.5 where, for large errors in relative camera positioning, systems with a higher density of cameras (i.e., a greater number of cameras in the same environment) exhibit a better ability to find feasible solutions compared to systems with lower densities. In these cases, when some cameras fail in practice (due to the large error) other cameras can take their place in the process.

8. Experimental results in simulation

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., incorrect estimations of neighbours' relative position and orientation), (ii) the effect of the heuristics on performance, (iii) scalability in large environments (keeping the density of cameras constant), and (iv) scalability for an increasing density of cameras (keeping the environment size constant), which corresponds to scalability of resources.

8.1. Performance metrics

The two main performance metrics we used for the system's evaluation were 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 sections, due to alignment errors the final path might be composed of disconnected sub-paths (Figure 3). The evaluation of a path consists in locally connecting its sub-paths to one another and in verifying its feasibility. This is done by 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 us to verify whether or not a solution has disconnected sub-paths that can feasibly be reconnected during the navigation phase. 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 a radius proportional to the degree of alignment error (i.e., 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) as opposed to 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 that all the sub-paths can feasibly be connected. Invalid means that the system has found a solution but that 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 in order to assess its performance. Therefore, we considered as an 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 to calculate a *safe* path which maintains a safe distance from the obstacles; rather, the objective is to have the shortest possible path to be used as a baseline reference. In the result plots showing the relative path lengths, we also indicate the length of the path calculated by the *centralized global planner* with an algorithm identical to that used in the distributed planner (i.e., using the Voronoi skeleton), but using a single global map, as described in Section 4.

8.2. General experimental setup

The experiments in simulation were run with a dedicated multi-process simulator developed for this study. The input for one simulation experiment is a set of three files including: an environment description, the camera network formation, and parametric properties. 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 two AMD Opteron 6128 (eight cores each, 2 GHz, 2x12 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 metre (i.e., cells with a size of 6.7 cm). As a unitary rotation, we used $\theta = 15^{\circ}$. The moving object has an "L" shape with two segments of the same length equal to 50 cm. During the planning process, the object is modelled by three control points, as shown in Figure 2. Aiming to be realistic, the characteristics for the field of view of the cameras, for wireless communications and for relative positioning errors, are derived from those of the flying robotic platform described in [35].

The simulation experiments are based on automatically generated maps, which differ in the placement of the obstacles. The maps are generated by placing a rectangular obstacle with a probability of 0.5 every square metre. Obstacle placement is performed by randomly selecting a 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 upon the experiment.

In Figure 9, 10 and 11, the results are presented with box plots; the bottom and top of the box are the lower and upper quartiles (i.e., the interquartile range, IQR) and the band near the middle of the box is the median. The whiskers represent the lowest datum still within 1.5 IQR of the lower quartile, and the highest datum still within 1.5 IQR of the upper quartile. Any data not included between the whiskers is plotted as an outlier with a small circle.

8.3. 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. The errors of the relative position and orientation between cameras are modelled as two independent zero-mean Gaussian distributions with configurable standard deviation. We vary the standard deviation of the

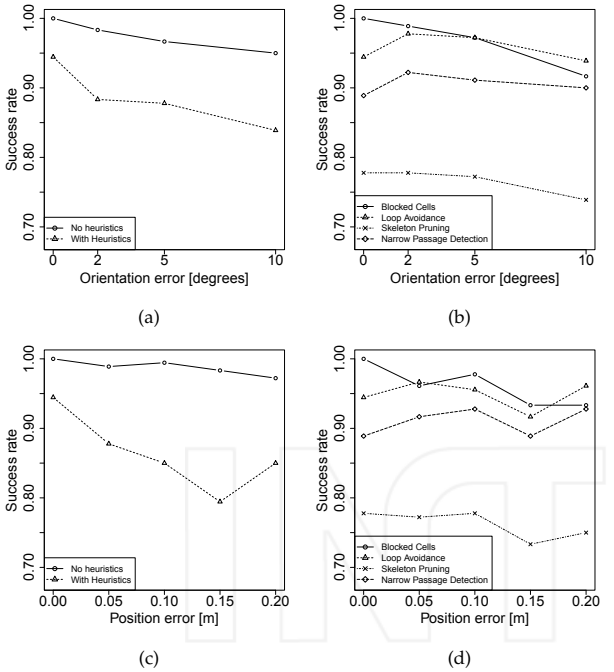


Figure 8. Success rate of the system with respect to alignment errors: comparison of different planners. (a) Planners with and without heuristics, varying the error in the relative orientation. (b) Planners with the four heuristics active independently, varying the error in the relative orientation. (c) Planners with and without heuristics, varying the error in the relative position. (d) Planners with the four heuristics active independently, varying the error in the relative position.

