Expert Systems with Applications 42 (2015) 5177–5191

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Path planning for mobile robots using Bacterial Potential Field for avoiding static and dynamic obstacles

Oscar Montiel *, Ulises Orozco-Rosas, Roberto Sepúlveda

Instituto Politécnico Nacional, CITEDI, Av. del Parque No. 1310, Mesa de Otay, 22510 Tijuana, B.C., Mexico

ARTICLE INFO

Article history:
Available online 27 February 2015

Keywords:
Path planning
Bacterial Potential Field
Bacterial Evolutionary Algorithms
Mobile robots
Collision avoidance

ABSTRACT

In this paper, optimal paths in environments with static and dynamic obstacles for a mobile robot (MR) are computed using a new method for path planning. The proposed method called Bacterial Potential Field (BPF) ensures a feasible, optimal and safe path. This novel proposal makes use of the Artificial Potential Field (APF) method with a Bacterial Evolutionary Algorithm (BEA) to obtain an enhanced flexible path planner method taking all the advantages of using the APF method, strongly reducing its disadvantages. Comparative experiments for sequential and parallel implementations of the BPF method against the classic APF method, as well as with the Pseudo-Bacterial Potential Field (PBPF) method, and with the Genetic Potential Field (GPF) method, all of them based on evolutionary computation to optimize the APF parameters, were achieved. A simulation platform that uses an MR realistic model was designed to test the path planning algorithms. In general terms, it was demonstrated that the BPF outperforms the APF, GPF, and the PBPF methods by reducing the computational time to find the optimal path at least by a factor of 1.59. These results have a positive impact in the ability of the BPF path planning method to satisfy local and global controllability in dynamic complex environments, avoiding collisions with objects that will interfere the navigation of the MR.

1. Introduction

The problem of autonomous navigation of a mobile robot (MR) consists in taking it from one position to another one without the assistance of a human operator, in particular, planning a reachable set of MR configurations to accomplish its mission. To solve this problem, it is necessary to have a methodology that allows guiding the MR to accomplish a set of navigation and operation goals, i.e., its mission. In general, this methodology deals with motion planning that includes two different but complementary tasks, path planning and trajectory planning. The first one consists in designing a dynamical system that can drive the MR from an initial position (state) to a target position (goal), whereas, trajectory planning focuses on determining how to move the MR along the solution given by the path planning algorithm in a way that the mechanical limitations of the MR are respected.

In this paper, we propose a new approach called Bacterial Potential Field (BPF) for path planning in MRs. The BPF proposal is based on Artificial Potential Field (APF) enhanced with a Bacterial Evolutionary Algorithm (BEA) to solve the limitations given by the original APF. At the present, this is the first implementation of a BPF algorithm for path planning for MRs.

The original APF method is a mathematical method widely used in autonomous navigation of MRs given that it provides an effective control on the movement (Kim, Heo, Wei, & Lee, 2011); some derivations of this method have been used for path planning (Chen, Di, Huang, Sasaki, & Fukuda, 2009a; Goerzen, Kong, & Mettler, 2010; He, Gao, & Nan, 2011; Kim et al., 2011; Weijun, Rui, & Chongchong, 2010). Nevertheless, it presents some limitations; the solutions were found to be subject to local minima (He et al., 2011), and far from the optimum in many cases, given that the planning was solely local and reactive (Chen, Lindsay, Robinson, & Abbass, 2009b). The path planning with the BPF proposal allows the MR to navigate in an autonomous form without being trapped in local minima, making the BPF proposal suitable to work in dynamic environments, which is very crucial in real-world applications. Some significant advantages of the BPF over the original APF method, as well as its derivatives with Evolutionary Artificial Potential Field (EAPF) are summarized in Table 1.

In Section 2, a bibliographic review of some important related work that embraces classical (including the APF), heuristic and meta-heuristic approaches of robot motion planning is provided. In Section 3, the main theoretical contributions of this work are stressed, as well as the pseudocode to implement the method is

* Corresponding author.
E-mail addresses: oross@ipn.mx (O. Montiel), uorozco@citedi.mx (U. Orozco-Rosas), rsepulvedac@ipn.mx (R. Sepúlveda).

http://dx.doi.org/10.1016/j.eswa.2015.02.033
0957-4174/

Table 1
Comparison between APF, EAPF and BPF, when they are used for the on-line global planning.

	APF	EAPF	BPF
Local planning	✓	✓	✓
Smooth path	✓	✓	✓
Global planning		✓	✓
Automatic gain search		✓	✓
Small time consumption			✓
Complex environments			✓
Adaptability to the environment			✓
Power of convergence			✓
Highly scalable			✓

given and explained. In Section 4, the conducted experiments for the BPF proposal are explained, analyzed and compared in performance with two EAPFs, in specific with a Genetic Potential Field (GPF) method based on a GA, and a second one EAPF called Pseudo-Bacterial Potential Field (PBPF) method based on a Pseudo-Bacterial Genetic Algorithm. Finally, in Section 5, conclusions and future work are provided.

