Swarm Robotics and Rapidly Exploring Random Graph Algorithms Applied to Environment Exploration and Path Planning

Cindy Calderón-Arce[1], Rebeca Solis-Ortega[2]
School of Mathematics,
Costa Rica Institute of Technology,
Cartago, Costa Rica
https://orcid.org/0000-0002-0077-225X[1]
https://orcid.org/0000-0002-3065-8386[2]

Abstract—**We propose an efficient scheme based on a swarm robotics approach for exploring unknown environments. The initial goal is to trace a map which is later used to find optimal paths. The algorithm minimizes distance and danger. The proposed scheme consists in three phases: exploration, mapping and path optimization. A cellular automata approach is used for the simulation of the fist two phases. For the exploration phase, a stigmergy approach is applied in order to allow for swarm communication in a implicit way. For the path planning phase a hybrid method is proposed. First an adapted Rapidly-exploring Random Graph algorithm is used and then a scalarized multiobjective technique is applied to find the shortest path.**

Keywords*—*Swarm robotics; cellular automata; path planning; Rapidly-exploring Random Graph (RRG); scalarized multiobjective optimization

I. Introduction

Swarm algorithms have been deeply studied to address problems such as food search and object collection. However, it has been recently used to the exploration and mapping of scenarios, given the importance that this entails in rescue situations. In particular, use swarms robotics for the examination of scenarios in the search for optimal, efficient and safe routes, in a prudential time, reduces the loss and waste of resources in trajectories inspection.

Swarm robotics study the coordination of a large group of relatively simple robots through local rules and implicit communication. This field emerges from the application of swarm intelligence to robots [1]. Swarm intelligence is inspired on insect colonies, bird flocks, fish schools and other types of animal clusters which accomplish complex tasks through simple rules and communication.

It has to be clear that a group of agents must meet specific requirements in order to be considered as a swarm. The agents must be: autonomous, homogeneous, able to sense and actuate in the environment [2]. The aforementioned set of characteristics ensure a distributed and scalable swarm.

There are several swarm applications that have been studied: aggregation, flocking, exploration, foraging, navigation, path formation, object assembly and others [1], [3], [4]. In many of this applications, the use of a swarm is desired given the dangerous nature of the task.

Even though the use of a swarm brings several advantages, the type of communication associated with it can create a possible drawback. There are two possible control schemes: centralized and decentralized. Neither of these control schemes facilitate the supervision of the swarm by a human operator [1].

When a centralized swarm is used its scalability is poor and the swarm becomes sensible to the loss of its central leader. The decentralized approach overcome this main issue, but does not allow to synthesize or access global data unless all individuals are connected to each other. Therefore a human controller cannot access the data thus it cannot predict or alter the behaviour of the swarm.

In addition, path planning methods that minimize not only distances, but also danger, cost, time or energy are of great relevance searching evacuation and access routes in buildings, urban centers or even forests.

In this paper we focus our work on the exploration of unknown, static and dangerous environments. Although these tasks can be executed by humans, the use of swarm robotics will allow to save resources and protect the people in charge of those tasks.

To solve the problem of creating an optimized pathway in an unknown environment, three phases are proposed. In the first stage, a simulated swarm based on a cellular automata scheme will be used for exploring unknown environments. This scheme autonomously propitiate an efficient dispersion of the swarm through the unknown area. In order to perform the task in a more efficient and scalable manner, a decentralized swarm is implemented.

The second part of the solution also uses a cellular automata approach, in which all the visited cells are recorded by an external server in order to trace the map. The non-visited cells are marked as obstacles. Finally, in the third stage, a discrete adapted RRG structures the zone by means of a graph. Then, a Dijkstra algorithm finds the shortest path between two given points.

This paper is structured as follows: in the next section works related with swarm robotics, exploration and path planning are shown. Methods and Materials section presents problem statement and the algorithms used to solve each problem stage. Section IV shows environments and experimental set up in the simulations were carried out and their final results. Finally, in the last one general conclusions of the work will be discussed.

II. Related Work

One aim of environments exploration is to know a throughout region, with the purpose of detect some targets distributed randomly in the area [5]. For accomplish this task, the most applied algorithms are bio-inspired. Among the approaches that have been developed, the stigmergy used by many colonies of insects to coordinate their activities, which allows a swarm indirect communication, has been widely researched [5], [6].

Palmieri et al. implemented and tested three biologically inspired coordination strategies: firefly, particle swarm and artificial bee behaviour [5], the better performance was obtained with the firefly-based strategy. This scheme focus on finding targets and recruit robots around them.

In addition, Tan et al. employed a stigmergy method for target search in unknown environments, by means of a swarm [6]. The stigmergy mechanism was employed to guide the robots motion. A pheromone map was used for helping them to reach the target. The most of exploration studies focus only on exploration for finding a target [7], [8]. Therefore, they do not guarantee the recognition of whole explored area.

On the other hand, the standard methods for solving path planning problems are based on using approximated schemes known as sampling based planning methods. These methods employ a random sample of the space (connecting points randomly) and deciding if a route or direction is feasible or if exists a possible collision [9]. Probabilistic RoadMaps (PRM) [10], Rapidly-exploring Random Graph (RRG) [11] and Rapidly-exploring Random Tree (RRT) [12] algorithms are example applications of these methods. RRG generates an undirected graph, possibly containing cycles, and RRT a directed tree. Similarly, PRM algorithms trace a roadmap (graph) which represents a set of collision-free trajectories for computing the shortest path that connects an initial node to a final one [13].

All of these algorithms differ on the process applied to construct a connecting graph [14]. They are probabilistically complete algorithms which have natural support for solving high dimensional complex problems [15]. However, they have the disadvantage of no capacity to stop execution upon failure nor the ability to report when no possible solution exists. Therefore, they are computationally expensive [16]. Another known graph search algorithm is called A*, which uses a discrete space and its success is highly dependent on grid resolution [17].

Dijkstra is one of most famous and simple optimization algorithm [18], with a quadratic time complexity [19]. Also bio-inspired optimization algorithms, as a metaheuristic methods, are often used to approximate an optimization problem solution. They obtain solutions on an efficient way but are not able to meet the real time constraints, neither to reproduce the same solution since they are stochastic [16].

Furthermore, for finding a path, it is possible to optimize not only distance but also other objectives like danger, time or energy needed to cross it. In that sense, it is possible to optimize a multiobjective problem, taking into account several objectives, instead of a problem with a unique objective [20]. Without loss of generality, we consider a biobjective problem, which optimizes distance and dangerousness, but it is also possible to extend the proposed solution to a problem with more than two objectives. A technique implemented to solve multiobjective problems combines the objectives by means of a linear combination of them. Thus, the multiobjetive problem becomes uniobjective and the general optimization methods could be applied to solve the scalarized problem. In which, the objective function incorporates performance indicators of different objectives [21].

III. Materials and Methods

A simulated swarm of robots and multiobjectives techniques are used as part of the proposed solution, which is organized in three phases: exploration, mapping and path planning, all carried out on simulated environments.