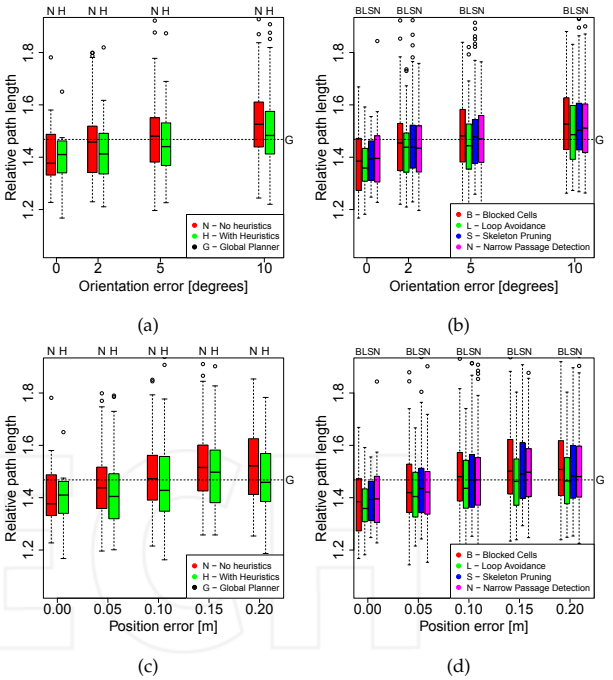


Figure 9. Relative path length for various levels of alignment error: comparison of different planners. (a) Planners with and without heuristics, varying the error in the relative orientation. (b) Planners with the four heuristics active independently, varying the error in the relative orientation. (c) Planners with and without heuristics, varying the error in the relative position. (d) Planners with the four heuristics active independently, varying the error in the relative position.

error on one measure while keeping the error for the other measure fixed to zero. In Figures 8 and 9, plots (a) and (b) show 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).

8.3.1. Experimental setup

For the experiments of Figures 8 and 9, we used 18 different maps of size 12×7 m^2 covered by a sensor network of 25 cameras, which were 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 neighbour's field of view with a width equal to 75 cm.

8.3.2. Robustness to alignment errors

Figure 8 shows that the success rate decreases with an increase in the alignment error. This is due to the fact that erroneous information about the overlapping area prevents the connection of partial paths during the planning phase.

Figure 9 shows the results for the relative path length metric. In addition, in this case the performance decreases for a high level of alignment error: the length of the final path is longer and 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, the connection paths are also longer. This increases the length of the final path.

8.3.3. Impact of the heuristics

The results for robustness in Figures 8 and 9 are reported for the different planners that 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 heuristics 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 9a) and 9c).

However, this beneficial effect has the disadvantage of a reduction in the success rate (Figure 8). 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 6, when the planner gets stuck in a local minimum, it executes a backtracking strategy which is costly in terms of resources. 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 the solution (path length). However, as a drawback they reduce the solution space and feasible paths might be removed from the search process. The effect of each heuristic is discussed in separate paragraphs below.

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 manoeuvres between the obstacles. Therefore, in some environments this heuristic closes valid paths towards the destination. In this way, it does prevents the algorithm from finding a valid solution. For this reason, the skeleton pruning heuristic noticeably reduces the success rate (Figure 8b and 8d). As an advantage, this heuristics lets the planner operate in a scenario free of local minima so that the process can rapidly converge on the solution without getting trapped. The effect is a more efficient planning process which avoids backtracking, produces shorter paths (Figure 9b and 9d), has a quicker execution (Table 1), and generates a lower number of messages (Table 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 9b and 9d), efficiency and execution time (Table 1), producing at the same time a lower reduction in the success ratio in comparison to skeleton pruning (the other heuristic for local minima). A side-effect of this heuristic is the higher number of messages it generates (Table 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 entire network.