2. Related work

The origins of robot motion planning can be tracked to the middle of the 1960s (Masehian & Sedighizaddeh, 2007). It has been dominated by classical approaches such as the Roadmap, Cell Decomposition, Mathematical Programming and APF. Representative proposals of Roadmaps approaches are the Visibility graph which is a collection of lines in the free space that connects the trait of an object to another; the Voronoi diagram of a collection of geometric objects is a partition of space into cells, each of which consists of the points closer to one particular object than any other (Eppstein, 1996); in the Subgoal Network, a list of reachable configurations from the start configuration is maintained. The Silhouette approach consists of generating the silhouette of the work cell and developing the Roadmap by connecting these silhouettes curves to each other (Bhattacharyya, Singla, & Dasgputa, 2007). The idea of Cell Decomposition algorithms is to decompose the *C*-space into a set of simple cells, and then compute the adjacency among cells. In the Mathematical Programming approach, the requirement of obstacle avoidance is represented by a set of inequalities on the configuration parameters; the idea is to minimize certain scalar quantities to find the optimal curve between the start and goal position (Siegwart, Nourbakhsh, & Scaramuzza, 2011). The APF concept was introduced by Khatib (1986). At the beginning, this method was used for obstacle avoidance in path planning for robot manipulators, through the time, it has been well adopted for MRs and groups of MRs, an example is presented in Barnes, Fields, and Valavanis (2009) where is proposed a swarm formation control with potential fields to control robot swarm formation, obstacle avoidance and swarm movement as a whole. In the APF, an MR is treated as a point representing in the configuration space a particle under the influence of an APF denoted by $U(q)$ whose local variations reflect the free space structure; the idea behind the APF method is to establish an attractive potential field force around the goal point, as well as to establish a repulsive potential field force around the obstacles. The two potential fields together (attractive + repulsive) form the total potential field called APF (Park & Lee, 2003).

In Masehian and Sedighizaddeh (2007) an amount of 1381 papers dating from 1973 to 2007 were surveyed, covering a sufficient depth of works in the robot motion planning field. In this work, a broad classification of heuristic techniques is presented, which facilitates its analysis and method's expectations. Broadly, the given classification is as follows: Probabilistic, heuristic and meta-heuristic approaches (Zhang, Chen, & Fei, 2006). In the former are the Probabilistic Roadmaps, Rapidly-exploring Random Trees, Level set and Linguistic Geometry. In the heuristic and meta-heuristic approaches are the Neural Networks, Genetic Algorithms (GAs) (Berger, Jabeur, Boukhtouta, Guitouni, & Ghanmi, 2010; Hocaoglu & Sanderson, 2001; Li, Ding, Cai, & Jiang, 2010), Simulated Annealing (Zhang, Collins, & Barbu, 2013), Ant Colony Optimization (Montiel, Sepúlveda, Castillo, & Melin, 2013), Particle Swarm Optimization, Stigmergy, Wavelets, Tabu Search and Fuzzy Logic. All the mentioned methods have their own strengths and drawbacks; they are deeply connected to one another, and in many applications, some of them were combined together to derive the desired robotic controller in the most effective and efficient manner.

In this paper, the BPF is proposed as a method for path planning for mobile robotics that ensures a feasible, optimal and safe path for the robot navigation. The BPF proposal uses concepts from the APF, mathematical programming, and meta-heuristic to solve efficiently a robot path planning problem, ensuring a reachable configuration set and controllability if it exists, outperforming current APF approaches. The APF is a reactive motion planning method with inherent well known difficulties to travel finding global optimal paths, because it cannot solve all local minima problems (Zhang, Chen, & Chen, 2012); hence, modern methods that overcome these challenges have been developed (Vadakkepat, Lee, & Xin, 2001). One of them is where the APF is blended with Evolutionary Algorithms (EA) obtaining a different potential field methodology named Evolutionary Artificial Potential Field (EAPF) Vadakkepat, Tan, and Wang (2000), here, the APF method is combined with GAs to derive optimal potential field functions (Vadakkepat et al., 2001). The variational planning approach uses the potential as a cost function, and it attempts to find a path to reach the goal point that minimizes this cost (Goerzen et al., 2010).