A. Exploration Phase

A cellular automata approach was used for representing the environment, obstacles and swarm. Von Neumann neighborhood, which is composed by a central cell and its four orthogonally adjacent cells, was used. The states associated to each cell were defined by integer numbers as follows: (0) free cell, (1) cell occupied by an agent and (2) obstacle. The first two states are changeable during the time, but the third one is static.

We implemented two approaches to control the behavior of the swarm: a classic scheme based on a random walk algorithm, and a bio-inspired one based on stigmergy concept. For both, a modification was made which constraints the agent direction.

The stopping criteria of the algorithms was based on the percentage of environment coverage. It was selected just to valid and compare the schemes but for real world applications others criteria can be used like number of iterations or elapsed time.

1) Random walk algorithm (RW): This algorithm is used for search strategies for both animal and robots. It is especially useful when the individuals do not know the environmental and do not have cues that can drive the motion, or when their cognitive abilities do not support complex localisation and mapping behaviours [22].

In its simplest form, a random walk can be thought of as a sequence of straight movements and direction changes [22]. Given this an agent can be in one of two states: moving randomly or changing direction for avoiding obstacles [23].

If we take this scheme and adapt it into a cellular automata environment, RW can be modeled assuming that each agent is located in a cell and randomly chooses another one free in its neighborhood.

A modification of RW, called random walk with direction (RWD), adds a priority direction to each agent. This priority will cause that an agent will never be able of choosing a free cell that is opposite to its priority direction. In that sense, an

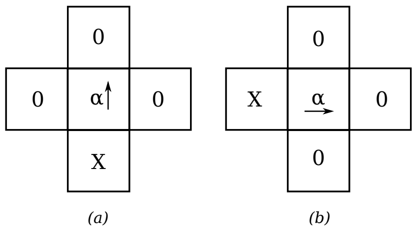


Fig. 1. Representation of the three directional neighbors of the cell α where the agent is located (X represents the non directional neighbor). (a) An agent with direction North can occupy any free cell situated West-North-East from its current position. (b) An agent with direction East can occupy any free cell situated North-East-South from its current position

agent located in α cell, will only have three possible cells to move on (directional neighbors). In Fig. 1 on the example (a) the priority direction is set north, in that case the agent could choose between three cell options: east, north or west. If obstacle collision occurs, the priority will randomly change before to continuing with the algorithm, as shown in Alg. 1.

Algorithm 1: Exploration algorithm by random walk with direction (RWD)

```
   Input: $\rho$
1  $\alpha \leftarrow \rho$ position cell
2  $\tau \leftarrow \rho$ direction
3  while the stop criteria are not satisfied do
4      foreach $\rho$ do
5          $w^t(\rho_i) \leftarrow$ state of the $i$-directional neighbor at time $t$
6          if $\exists\ w^t(\rho_i) = 0$ then
7              $j$: randomly choose one directional neighbor
8              $\alpha \leftarrow \rho_j$
9          else
10             obstacle detected
11             $\tau \leftarrow$ choose new direction
12         end
13     end
14     $t++$
15 end
```

Even though random walk performs very well on a swarm [23], it does not allow agents to learn about the swarm decisions or to communicate with the environment (beyond the detection of obstacles). In order to take thoughtful decisions for accomplish a faster dispersion through the zone, an alternative algorithm based on a stigmergy approach is proposed.

2) Virtual pheromone algorithms: The stigmergy approach, proposed by Grassé and mentioned by Tan et al. in [6] is a mechanism of indirect communication. The agents leave a trace in the environment to propitiate different actions in others agents for leading to a spontaneous and systematic behavior emergence.

Based on this concept, we developed an algorithm called pheromone walk (PW). In this the agents (ρ) move through the environment sensing and leaving virtual pheromone on visited cells, with the aims of communicate the space already explored.

The pheromone has two parameters: intensity and evaporation rate. The intensity (Υ) determines how strong the initial signal is in each cell α and the evaporation rate indicates how fast this signal will fade away. This pheromone is modeled by an iterative relation, that depends on time t and a decay rate κ, as follows:

$$\Upsilon^{t+1}(\alpha) = (1-\kappa)\cdot\Upsilon^t(\alpha) \tag{1}$$

where Υ^0 is the initial pheromone intensity and κ is a constant selected properly.

Under this approach, the agents choose a free cell with the lowest pheromone intensity, in the case that two or more cells have the same lowest intensity, then one will be chosen randomly.

Similar to the random walks algorithms, a modification of PW called directional pheromone walk (PWD) was implemented. In this case, also a priority direction is assigned to agents, which will select a free cell with the lowest pheromone intensity and located in one of its three directional neighbors. If an obstacle collision is detected, the agent will have to change its direction and move on (see Alg. 2).

Algorithm 2: Exploration algorithm by directional pheromone walk (PWD)

```
   Input: $\rho$
1  $\alpha \leftarrow \rho$ position cell
2  $\tau \leftarrow \rho$ direction
3  while the stop criteria are not satisfied do
4      foreach $\rho$ do
5          $w^t(\rho_i) \leftarrow$ state of the $i$-directional neighbor of $\rho$ at time $t$
6          if $\exists\ w^t(\rho_i) = 0$ then
7              $j$: randomly choose one directional neighbor with $min(\Upsilon^t(\rho_i))$
8              $\alpha \leftarrow \rho_j$
9          else
10             obstacle detected
11             $\tau \leftarrow$ choose new direction
12         end
13     end
14     foreach $\alpha$ do
15         $\Upsilon^{t+1}(\alpha) \leftarrow (1-\kappa)\cdot\Upsilon^t(\alpha)$
16     end
17     $t++$
18 end
```

B. Path Planning Phase

Once the obstacle positions are known and the map is already constructed, an adapted RRG algorithm is used to structure the space. After that, a Dijkstra algorithm is applied to find a shortest path between an initial and goal given points.

1) Adapted RRG algorithm: RRG original algorithm operates in a space, in which a configuration is represented by any point on the work space, including obstacles [9], [15]. In this work, each agent is a point into a two dimensional space.

Since the amount of possible configurations is uncountable, the work space is discretized through a rectangular uniform partition $\mathcal{M}$ of $segh \times segv$ dimension. Thus, adapted RRG algorithm generates a random point on each cell or partition's element.

The adapted RRG algorithm structures the search in a partition of the configuration space, using a discretization and forcing the graph to explore the whole space by including a vertex from each cell of the partition, if it is possible. Let $\mathcal{C}$ be the space configuration, $\mathcal{C}_{free} \subset \mathcal{C}$ the set of collision-free configurations and $\mathcal{G}(V, E)$ the graph defined by the vertices set V and the edges set E. The adapted RRG algorithm initializes the graph with q_{init} as a unique vertex without edges. A new configuration is generated creating a random point in $\mathcal{C}$ and looking for the nearest vertex $q_{near} \in \mathcal{G}$, by means of a Breadth-First Search (BFS) and a First In First Out (FIFO) buffer [9], considering possible collisions. The new point and its edge with q_{near} are added to the graph and so on, until each cell of the partition has a vertex into the graph. When a collision between the graph and any obstacle is detected, the algorithm looks for the nearest collision-free configuration in the same direction as q_{new} [24].