The *blocked cells* heuristic aims to reduce the exchange of messages between neighbours. Table 2 shows that the heuristic succeeds in this purpose. However, and similar 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 6). In other words, it aims to improve the quality of the resulting paths. Its effect is visible in the plots of Figure 9b and 9d: when the loop avoidance heuristic is active, the system can calculate the paths with the shortest length.

8.4. Scalability to environment size

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

8.4.1. Experimental setup

The experiments were 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 the environment size increases, the camera network must be increased correspondingly to cover the entire area. In order to maintain a minimum overlapping area of the fields of view of 75 cm, the network scales to 5×11 cameras for environments of twice the size, and to 5×17 cameras for those of three-times the size.

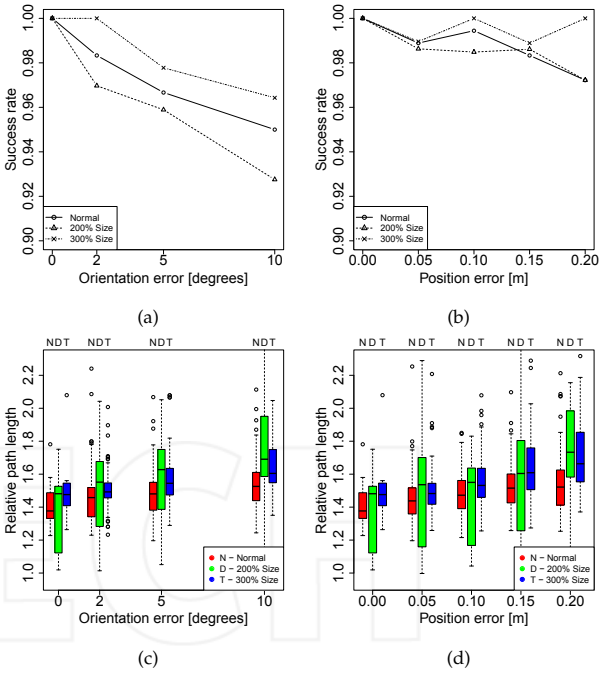


Figure 10. Scalability for environments of increasing size. (a) Success rate varying the error in the relative orientation. (b) Success rate varying the error in the relative position. (c) Relative path length varying the error in the relative orientation. (d) Relative path length varying the error in the relative position.

8.4.2. Results for efficacy and efficiency

Figure 10 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 an indicator of scalability for increasing environment size. The success rate values oscillate within the range between 100% and 93%. The plots in Figure 10a and 10b are jagged due to the strong heterogeneity of the scenarios and the random generation of errors. However, it is worth noticing that the percentage of success also remains above 93% for high levels of error.

The box plots in Figure 10c and 10d show the results in terms of relative path lengths. Moreover, 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 an 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 were randomly generated.

Figure 11 shows the *efficiency* of the system in terms of the communication overhead. We study how the number of messages changes by scaling-up the environment and the network. In order to perform a fair comparison of the efficiency performance in the different environments,

Position 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	12.30	15.96	7.89	4.08	8.92	9.23
0.05	0	6.27	7.04	4.95	1.86	4.97	4.50
0.1	0	6.61	9.85	5.84	1.94	5.90	7.25
0.15	0	15.91	12.37	11.66	4.73	9.45	15.29
0.2	0	10.45	16.63	11.60	3.08	6.63	11.42

Table 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 Err.	Orient. Err.	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	85.00	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	164.90	167.36
0.05	0	86.88	85.54	85.72	85.08	90.84	89.88
0.1	0	87.44	86.02	86.30	85.50	138.76	171.06
0.15	0	91.28	86.00	87.12	87.50	168.08	171.00
0.2	0	89.92	86.16	87.08	86.80	145.68	174.54