Furthermore from the GAs, Nawa, Hashiyama, Furuhashi, and Uchikawa (1997) proposed a novel kind of evolutionary algorithm called Pseudo-Bacterial Genetic Algorithm (PBGA) which was successfully applied to extract rules from a set of input and output data. This algorithm introduced a genetic operation called *bacterial mutation* that has demonstrated to be useful in environments with a weak relationship between the parameters of a system. It is a simple algorithm that presents a fast convergence and improvement in the solutions (Botzheim, Gál, & Kóczy, 2009, chap. 3; Botzheim et al., 2011), without detrimental in the landscape exploration. Furthermore, Nawa and Furuhashi (1999) proposed the Bacterial Evolutionary Algorithm (BEA) for fuzzy rule base extraction. This algorithm was based on PBGA supported by a new genetic operation called *gene transfer*, which establishes relationships among the individuals from the population; it can also be used for decreasing or increasing the number of the rules in a fuzzy rule base. Both the PBGA and the BEA are global search methods (Botzheim et al., 2009, Botzheim, Toda, & Kubota, 2010, cha 3). The convergence to the global optimal of the BEAs in comparison with the GAs is faster, which is important since the calculation process cost of combination is omitted (Fallah, Akbari, & Javan, 2010). The bacteriological approach is more an adaptive approach than an optimization approach as with GAs. It aims at mutating the initial population to adapt it to a particular environment. The adaptation is only based on small changes in the individuals. The individuals within the population are called bacteria and correspond to atomic units. Unlike the genetic model, the bacteria cannot be divided. The crossover operation cannot be used anymore. Bacteria can only be reproduced and altered to improve the population (Baudry, Fleurey, Jézéquel, & Traon, 2005).

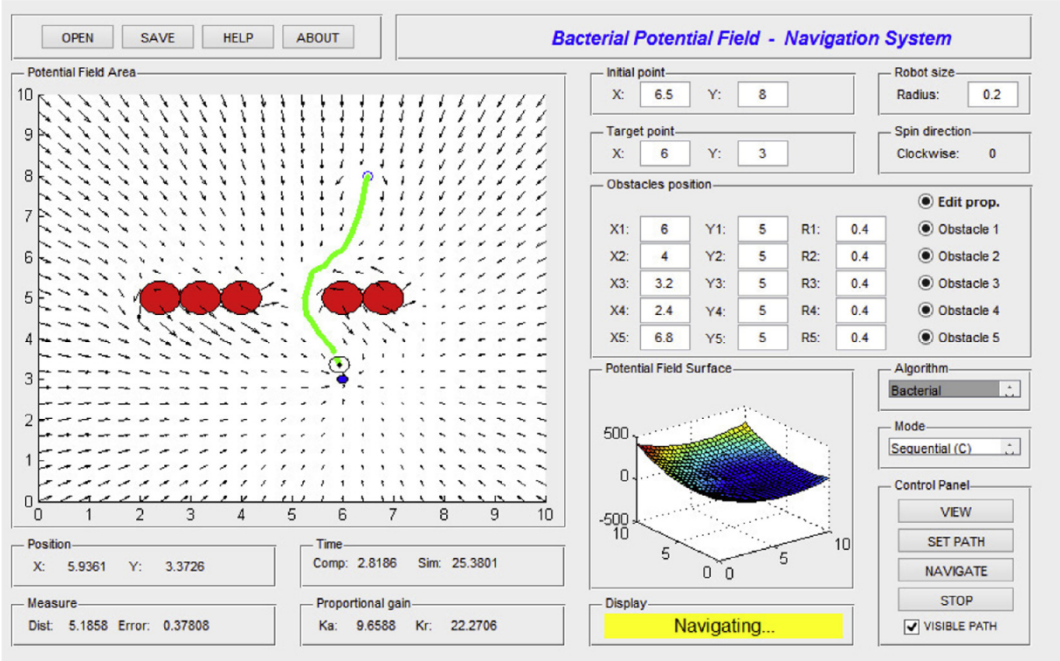


Fig. 1. Experimental simulation platform.

3. BPF path planner development

Path planning is generating a collision-free path (feasible), in an environment with obstacles and optimizing it with respect to some criterion (Sedighi, Ashenayi, Manikas, Wainwright, & Tai, 2004) such as distance, time or energy, distance being the most commonly adopted criterion and used for this work. The next points are taken into consideration for the development of the BPF path planning method:

- A path $\mathcal{P}$ is a set of configurations $\vec{q} = [q_0, q_1, \ldots, q_f] \in \mathbb{R}^n$ of the MR that connects the starting position q_0 to the final position q_f.
- Since, having a reachable configuration set and controllability are intimately related, the former must exist in order to accomplish the control task of steering the MR through $\vec{q}$ fulfilling the requirement of the total time $T = \sum_{i=1}^{m} T_{st}^i$ imposed by the system designer for global controllability.
- The small-time local (T_{st}) controllability of the system implies that the MR can go from any point in $\mathcal{P}$, say q_a to q_b, in a time given by $T_{st} = t_p + t_j$, where t_p is the time that the planner takes to calculate the path, and the time t_j is the time consumed by the trajectory planner and following algorithm, which depends on the MR architecture.
- For static environments in on-line planning with an already defined $\mathcal{P}$, the time t_p is zero since there were no changes in scenery. However, for sceneries with dynamic environments, the time t_p rises its importance since slow path planning algorithms will not fulfill the local and global controllability criteria.