A graph $\mathcal{G}$ has a collision if $\mathcal{G}$ goes through a configuration $q \in \mathcal{C} \backslash \mathcal{C}_{free}$. Let $\mathcal{W}$ be the set of obstacle borders in the work space. Now suppose that $\mathcal{G}$ does not have a collision on the ith iteration, but it is obtained in the next one by adding the vertex q_2 and the edge $\overline{q_1 q_2}$. Then $q_1 \in \mathcal{C}_{free}$ but there exists at least a point in $\overline{q_1 q_2}$ that is not in $\mathcal{C}_{free}$, i.e, is in $\mathcal{W}$.

Consider the function f defined as follows:

$$f(x, y) = (x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1) \tag{2}$$

where (x_1, y_1) and (x_2, y_2) are points on the Cartesian plane. The equation $f(x, y) = 0$ defines the locus of the points from the line l passing through (x_1, y_1) and (x_2, y_2), which divides the plane into two regions. Since f is continuous on $\mathbb{R}^2$, if a point (x, y) over one side of l produces $f(x, y) > 0$, then $f(x, y) > 0$ for all points in that region and $f(x, y) < 0$ for all points in the opposite region. By means of that property the intersection between two segments $\overline{AB}$ and $\overline{CD}$ occurs if and only if $f(A) \cdot f(B) < 0$ with respect to $\overline{CD}$ and $f(C) \cdot f(D) < 0$ with respect to $\overline{AB}$.

Identifying when the graph intersects $\mathcal{W}$, it is possible to define when accept or discard q_2. In that sense, a COLLISION() function determines if there is an intersection between $\overline{q_1 q_2}$ and $\mathcal{W}$ and helps to determine if a new point q_2 is inside or outside an obstacle, based on the previous function (2) and the Jordan curve theorem [24], [25].

Alg. 3 shows how a network is generated where, $D(m)$ is the set of all possible directions to go from the reference point on cell m to an adjacent cell, and $F(m, d)$ indicates the selected cell after applying $d \in D(m)$ from m. CELL function provides the index of the cell corresponding to a configuration q. All points generated on each cell are contained on $PointList$. On each iteration a center cell is picked from a list called $Queue$ and a point is generated on every adjacent cell following the BFS method. $Queue$ originally contains the index corresponding to q_{init} and adds indices of those adjacent cells from each iteration. Central indices that were already searched are deleted from $Queue$ and added to $Memory$. The GRID_POINT() function gives a point into the cell of index $F(m, d)$, if it is possible. If not, the answer will be $\emptyset$. The indices are arranged into a sequential order beginning on CELL(q_{init}) and continues bordering the adjacent cells. The algorithm ends when all the cells have been "visited" by the graph.

Algorithm 3: Space structuration

```
   Input: $\mathcal{W}$, $q_{init}$, $q_{goal}$, $segh$, $segv$
   Output: $\mathcal{G}(V, E)$
1  $V \leftarrow q_{init}$
2  $E \leftarrow \emptyset$
3  $\mathcal{M} \leftarrow$ Uniform Partition $segh \times segv$
4  $PointList \leftarrow [\ ]$
5  $i \leftarrow$ CELL($q_{init}$)
6  $g \leftarrow$ CELL($q_{goal}$)
7  $PointList(i) \leftarrow q_{init}$
8  $PointList(g) \leftarrow q_{goal}$
9  $Queue \leftarrow \{i\}$
10 $Memory \leftarrow \emptyset$
11 while $Queue \neq \emptyset$ do
12     $m \leftarrow Queue(1)$
13     $Queue \leftarrow Queue \setminus \{m\}$
14     if $m \notin Memory$ then
15         $q \leftarrow PointList(m)$
16         for $d$ in $D(m)$ do
17             $q_{new} \leftarrow$ GRID_POINT($\mathcal{W}, q_{init}, ...$
                 $...PointList, F(m, d)$)
18             $PointList(F(m, d)) \leftarrow q_{new}$
19             if $q_{new} \neq \emptyset$ and $q \neq \emptyset$ then
20                 if not COLLISION($\mathcal{W}, q, q_{new}$) then
21                     $V \leftarrow V \cup \{q_{new}\}$
22                     $E \leftarrow E \cup Edge(q, q_{new})$
23                 end
24             end
25             $Queue \leftarrow Queue \cup \{F(m, d)\}$
26         end
27         $Memory \leftarrow Memory \cup \{m\}$
28     end
29 end
```

Space structuration algorithm (Alg. 3) shows an unidirectional search method, which generates just one graph from the initial configuration to structure the complete work space. But also a sequential and parallel bidirectional search was implemented, on which two sub-graphs are generated; one from q_{init} and the other one from q_{goal}, and at the end both sub-graphs are joined to structure all the work space. The sequential algorithm alternates each sub-graph generation looking for a balance in the amount of vertices on each one and parallel one generates each sub-graph simultaneously.

2) Optimization scalarized problem: Once the graph has been generated, a Dijkstra Shortest Path (DSP) algorithm is used to find the optimal path from q_{init} to q_{goal} [18]. Taking into account not only distance but also dangerousness, by means of the following multiobjective optimization scalarized problem

$$\min_{p \in \mathcal{P}} \left(\omega D_1(p) + (1 - \omega) D_2(p)\right) \tag{3}$$

where $D_1(p)$ and $D_2(p)$ are the distance and dangerousness values of the path p, respectively, $\mathcal{P}$ is the set of all possible paths between q_{init} and q_{goal} throught the graph and $\omega \in [0,1]$ is the scalarizing constant.

A filter is applied before using the DSP algorithm, which simplifies the graph removing all the vertices and its corresponding edges with dead ends. Then a cost matrix $CostMat$ is calculated which assigns each element with the objective function value $(\omega D_1(e) + (1-\omega) D_2(e))$, for all edges $e \in E$. If two vertices of V are not directly connected its cost value will be infinity. $D_1(e)$ assigns length and $D_2(e)$ dangerousness of e. Since, it is not possible to know a priori a continuous danger function defined on all the environment but it is possible to know information about dangerousness in some places of environment, we simulate danger information as a discretized heat distribution on the explored environment. Then, that information is used to construct a continuous dangerousness function into the work space. A Radial Basis Function (RBF) is selected to construct that function, because the most likely the information has lots of different features and nonlinear behaviour and it is known that RBF has a good performance ajusting nonlinear data. Although there are different kernels to used with RBF, for simplicity without loss of generality we consider a polyharmonic spline (PHS) kernel. Let $\psi : \mathcal{C} \to \mathbb{R}$ be the resulting heat RBF, which is defined as follows

$$\psi(x,y) = \sum_i \left(\eta_i \cdot \phi_i(r)\right) \tag{4}$$

with $r = \|(x,y) - (x_c, y_c)\|_2$, (x_c, y_c) represents where the RBF is centered, ϕ_i each PHS function, η_i its corresponding weights and $\phi(r)$ is defined as

$$\phi(r) = \begin{cases} r^k \ln(r) & \text{; for an even } k \\ r^k & \text{; otherwise} \end{cases} \tag{5}$$

where k represents the PHS order [26], [27]. Thus, ψ is used to assign dangerousness rate on each edge e of graph $\mathcal{G}$, through a line integral over ψ as shown in (6), which is approximated by means of numerical methods.

$$D_2(e) = \left| \int_\psi e \, ds \right| \tag{6}$$

Finally, the DSP algorithm is executed to obtain the optimal path, using *CostMat* defined above.