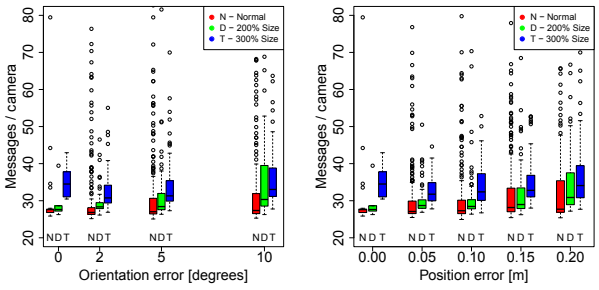
Table 2. Number of messages per camera for the planners with and without heuristics with various levels of error. The reported value is the median of the distribution. Normal: without heuristics; BC: Blocked cells; LA : Loop avoidance; SkP: Skeleton pruning; NPD: Narrow passage detection; All Heuristics: BC + LA + NPD.

the reported values are normalized with respect to the average number of neighbours (which is different for the different topologies). In fact, the cameras on the edges of the network have a lower number of neighbours than the cameras in the middle. In our scenario, cameras on the corner have two neighbours, cameras on the edge have three and cameras in the middle have four. To make the performance measure as independent as possible from this effect, the values for the number of exchanged messages are calculated as follows:

$$Messages = \frac{1}{kN} \sum_{n}^{N} p_n, \qquad (2)$$

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 neighbours in the network (e.g., in the case of a network 5×5, $k = 3.2$).

For the correct interpretation of the data, it is necessary to describe how, in our system, a camera identifies that the phase of potential field diffusion is concluded. This has been implemented with a simple *message-repetition protocol*. While a camera is waiting for new potential field messages from its neighbours, periodically (every δt seconds) it communicates to them the most up-to-date values that it holds for the potential field. Therefore, each node waiting for updated potential field values in turn continues to receive potential field messages. If the same message is repeatedly received M times, this indicates that during the last $M\delta t$ seconds no changes happened regarding the calculation of the potential field. Therefore, the camera node can safely consider the received information to be definitive and conclude the potential diffusion phase. In our experiments, to establish a good trade-off between bandwidth consumption and safety in assessing the convergence for the potential field calculation, we set M to 21 and δt to 0.01s. If bandwidth is a major issue (it was not in our robot implementation), the message-repetition protocol could be safely substituted with an alternative mechanism based on an explicit time threshold. It is worth pointing out that, in slower diffusion processes, a larger number of repeated messages is exchanged. Analogously, when the network size increases, the diffusion process obviously takes more time and the number of the (often repeated) messages increases. The number of messages communicated during the other phases is very limited compared to the selected M, and remains practically constant for all the studied scenarios within the range of 3 to 10 messages per camera.

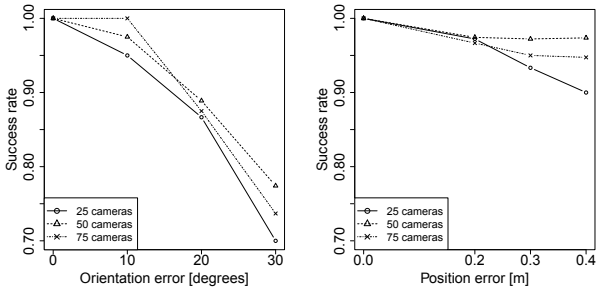


(a) (b)

Figure 11. Scalability for environments of increasing 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.

8.5. Scalability of resources

The set of experiments in this subsection 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 12. Scalability for higher numbers of cameras keeping the environment size constant. (a) Success rate varying the error in relative orientation. (b) Success rate varying the error in relative position.

8.5.1. Experimental setup

The experimental setup is the same as in Subsection 8.3. We ran experiments over 10 maps, and for every map we increased the density of the camera network. The initial network was 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 a randomly selected position and orientation. For each of the three setups, we ran experiments varying the level of error in position and orientation and executed 10 runs for each error level. This means that every point in the plots of Figure 12 corresponds to the average of $10 \cdot 10 = 100$ runs.

8.5.2. Effect of varying the number of camera nodes

For high levels of error, the performance of zePPeLIN decreases: Figure 12 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 12). 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 12). A more dense network has wider overlapping areas, which is a characteristic that allows the system to effectively plan valid paths more often (and even for high levels of error).