The BPF proposal is an original path planning method that uses a potential function (Khatib, 1986) as the cost function to find optimal paths; hence, the BPF and APF methods have a common mathematical foundation; in this work, they have the next similarities:

- The MR q, is in the 2-D Euclidian coordinate system.
- The MR is modeled by a circle with radius r, center $c = (x, y)$, and orientation θ; therefore, one position (configuration) of the MR in the system is given by $q = (c, r, \theta) \in \mathbb{R}^n$. In the APF, $r = 0$, but in BPF, r serves to the represent physical dimension of the MR.
- They need a start position q_0, a target position q_f, and the obstacles position $O_1, \ldots, O_n$.
- They use the potential function $U_{total}(q)$ given by (1), it comprises two terms, the attractive potential function $U_{att}(q)$ described by (2), and the repulsive potential function $U_{rep}(q)$ described in (3).

$$U_{total}(q) = U_{att}(q) + U_{rep}(q) \tag{1}$$

$$U_{att}(q) = \frac{1}{2} k_a (q - q_f)^2 \tag{2}$$

$$U_{rep}(q) = \begin{cases} \frac{1}{2} k_r \left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho \leqslant \rho_0 \\ 0 & \text{if } \rho > \rho_0 \end{cases} \tag{3}$$

- k_a in (2) and k_r in (3) are scalar variables that represent the attractive and repulsive proportional gains of the functions, respectively.
- In (3), ρ_0 is the limit distance of influence of the potential field, and ρ is the shortest distance to the obstacle.
- The MR navigates using the total force given by $F(q) = -\nabla U_{total}(q)$.

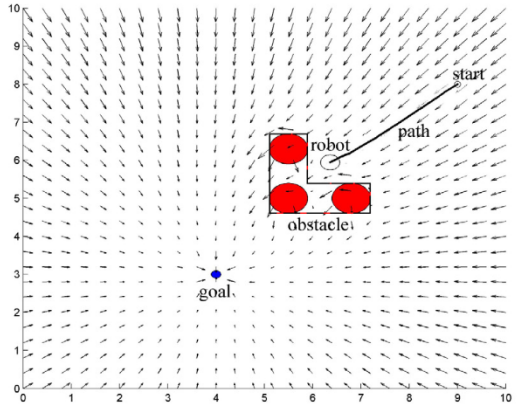
3.1. BPF Algorithm

The BPF proposal uses the start, goal and obstacle positions as features to obtain a sequence of objective points that the MR must attain, and gradient information influenced by the attractive and repulsive forces to transform a sequence of objective points to a path $\mathcal{P}$, hence the proposal achieves the task of path planning generation, with the particular characteristics that it provides an optimal or nearly optimal reachable set of configurations (path) if it exists in dynamic environments, at very low computational cost.

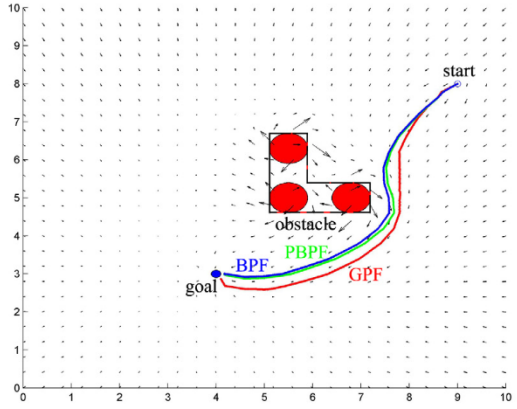
The classic APF method cannot guarantee to reach the goal position in many cases, literature has reported some of them (He et al.,

2011; Lee, 2004; Zhang et al., 2006). In this paper, we previously referred to some popular problems that have been identified as limitations for the APF method; however, using the BPF proposal, the previous problems have been solved successfully and described in Section 4.

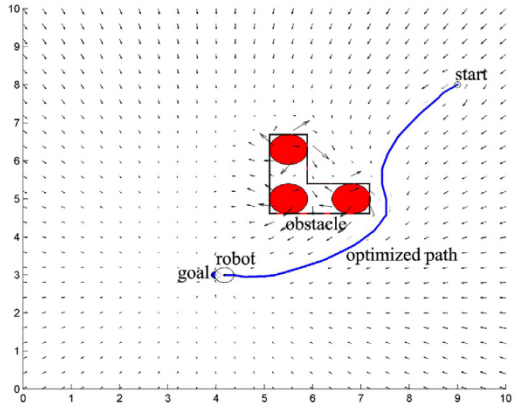
The backbone of this novel proposal is the use of bacterial evolutionary computation, which enables the MR to find at a low computational time the optimal values $k_{a(opt)}$ and $k_{r(opt)}$'s that correspond to the attractive and repulsive forces, respectively. This characteristic is very important since it allows the MR to navigate without being trapped in local minima, making the BPF method suitable to work in dynamic environments, which is critical in real-world applications. The aforementioned is crucial, because the traditional practice of interacting with the user to change the gains by trial and error is a limitation of the classic APF method.