IV. Experiments

The work space was portrayed as a two-dimensional map composed by different kind of obstacles, randomly distributed.

A. Experimental Setup

Three environments have been created in order to analyze the execution of the system. We take under consideration different scenarios like closed and open spaces, convex and concave obstacles, dead ends and others.

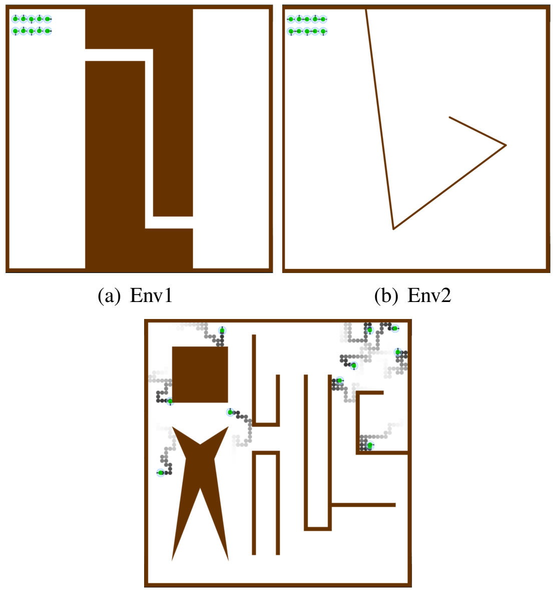

(a) Env1 (b) Env2 (c) Env3

Fig. 2. Representation of environments used for analysing the execution of the proposed solution.

The simulations were carried out using the software Processing 3.3.7, with a 1005×1005 pixel map for all the environments. Brown shapes represent the obstacles, while the agents of the swarm (which are a 1:67 scale of the map) are shown as two concentric circles that portray the body and the range of sensing. Also, the pheromones was represented by a circle in the corresponding cell (Fig. 2).

For the exploration phase we employ three different sized swarm of 10, 15 and 20 agents. The swarm, in each case, was deployed from the same area. For the PWD and PW algorithms the parameters selected for the evaporation rate was $\kappa = 1/\Upsilon^t(\alpha)$. Also five different initial pheromone intensities (Υ^0 = 100, 300, 500, 700, 900) where defined.

All the agents are autonomous, homogeneous and have the ability to communicate with the environment through pheromones. Each robot not only walks around the environment avoiding obstacles, but also leaves a trace behind it of pheromone and senses the virtual substance of others and determines the intensity of it.

For the optimization phase, work space partitions of 5×5, 10×10, 15×15 and 20×20 were used in order to generate the possibles optimal paths. Three order PHS functions ϕ_i are using to compute ψ, by means of a RBF with the training data as centres and the heat distribution shown in Fig. 3.

The heat distribution is the same for all environments and *CostMat* is calculated with five different values for the scalarizing constant ω: 0, 0.125, 0.25, 0.375 and 0.5.

Problem shown in (3) is solved 100 times on each environment for each ω values defined above and for the three work space partitions, generating the graph with three different searches: unidirectional, sequential bidirectional and parallel bidirectional, denoted Graph 1, Graph 2 and Graph 3, respectively.

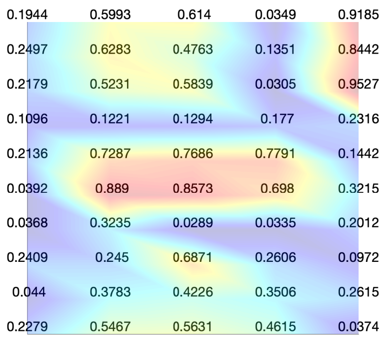


Fig. 3. Heat distribution used to compute D_2, higher values represent a higher dangerousness on the environment.

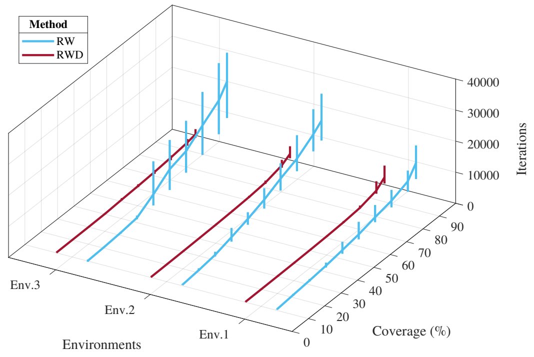


Fig. 4. Comparison between the performance between RW and RWD in all three environments, the graph shows the number of iterations need it to cover certain percentages of the terrain.

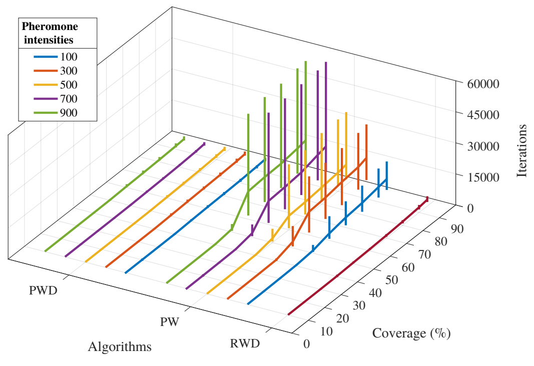
B. Results and Discussion

Comparing the results of the RW and RWD algorithms, it is clear that adding a direction to the agent helps to improve the performance of the swarm. In Fig. 4 is shown that in the three environments RWD reduces the iterations needed for cover each of them by more than 50%.

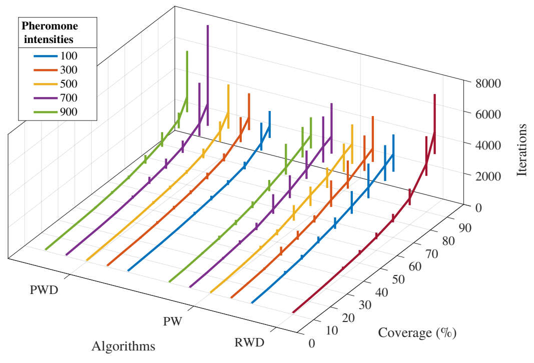
Analyzing RWD against the two pheromone methods, we can see that under certain configuration of Υ^0, the performance of PW can be similar to RWD, however is PWD the one that stand out (Fig. 5). Given this results we can assure that if we add direction to the movement of an agent and give him the ability to communicate its path through the environment with pheromones, the dispersion and coverage of the swarm in the area can be accomplish in a more faster and efficient way.