The system improves its performances in response to an increase in the resources within the system (i.e., the number of cameras in the sensor network). In zePPeLIN, the increase in 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.

9. 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 the planning and navigation phases: at first, it detects the moving robot's position, then it calculates the path from the current position to the given destination, and finally it 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 remainder of this section.

Figure 13. Scenario for real robot experiments. (Top) An example arena. The moving object has to move in the arena while avoiding collisions with the red obstacles. (Bottom) The moving object which follows camera instructions, and which is implemented as a set of two e-puck robots interconnected by a rigid structure.

9.1. Environment

The ground robot moves in an area of 33 m^2, where the grey floor is occluded by red obstacles. This colour configuration has been selected to ease the obstacle detection, since it is not the main focus of the work. A sample image of the arena is shown in Figure 13 (top).

9.2. Camera network

The camera network is implemented with a set of four 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 wirelessly with the other processes via UDP sockets. The cameras are placed at a height of 3 m and have a field of view of 4.56×3.06 metres, 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 their neighbouring cameras. To overcome this issue, for each camera an input configuration file specifies the IP addresses and the relative positions of its neighbours. The cameras are deployed in a rectangular formation, such that each camera has an overlapping field of view with two neighbouring cameras. All the cameras have the same orientation, relative distances of 3 m (with the neighbour on the x axis) and 2.38 m (with the neighbour 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.

9.3. Moving robot

The holonomic object moving on the ground is implemented by connecting two non-holonomic robots, the *e-puck* [2] [39], through a rigid structure, as shown in Figure 13 (Bottom). Each robot can freely rotate in place, while straight movements are constrained by the rigid structure and must be executed by coordination between the two robots. In this way, the two robots form an object with a relatively large shape which is able to rotate and move in any direction. This implementation choice allows us to generate objects with any kind of custom shape by simply adding more e-pucks to the structure. Here, we report the results for only two robots; however, we aim to employ this design for further tests with larger objects of different shapes. As discussed in Section 4, the path planning algorithm models a moving object by using control points. In this case, the *control points* correspond to the centre of the two e-pucks. Each of the two e-pucks has a coloured patch applied to its top in order to allow the cameras to track their positions and orientations through a custom tracking software.

9.4. Path planning process

Each camera is controlled by an independent process. Through 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.

9.5. Navigation in the environment

Once the path is defined, the system begins 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 to them, via Bluetooth, two independent messages containing information about the relative movement to be performed (which 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.

[2] The e-puck is a low-cost, general-purpose robot, built for educational purposes. It has a diameter of 7*cm* and limited sensory capabilities.

When process c_s navigates the robots to the final position of 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 allow it to take control and continue robot navigation. It might be the case 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 small errors in navigation or because of a relatively large overlap 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, this 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. Upon receipt of this message, c_s interrupts the navigation and hands the control to c_n.

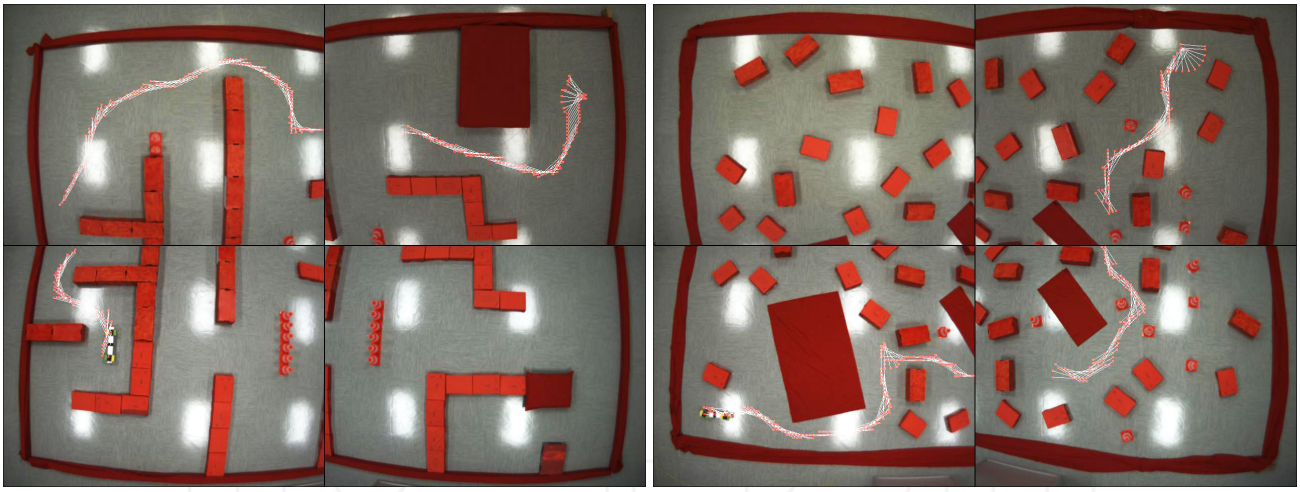
The process described for cameras s and n is then iterated between all the cameras involved in the planned path until the moving robots reach the desired destination. A few sample videos showing path calculation and path navigation are available at the supplementary page [40].