(a) Local minima problem



(b) Found suitable paths using different algorithms



(c) Local minima problem solved and optimal path found

Fig. 2. Path planning for test problem 1: Local minima.

In Algorithm 1, the pseudocode of the BPF method is given; the proportional gains evolve by means of the BEA which uses the APF method that is explained in Algorithm 3 as the fitness function. Particularly, our prototype BEA function is given by (4),

$$[var_1,\ldots,var_n,fitValue]_{opt} = \mathrm{BEA}([var_1,\ldots,var_n],BEA_{param},FitFunc,N_p) \tag{4}$$

where at the right hand of the function, $[var_1,\ldots,var_n]$ are the parameters that are going to be optimized using a specific mutation rate, number of infections, number of bacteria, etc., with the aim of fulfilling an optimization criterion according to the fitness function given by *FitFunc*, the last parameter N_p is the number of processors to be used. The returning values are the optimized variables $[var_1,\ldots,var_n]$ and the corresponding fitness value *fitValue*.

Algorithm 1. BPF path planner

1: **procedure** BPF($q_0, q_f, O_N, \eta, \epsilon, M, N_p$)
2: $FitFunc = @$ APF $(q_0, q_f, k_a, k_{rN}, O_N, \eta, \epsilon, M)$ ▷Assign a function handler to the APF fitness function
3: $[k_a, k_{rN}, \eta, d_a, m, goal] \leftarrow$ BEA $([k_a, k_{rN}, \eta], BEA_{param}, \mathrm{FitFunc}, N_p)$
4: **if** $goal == 1$ **then** ▷Target achieved
5: $k_{a(opt)} \leftarrow k_a$
6: $[k_{r1(opt)},\ldots,k_{rN(opt)}] \leftarrow [k_{r1},\ldots,k_{rN}]$
7: $\eta_{(opt)} \leftarrow \eta$
8: $M \leftarrow m+1$
9: APF $(q_0, q_f, k_{a(opt)}, k_{rN(opt)}, O_N, \eta_{(opt)}, \epsilon, M)$ ▷Performs optimal path planning
10: **else**
11: Target not achieved, a reachable path does not exist
12: **end if**
13: **end procedure**

Therefore, the input parameters of the BPF method are those required by the BEA and APF algorithms. In the case of one processors ($N_p = 1$); we are using the term *BPF* just to distinguish the obtained results with sequential programming, from those that use parallel programming running on several processors. In line 2 of Algorithm 1 a function handler *FitFunc* is used to facilitate the pseudocode writing, it is equivalent to write the APF fitness function, including all its parameters (Algorithm 3). In line 3, the BEA evolves the parameters, $k_a, [k_{r1},\ldots,k_{rN}]$, and η, to obtain the corresponding optimum values if the goal was achieved; i.e,

Table 2
Results for test problem 1.

Factor	APF	GPF	PBFP	BPF
Gain k_a	7.00	6.46	7.59	1.38
Gain k_r	10.00	43.35	42.60	7.02
Last position x	6.36	4.14	4.17	4.14
Last position y	5.94	2.91	3.01	3.01
Distance d_a	3.776	0.173	0.173	0.173
Path length	3.350	8.338	8.088	7.938
Num of config m	2000	466	438	391
Reached goal	No	Yes	Yes	Yes
Collision	No	No	No	No

$goal = 1$, in this case with the d_a value the user can verify that the MR has reached the goal; i.e., $d_a \leqslant \epsilon$. Otherwise, it returns $goal = 0$ which means that the goal was not achieved, in this case the BEA returns the best values found so far (not necessarily the optimal values) as well as the final distance, d_a, to the obstacle to know how far the MR is from the goal. For the case when $goal = 1$, lines 4–9 are to emphasize that the APF function in line 9 will use the optimal values to perform the desirable path planning. When $goal = 0$, in line 10, special programming instructions can be included to indicate the fact that the goal was not achieved.

3.2. Bacterial Evolutionary Algorithm

The Bacterial Evolutionary Algorithm (BEA) introduces two operations inspired by the microbial evolution phenomenon. The *bacterial mutation* operation which labor is to optimize the chromosome of a single bacterium, and the *gene transfer* operation to provide the transfer of information between the bacteria in the population (Botzheim & Kóczy, 2004).

Algorithm 2. BEA

1: **procedure** BEA($[var_1, \ldots, var_n], BEA_{param}, FitFunc, N_p$)
2: $t \leftarrow 0$ ▷Iteration counter
3: **Create** an initial population $P(t)$ of N_B bacteria
4: **Evaluate** each bacterium in $P(t)$
5: **while** not termination **do**
6: **Perform bacterial mutation** $P'(t)$
7: **Perform genetic transfer** $P'(t)$
8: **Divide** the workload by N_P processors to obtain $P_i(t)$ subpopulations
9: **for each** processor **do** in parallel
10: **Evaluate** each chromosome of subpopulation $P_i(t)$ with APF
11: $P'(t) \leftarrow \bigcup_{\forall i} P_i(t)$
12: $P(t+1) \leftarrow P'(t)$
13: $t \leftarrow t+1$
14: **end while**
15: **return** $[var_1, \ldots, var_n, fitValue]_{opt}$
16: **end procedure**