Fig. 5 also shows that, as expected, while the coverage percentage of the environments increases, the amount of iterations necessary to achieve it also increase. This situation occurs because unexplored points are often very sparse in the environment, which makes difficult for the swarm to find them. This fact also causes that the standard deviation (SD) grows along with the percentage of coverage. A special result is obtained with the Env1 (Fig. 5(a)) in which the data presented a big SD. This occurs because the structuring of the environment, in which, there is only one connection path between two large unexplored areas .

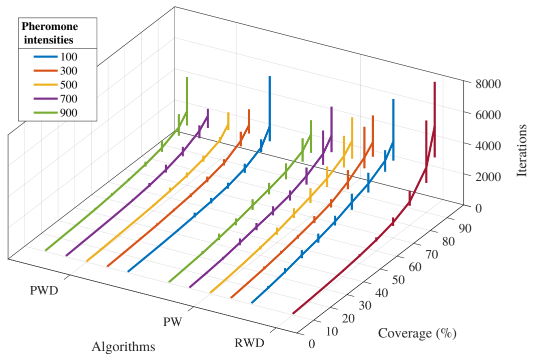
Fig. 6 presents a summary of the PWD results in every



(a) Env1



(b) Env2



(c) Env3

Fig. 5. Results of iterations needed it to cover certain percentages of the environments by the algorithms RWD, PW, PWD

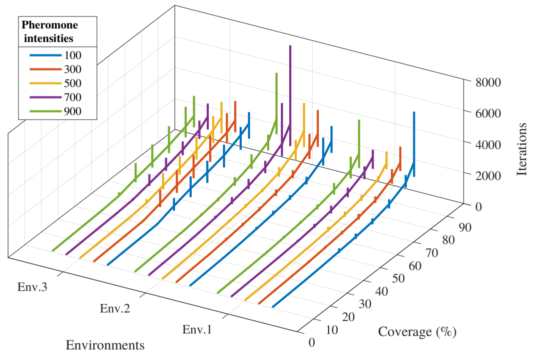
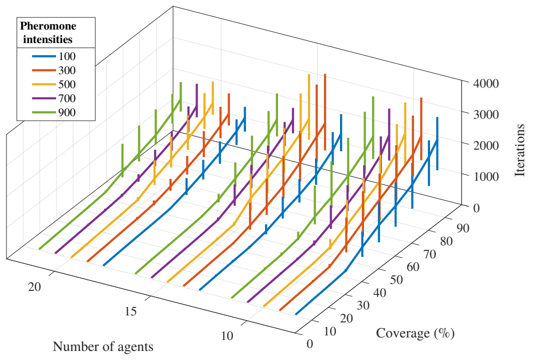
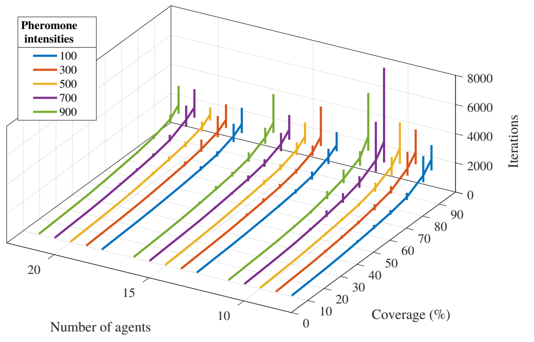


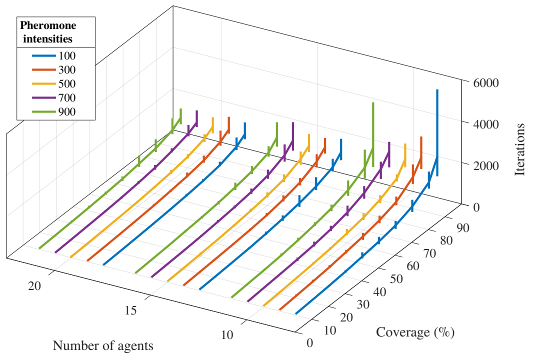
Fig. 6. Summary of the results of PWD in all three environments



(a) Env1



(b) Env2



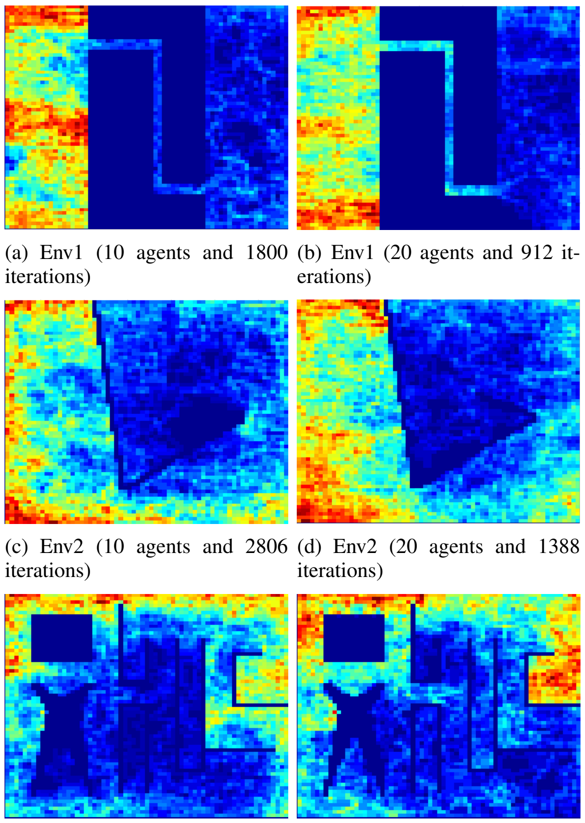
(c) Env3

Fig. 7. Results of the iterations need it to cover certain percentages of the environments by the algorithm PWD with three swarm of 10, 15 and 20 agents

environment and in certain percentages of coverage. In this we can see that there is not an ideal initial pheromone intensity that gives best results in all the configurations, because this will vary depending the environment an the percentage coverage. However we can hypothesize that a low level of Υ^0 will produce similar results to the RWD while highest one will create a very saturated space and will not allow a faster dispersion.

All exploration results were performed by a swarm of ten agents. For comparing the changes that can produce the introduction of a bigger group of simulated robots, the same experiments were preformed by a swarm of fifteen and twenty agents. As expected, while the number of agents increases the faster is the coverage and dispersion of the area (Fig. 7). Also it can be seen that the SD of the data and the differences between the pheromone intensities decreases.