9.6. 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 14a) for an example), while in the remaining six the obstacles are deployed completely randomly (see Figure 14b 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 14a and 14b show the composite screenshot of the four fields of view. The planning time is relatively low - for example, the planning time for the paths shown in Figures 14a and 14b is, respectively, 4.6 s and 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 robots' position and the recalculation of the next relative movements to be communicated.

Through simulations, we have exhaustively tested the systems for different scenarios and error levels. By the real system implementation, we completed the process by adding, after path planning, the path actuation by distributed navigation. This set of experiments successfully verified the applicability of the proposed approach. It is important to note that the path planning process did not present noticeable differences in moving from simulation to the real world, both as regards implementation issues and the observed results. For the scenario considered, the main challenges for a real system implementation concern sensing and actuation errors. When performing path planning, sensing errors might



(a) Structured environment (b) Randomly displaced obstacles

Figure 14. Composite screenshot 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 about the final path is segmented into three partial paths distributed among the cameras. The bright spots are the reflection of the ceiling light.

affect the computation of the navigable areas in the local maps and the relative alignments between cameras. The former error has been removed in practice by exclusively using red boxes over a grey floor as obstacles: this setup allowed each camera to compute an error-free occlusion map. Instead, alignment errors have been estimated in a more or less accurate way (resulting in the values reported in Section 9.2), and have affected the performance accordingly.

10. Conclusions and future work

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

In simulation, we performed an extensive analysis of the system's performance and the heuristics. We validated the system through a series of experiments using real camera devices and moving robots. Compared to systems with a single camera or using centralized computation, 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 neighbouring cameras - a type of error which we consider to be inherent to any distributed system of local, partially overlapping maps.

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

In principle, the zePPeLIN distributed algorithm can be used with any networked system of devices, so long as the devices are capable of building a local map of the environment in their 'field of view' (e.g., a Kinect). In future work, we intend to implement zePPeLIN with a network of different devices instead of cameras. Moreover, we intend to extend the system by adding general mechanisms to manage - and possibly reduce - the overlapping errors. Additional tests will consider larger environments and the inclusion of dynamic obstacles (e.g., humans or moving robots).

11. Acknowledgements

This work was partially supported by the European Union through the ERC Advanced Grant "E-SWARM: Engineering Swarm Intelligence Systems" (contract 246939), and was partially supported by the Swiss National Science Foundation (SNSF) through the National Centre of Competence in Research (NCCR) Robotics. Marco Dorigo acknowledges support from the F.R.S.-FNRS of Belgium's French Community, of which he is a Research Director.

12. References

[1] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic, Norwell, MA, USA, 1991.

[2] 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.

[3] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[4] M. Onosato, S. Tadokoro, H. Nakanishi, K. Nonami, K. Kawabata, Y. Hada, H. Asama, F. Takemura, K. Maeda, K. Miura, and A. Yamashita. Disaster information gathering aerial robot systems. In *Rescue Robotics*, pages 33–55. Springer, London, 2009.