In the most basic form, the BEA can be algorithmically modeled for computer simulation using the difference equation described by (5),

$$P(t+1) = s(v(P(t))) \tag{5}$$

where t represents the time, the new population $P(t+1)$ is obtained from the actual population $P(t)$ after it was operated by random variation v, and selection s (Fogel, Bäck, Hammel, & Schwefel, 1998, chap. 1). In this context, considering that Ω is a subset of $\mathbb{R}^n$ known as the constrained set (feasible set), a solution vector $p \in \Omega$ is called an individual (bacterium) which is made up of discrete units. Each discrete unit controls one or more features of the bacterium; the units are assumed by default to be binary bits (Mitchel, 2001). The population is a collection of N_B bacteria that are randomly initialized. The BEA uses the bacterial mutation and genetic transfer operations to generate new solutions from existing ones.

Algorithm 2 is intrinsically parallel. The divide-and-conquer strategy can be applied in many different ways; we have used the Single-population master–slave approach, in which, one master node executes the BEA operations working on a single population $P'(t)$, and the fitness evaluation of the individuals in $P'(t)$ is divided in $P_i(t)$ subpopulations distributed among several slaves processors (Cantú-Paz, 2001). For the particular case where the input parameter (N_p) is one, we have the sequential implementation; the other input parameters are: an array of variables $[var_1, \ldots, var_n]$ that will conform the bacteria, the BEA parameters are contained in BEA_{param} where the information about the mutation rate, number of infections, stop condition, etc. is given, and the fitness function *fitFunc*. The algorithm will return an array

Table 3
Computation time to solve the test problem 1 with BPF.

BPF	Sequential		Parallel
	Matlab	C/C++	CPU
Mean (μ)	4.91 s	2.30 s	0.74 s
Std. Dev. (σ)	1.36	0.26	0.14

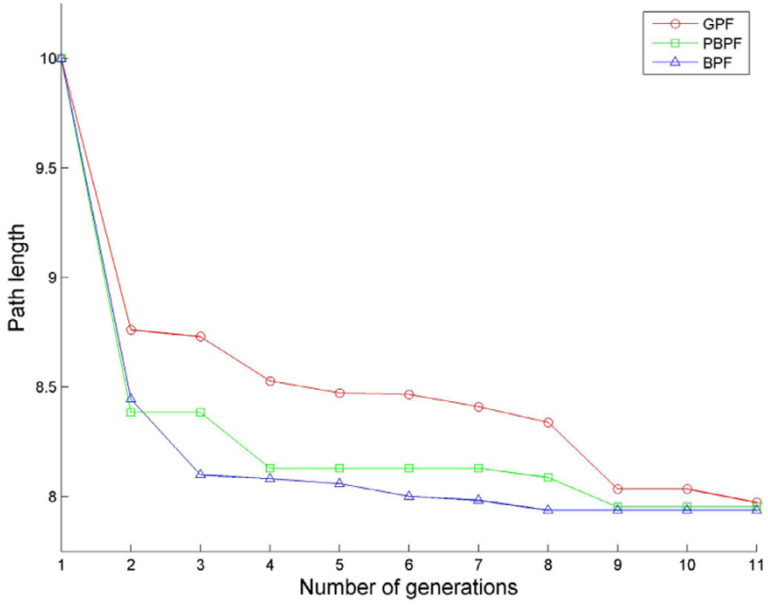


Fig. 3. Plot of convergence of the algorithms: GPF, PBPF, and BPF for the test problem 1.

containing the optimized variables, and the best fitness value *fitValue* found.

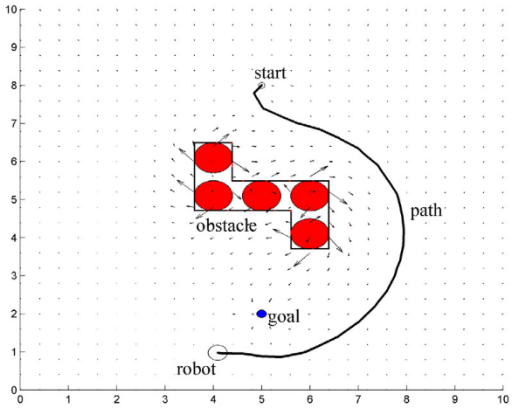
3.3. APF fitness function

The APF method has great advantages in the path planning for robots, where the basic inputs from the user are the starting position, the goal position and the value of attractive and repulsive potential gains. In this work, Algorithm 3 is based on the APF approach, and it has been designed to be used as the fitness function of the proposed BPF method. The classic APF is capable of find feasible paths, and the goal position is reached by the MR in an autonomous form; this is achieved avoiding obstacles that lie between set points. Unfortunately for the classic APF approach, it is necessary to provide the method with the appropriated gain values (usually, the programmer provides the value by trial and error); otherwise, it is very likely that feasible paths are not found; hence the importance of finding the optimal set of gains. It is well known that such task in path planning applications is hard-work because the problem leads to several computational problems, some of them, at the present, they are still being open problems since for large spaces, there are no convincing solutions since the existing ones can last a lot of time making the MR controllability unfeasible. Therefore, the importance of having a method, such as the BPF, that can find the optimal parameters at high-computational speeds, making the robotic system controllable in huge and dynamic sceneries.