A cellstep heatmap, that shows the number of visits for each cell of the environment, was also created in all the experiments in order to analyze the behavior of the agents. The results show that there are areas that are constantly being explored, this creates what we call over-exploration of the environments. This situation causes that the agents invest time on exploring areas that are already known instead of visiting unexplored ones, what can result on a increases in the iteration need it to accomplish the overall task. In the examples showed in Fig. 8, it can be seen that the area where the agents were deployed are usually highly explored. Also even though the size of the swarm augmented and the number of iterations reduced, the over-exploration is not necessarily decreased.



(a) Env1 (10 agents and 1800 iterations) (b) Env1 (20 agents and 912 iterations)

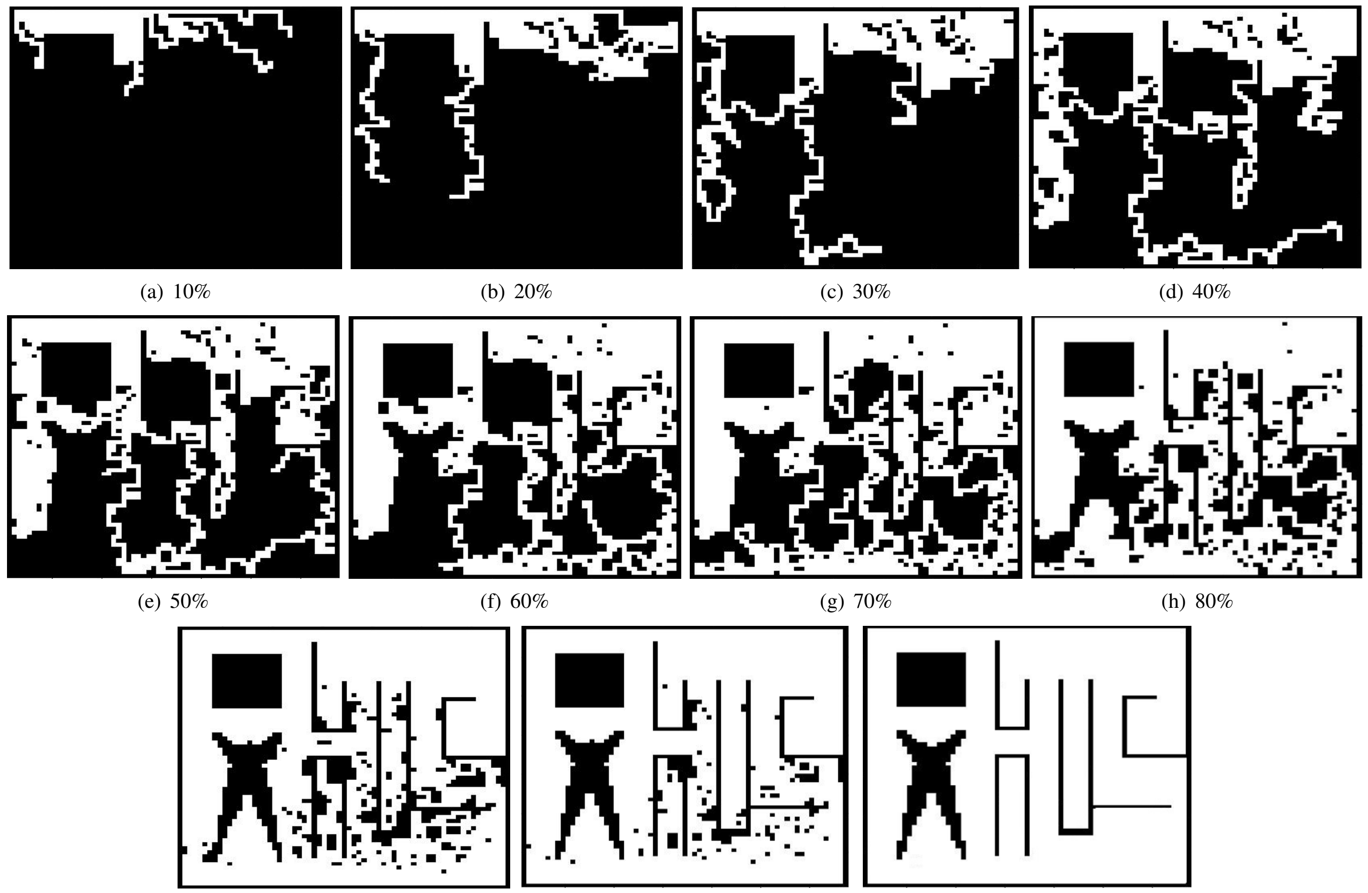
(c) Env2 (10 agents and 2806 iterations) (d) Env2 (20 agents and 1388 iterations)

(e) Env3 (10 agents and 2298 iterations) (f) Env3 (20 agents and 845 iterations)

Fig. 8. Cellstep heatmaps that shows the number of visits for each cell of the environment. The size of the swarm and the number of iterations need it to cover the 95% of the area is described in each case.

At the same time that exploration is occurring, a external server stores all visited cells for construct an environment map. Fig. 9 shows the evolutionary process of creating the map on Env3, guided by the coverage percentage of area. Once the exploration is finished, the Adapted RRG algorithm proposed takes this map to structure the space and find a path, minimizing the distance but taking into account the dangerousness.

Fig. 10 shows a small variations on graph generation time by means of unidirectional and sequential bidirectional search, but variations resulting of parallel bidirectional search are slightly larger. Regardless of search type used, execution time of graph generation increases as mesh becomes more refined. Execution time with parallel bidirectional search is always



(a) 10% (b) 20% (c) 30% (d) 40%

(e) 50% (f) 60% (g) 70% (h) 80%

(i) 90% (j) 95% (k) 100%

Fig. 9. Example of the map creation process of Env3 carried out during the exploration phase, according the coverage percentage of the area

TABLE I. AVERAGE ELAPSED TIME IN 100 EXPERIMENTS BY FINDING THE OPTIMAL PATH, WITH DIFFERENT PARTITION SIZES AND ω VALUES ON ENV3