[5] A. Mohammed, A. Mehmood, F.-N. Pavlidou, and M. Mohorcic. The role of High-Altitude Platforms (HAPs) in the global wireless connectivity. *Proceedings of the IEEE*, 99(11):1939–1953, 2011.

[6] A. Reina, G. A. Di Caro, F. Ducatelle, and L. M. Gambardella. A distributed approach to holonomic path planning. In *Workshop on Motion Planning: From Theory to Practice,* at *Robotics: Science and Systems (RSS)*, 2010.

[7] A. Reina, G. A. Di Caro, F. Ducatelle, and L. M. Gambardella. Distributed motion planning for ground objects using a network of robotic ceiling cameras. In *Proceedings of 12th Conference Towards Autonomous Robotic Systems (TAROS)*, pages 137–148. Springer, Berlin, 2011.

[8] S. Zickler, T. Laue, O. Birbach, M. Wongphati, and M. Veloso. SSL-Vision: The shared vision system for the RoboCup Small Size League. In *RoboCup 2009: Robot Soccer World Cup XIII*, volume 5949 of *LNCS*, pages 425–436. Springer, Berlin, 2010.

[9] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich. The vSLAM algorithm for robust localization and mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 24–29. IEEE Computer Society Press, Los Alamitos, CA, 2005.

[10] A. Gil, O. Mozos, M. Ballesta, and O. Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual slam. *Machine Vision and Applications*, 21(6):905–920, 2010.

[11] L. E. Parker. Path planning and motion coordination in multiple mobile robot teams. In *Encyclopedia of Complexity and System Science*, pages 5783–5800. Springer, 2009.

[12] 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 2nd International Workshop on Multi-Robot Systems*, volume 2 of *Multi-Robot Systems: From Swarms to Intelligent Automata*, pages 267–278. Springer, Berlin, 2003.

[13] 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.

[14] 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)*, pages 953–959. IEEE Computer Society Press, Los Alamitos, CA, 2010.

[15] 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.

[16] M. Otte and N. Correll. Any-Com multi-robot path-planning: Maximizing collaboration for variable bandwidth. In *Distributed Autonomous Robotic Systems*, volume 83 of *Springer Tracts in Advanced Robotics*, pages 161–173. Springer, Berlin, 2013.

[17] G. A. Di Caro, E. Feo, and L. M. Gambardella. Use of time-dependent spatial maps of communication quality for network-aware multi-robot path planning. In *Proceedings of the 8th International Workshop on Wireless Sensor, Actuator and Robot Networks (WiSARN)*, Benidorm, Spain, June 22–27, 2014.

[18] R. Luna and K. E. Bekris. Network-guided multi-robot path planning in discrete representations. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'10)*, pages 4596–4602. IEEE Computer Society Press, Los Alamitos, CA, 2010.

[19] 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, 2004.

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

[21] 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.

[22] C. Buragohain, D. Agrawal, and S. Suri. Distributed navigation algorithms for sensor networks. In *Proceedings of IEEE INFOCOM*, pages 1–10. IEEE Computer Society Press, Los Alamitos, CA, 2006.

[23] 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)*, pages 2072–2077. IEEE Computer Society Press, Los Alamitos, CA, 2006.

[24] 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)*, pages 777–784. IFAAMAS, 2007.

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

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

[27] A. Reina and V. Trianni. Deployment and redeployment of wireless sensor networks: a swarm robotics perspective. In N. Mitton and D. Simplot-Ryl, editors, *Wireless Sensor and Robot Networks*, pages 143–162. World Scientific, Singapore, 2014.

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

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

[30] 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.

[31] 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.

[32] F. Ducatelle, G. A. Di Caro, C. Pinciroli, F. Mondada, and L. M. 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.

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

[34] 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.

[35] 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)*, pages 3659–3664. IEEE Press, Piscataway, NJ, 2009.

[36] 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.

[37] 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.

[38] A. C. Thompson. *Minkowski Geometry*. Cambridge University Press, New York, 1996.

[39] 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.

[40] Supplementary page. http://iridia.ulb.ac.be/supp/IridiaSupp2012-013/index.html, Accessed on 14 March 2014.