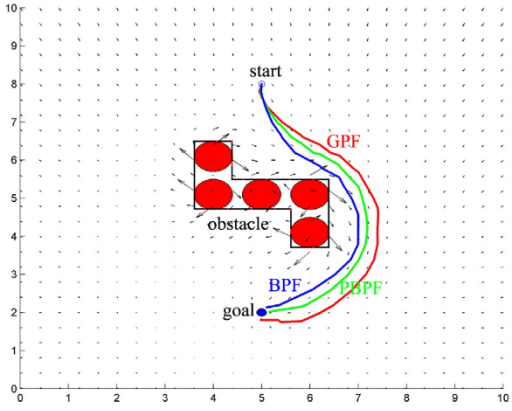
Algorithm 3. APF (fitness function)

1: **procedure** APF(q_0, q_f, k_a, k_{rN}, O_N, η, ϵ, M)
2: $m \leftarrow 0$ ▷Iteration counter
3: $d_a \leftarrow \sqrt{q_f^2 - (q(0) \leftarrow q_0)^2}$ ▷d_a is the distance from the desired to the start position of the MR
4: **while** ($d_a > \epsilon$ and $m < M$) **do**
5: $U_{total}(q(m)) \leftarrow U_{att}(q(m)) + \sum_{i=1}^{N}(U_{rep}(q(m)))$
6: $F(q(m)) \leftarrow -\nabla U_{total}(q(m))$
7: $q(m+1) \leftarrow q(m) + \eta * F(q(m))/\|F(q(m))\|$ ▷MR configuration position
8: $d_a \leftarrow \sqrt{q_f^2 - q(m)^2}$
9: $m \leftarrow m + 1$
10: **end while**
11: **if** $d_a \leqslant \epsilon$ **then**
12: ▷The algorithm returns the path planned (MR configuration position array, q)
13: **return** $[q, d_a, m, goal \leftarrow 1]$ ▷Target position achieved (goal = 1)
14: **else**
15: **return** $[q, d_a, m, goal \leftarrow 0]$ ▷Target position not achieved (goal = 0)
16: **end if**
17: **end procedure**

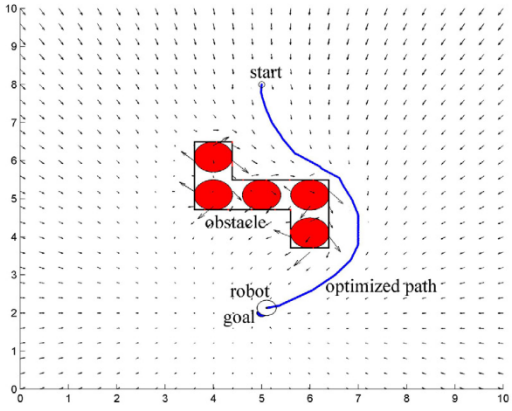
The Algorithm 3 requires the next set of values: a starting position ($q(0) \leftarrow q_0$), a goal position (q_f), the proportional gains (k_a and k_{rN}) which are the gains of attractive and repulsive potential respectively, where N represents the number of obstacles. The step size (η) used in the iterative calculation of the next position q (step 7), the convergence radius (ϵ) that can be also be given by the user as an input to the algorithm, and the number of maximum number of iterations (M) which represents the robot configuration position. The parameters ϵ and M are the algorithm stop conditions, as it is shown in step 4. Note, that depending on the number of obstacles $O_N = [O_1, \ldots, O_N]$ several $[k_{r1}, \ldots, k_{rN}]$ repulsive proportional gains may exist. In the APF, the k_a and k_r's gains are provided heuristically, therefore, for being compatible and achieving fair comparisons against the APF, only one k_r is used for all the obstacles. The algorithm starts calculating the Euclidean distance (d_a) to determinate de distance between the MR position and the goal position. Next, it enters into a loop to move the MR step by step. Inside the loop, the $U_{total}(q)$ (APF) is calculated using equation of



(a) Trajectory prediction problem



(b) Found suitable paths using different algorithms



(c) Trajectory prediction problem solved and optimal path found

Fig. 4. Path planning for test problem 2: Trajectory prediction.

Table 4
Results for the test problem 2.