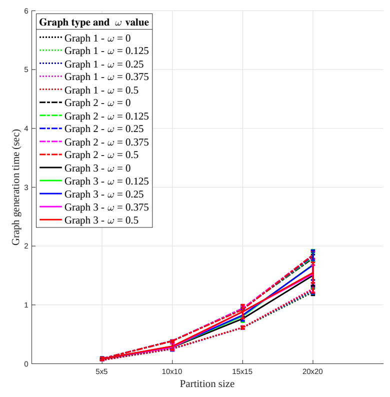
Partition size	Graph type	$\omega = 0$		$\omega = 0.125$		$\omega = 0.25$		$\omega = 0.375$		$\omega = 0.5$	
		mean	std	mean	std	mean	std	mean	std	mean	std
5×5	Graph 1	0.4268	0.1236	0.4127	0.1135	0.4403	0.1431	0.4406	0.1004	0.4393	0.2025
	Graph 2	0.5344	0.1158	0.4380	0.0517	0.6308	0.1609	0.5717	0.1844	0.4810	0.0845
	Graph 3	0.6740	0.1957	0.7324	0.2828	0.7331	0.2666	0.7029	0.2407	0.7215	0.2549
10×10	Graph 1	5.2220	0.2101	5.2524	0.2006	5.2481	0.2114	5.3179	0.1993	5.3145	0.2004
	Graph 2	5.4593	0.1860	5.5821	0.2236	5.6681	0.1879	5.6588	0.2200	5.6591	0.2328
	Graph 3	6.8302	0.4058	7.2021	0.7134	6.8418	0.3448	6.9627	0.4594	7.1016	0.6783
15×15	Graph 1	19.7240	0.4468	20.5150	0.5672	20.6680	0.4568	20.5630	0.5486	20.2910	0.5838
	Graph 2	19.0830	0.5272	20.0130	0.5316	20.0340	0.5244	20.4890	0.5410	20.2180	0.5674
	Graph 3	22.1300	1.4991	24.4360	2.2988	25.1070	2.0405	25.0750	2.0488	25.4720	2.8845
20×20	Graph 1	41.7990	1.1626	44.9960	1.0414	45.3290	0.8837	45.9970	0.9466	46.6880	0.9544
	Graph 2	47.0210	1.1203	51.5370	1.0564	50.4570	1.2023	51.2210	1.1727	51.7330	0.9533
	Graph 3	45.3380	4.0262	43.4460	4.0006	42.3570	2.3770	42.8490	2.6175	45.2010	4.1474

lower than execution time with unidirectional search. Elapsed graph generation time on Env3 is always greater than the elapsed on Env1 and Env2.

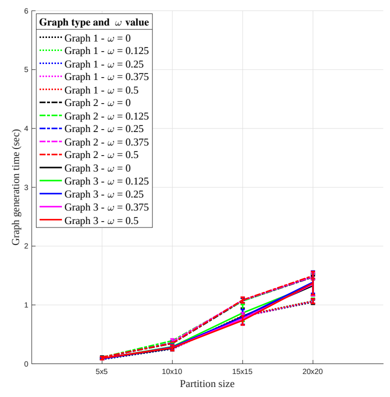
Once the graphs have been generated, DSP algorithm is applied on the same way for three cases: unidirecional, sequential and parallel bidirectional graph search. Table I summarizes the elapsed time at path planning process on Env3, we can see that the variations on Graph 3 (parallel bidirectional search) are also greater than Graph 1 and 2. There are no significant differences in path planning time with different ω values into the same partition, but there are significant differences between path planning time into different partition sizes. Path planning time increases as partition size increases. Results for Env1 and Env2 are quite similar.

The path distance depends on the environment as shown in Fig. 11. Although there are also slight differences between path distance obtained with different ω values, the greatest differences appear with different partition size. Path distance presents larger variations with a coarse mesh, i.e., with a small partition sizes. Better results are obtained as the number of cells increases.

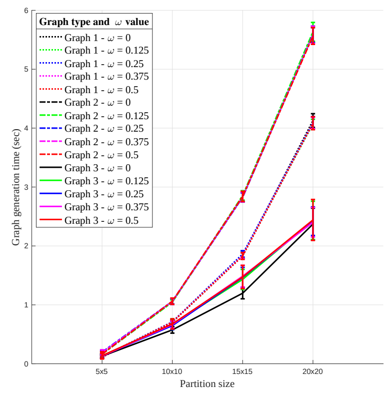
Other general results were made, equivalent to those showed in [24]. The shortest path found improve according to refinement of the mesh, but it also depends on the type of environment. For example 5×5 partition is not always



(a) Elapsed time by adapted RRG algorithm on Env1

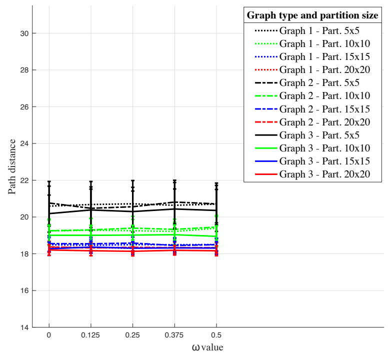


(b) Elapsed time by adapted RRG algorithm on Env2

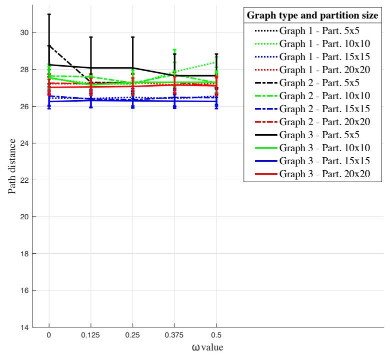


(c) Elapsed time by adapted RRG algorithm on Env3

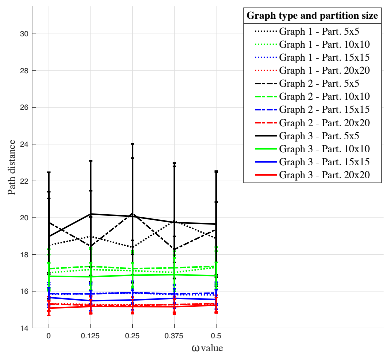
Fig. 10. Average elapsed time of 100 experiments generating graph on each environment, with uni and bidirectional search methods and based on different partition sizes



(a) Path distance obtained on Env1



(b) Path distance obtained on Env2



(c) Path distance obtained on Env3

Fig. 11. Average path distance of 100 experiments in graph generation on each environment, with uni and bidirectional search methods and based on different ω values

successful, not only because in some cases it is not possible to structure all space with the graph and obtain a path, but also because sometimes it is possible to obtain a path but not necessarily the optimal one. Moreover, it is possible to obtain different paths based on different partitions. Even though a finer mesh allows being closer to the optimal solution, it produces a denser and more complex graph.

Fig. 12 shows the performance of graph generation and path planning process in Env3. It presents some differences when using a scalarized objetive function, which combines distance and dangerousness, and when using an objective function

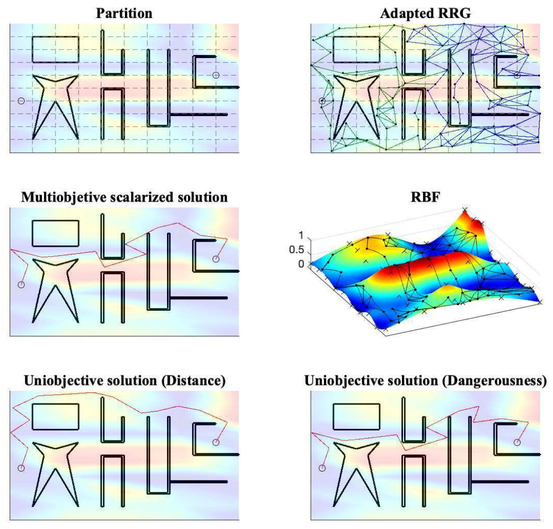


Fig. 12. Experimental results on Env3 with 10×10 as a partition size, ω =0.375 and an parallel bidirectional search

that optimize just one of these elements. It is possible to see the RBF resulting with which dangerousness is calculated and the correponding line integrals.

V. Conclusions

A solution that involves swarm robotics and multiobjectives techniques for finding paths, minimizing distance and danger in an unknown environment, was presented. The solution was composed by three phases: exploration, mapping and path planning, and was made under computer simulations with swarms of ten, fifteen and twenty agents.

Exploration and mapping phases were implemented by means of a cellular automata, in which the environments were mapped into a two dimensional map. For the first phase four algorithms, organized into two approaches, were presented and applied in three different environments.

The best performance scheme was PWD, which provides a direction to the agent movements and give him the ability to communicate its path through the environment. PWD achieves the complete tasks faster and more efficiently than the others. Even though all the experiments were carried out with five different initial amount of pheromone intensity, the ideal value varies depending on the structure of the space and the coverage percentage.

As was expected, the coverage time decreases with increase the number of agents. Besides, as the number of agents increases, differences between results with different pheromone intensities also decrease.

On the other hand, ω values does not interfere with the graph generation and graph type (unidirectional, sequential and parallel bidirectional search) does not affect the optimization process. But both, graph generation and optimization process, always depend on partition sizes.

A fine partition allows a complete work space structuring and to find the optimal path. Shapes and quantity of obstacles scattered, through the environment, will affect the effectiveness of the partition dimension. Results show that even though a more refined partition produces a shorter path, the graph generation will take longer execution time and produce a denser and more complex graph.

Parallel bidirectional search has better performance than unidirectional search, but sequential bidirectional search sometimes has a shorter run-time than parallel one, depending on the environment.

As future work a more robust algorithm has to be created in order to reduce the over-exploration of the environments and a genetic multiobjective optimization algorithm will be used, without scalarizing.

Acknowledgment

Special thanks to the Research and Outreach of the Costa Rica Institute of Technology (VIE, ITCR), for their support to the PROE project (code VIE 1440011). These results are part of that project.

References

[1] J. C. Barca and Y. A. Sekercioglu, “Swarm robotics reviewed,” *Robotica*, vol. 31, no. 3, pp. 345–359, 2013.

[2] I. Navarro and F. Matía, “An introduction to swarm robotics,” *Isrn robotics*, vol. 2013, 2012.

[3] L. Bayındır, “A review of swarm robotics tasks,” *Neurocomputing*, vol. 172, pp. 292–321, 2016.

[4] L. Garattoni and M. Birattari, “Autonomous task sequencing in a robot swarm,” *Science Robotics*, vol. 3, no. 20, p. eaat0430, 2018.

[5] N. Palmieri, X.-S. Yang, F. De Rango, and S. Marano, “Comparison of bio-inspired algorithms applied to the coordination of mobile robots considering the energy consumption,” *Neural Computing and Applications*, pp. 1–24, 2017.

[6] Q. Tang, F. Yu, Y. Zhang, L. Ding, and P. Eberhard, “A stigmergy based search method for swarm robots,” in *International Conference on Swarm Intelligence*. Springer, 2017, pp. 199–209.

[7] D. A. Lima and G. M. Oliveira, “A cellular automata ant memory model of foraging in a swarm of robots,” *Applied Mathematical Modelling*, vol. 47, pp. 551–572, 2017.

[8] C. R. Tinoco, D. A. Lima, and G. M. Oliveira, “An improved model for swarm robotics in surveillance based on cellular automata and repulsive pheromone with discrete diffusion,” *International Journal of Parallel, Emergent and Distributed Systems*, pp. 1–25, 2017.

[9] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[10] R. Valencia and J. Andrade-Cetto, “Path planning in belief space with pose slam,” in *Mapping, Planning and Exploration with Pose SLAM*. Springer, 2018, pp. 53–87.

[11] J. Faigl, “On self-organizing map and rapidly-exploring random graph in multi-goal planning,” in *Advances in self-organizing maps and learning vector quantization*. Springer, 2016, pp. 143–153.

[12] A. Bircher, K. Alexis, U. Schwesinger, S. Omari, M. Burri, and R. Siegwart, “An incremental sampling-based approach to inspection planning: the rapidly exploring random tree of trees,” *Robotica*, vol. 35, no. 6, pp. 1327–1340, 2017.

[13] A. D. Mali, “Probabilistic roadmaps with higher expressive power,” *International Journal on Artificial Intelligence Tools*, vol. 25, no. 04, p. 1650027, 2016.

[14] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] I. Noreen, A. Khan, and Z. Habib, "A comparison of rrt, rrt* and rrt*-smart path planning algorithms," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 16, no. 10, pp. 20–27, 2016.

[16] B. Saicharan, R. Tiwari, and N. Roberts, "Multi objective optimization based path planning in robotics using nature inspired algorithms: A survey," in *International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*. IEEE, 2016, pp. 1–6.

[17] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[18] H. Wang, Y. Yu, and Q. Yuan, "Application of dijkstra algorithm in robot path-planning," in *Second International Conference on Mechanic Automation and Control Engineering (MACE)*. IEEE, 2011, pp. 1067–1069.

[19] R. Arjun and P. Reddy, "Shama, and m. yamuna,"research on the optimization of dijkstra's algorithm and its applications"," *International Journal of Science Technology and Management*, vol. 4, no. 1, pp. 304–309, 2015.

[20] C. Calderón-Arce and P. Alvarado-Moya, "Optimización multiobjetivo con funciones de alto costo computacional. revisión del estado del arte," *Revista Tecnología en Marcha*, pp. 16–24, 2016.

[21] N. Gunantara, "A review of multi-objective optimization: Methods and its applications," *Cogent Engineering*, vol. 5, no. 1, pp. 1–16, 2018.

[22] C. Dimidov, G. Oriolo, and V. Trianni, "Random walks in swarm robotics: an experiment with kilobots," in *International Conference on Swarm Intelligence*. Springer, 2016, pp. 185–196.

[23] R. Morlok and M. Gini, "Dispersing robots in an unknown environment," in *Distributed Autonomous Robotic Systems 6*. Springer, 2007, pp. 253–262.

[24] C. Calderón-Arce, R. Solís-Ortega, and T. Bustillos-Lewis, "Path planning on static environments based on exploration with a swarm robotics and rrg algorithms," in *38th Central America and Panama Convention (CONCAPAN XXXVIII)*. IEEE, 2018, pp. 1–6.

[25] T. C. Hales, "The jordan curve theorem, formally and informally," *American Mathematical Monthly*, vol. 114, no. 10, pp. 882–894, 2007.

[26] H. Yu, T. Xie, S. Paszczyñski, and B. M. Wilamowski, "Advantages of radial basis function networks for dynamic system design," *Transactions on Industrial Electronics*, vol. 58, no. 12, pp. 5438–5450, 2011.

[27] G. A. Barnett, N. Flyer, and L. J. Wicker, "An rbf-fd polynomial method based on polyharmonic splines for the navier-stokes equations: Comparisons on different node layouts," *arXiv preprint arXiv:1509.02615*, 2015.