Factor	APF	GPF	PBFP	BPF
Gain k_a	1.50	5.71	4.20	15.31
Gain k_r	23.00	31.68	20.76	45.80
Last position x	4.09	5.12	5.16	5.11
Last position y	0.98	1.88	2.05	2.13
Distance d_a	1.367	0.173	0.173	0.173
Path length	11.213	8.174	7.903	7.562
Num of config m	2000	555	507	492
Reached goal	No	Yes	Yes	Yes
Collision	No	No	No	No

step 5 based on (2) and (3), where N is the number of obstacles. Step 6 consists in calculating the negative gradient of $U_{total}(q)$ to obtain the generalized force $F(q)$, which is used in step 7 to iteratively calculate the new position of the MR, in step 8 it is calculated the new existing distance d_a between the actual MR position and the goal position. The loop iterates until the stop conditions are presented. From steps 11–17, the Algorithm 3 verifies whether the MR reached the target or not and returns the path planned. For the failure case, the target position has not been achieved, and probably the MR was trapped in a local minimum problem, trajectory prediction problem or on a goal non reachable problem, these are the most common problems in classic APF method, but successfully solved by the BPF method as it is described in Section 4.

3.4. Implementation

The focus on this work is path planning with BPF method. To test the method in off-line and on-line mode, we have developed an experimental navigation system that is described by Algorithm 4 and it was implemented in Matlab/C/C++, see Fig. 1. The Algorithm 4 is used by the MR to navigate from the start to the goal position. The navigation algorithm uses the BPF algorithm for path planning, before and during the navigation. Before the navigation starts, the BPF algorithm is called to get the path that will drive the MR. At this first stage the path planning is known as off-line path planning. Then when the navigation has started in the while loop, the MR is driven point to point following the points contained in $\mathcal{P}$. The BPF algorithm will be called when the environment suffers a change (e.g., some obstacles are added, or they are moving), in this stage the process is known as on-line path planning. Line 13 in Algorithm 4 remarks that it is necessary to achieve trajectory planning for real robots before navigate from the point $\mathcal{P}(i-1)$ to $\mathcal{P}(i)$. As it was mentioned before, trajectory planning depends on the MR architecture and consumes a time t_j, for the case of our simulator shown in Fig. 1, $t_j = 0$.

Algorithm 4. Mobile robot navigation

```
1: procedure NAVIGATION(q_0, q_f, O_N, η, ϵ, M, N_p)
2:     i ← 0
3:     navigation ← True
4:     P ← BPF(q_0, q_f, O_N, η, ϵ, M, N_p)        ▷P is an array
5:     while navigation do
6:       Perform verification of the environment
   ▷Sensing
7:       if environment has changed then
8:         q_0 ← P(i)
9:         P ← BPF(q_0, q_f, O_N, η, ϵ, M, N_p)
10:        i ← 0
11:      end if
12:      i ← i + 1
13:      Perform trajectory planning to navigate from
   P(i − 1) to P(i)
14:      if i ⩾ length of P then
15:        navigation ← False
16:        Display Goal has been achieved
17:      end if
18:    end while
19: end procedure
```

An experimental simulation platform was developed to compare and evaluate the performance of the proposed BPF algorithm, this platform is integrated by five main modules: (1) The first one contains the software implementation of the BPF algorithm; i.e., the proportional gains of the APF are optimized using the BEA with $N_p = 1$ for sequential programming. (2) The second one, it actually follows from the first module, but using $N_p > 1$; i.e., the BPF runs in parallel mode. (3) The third module contains an implementation of the PBA combined with the APF to form the PBPF. (4) The fourth

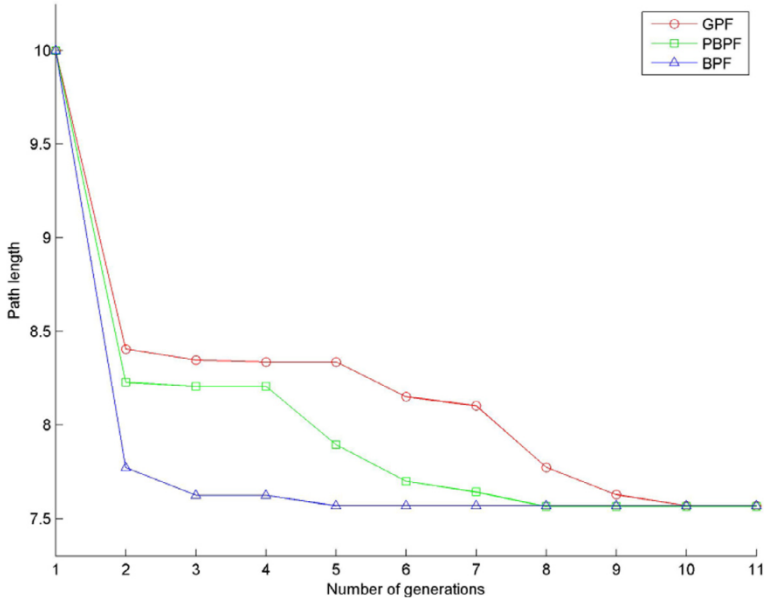
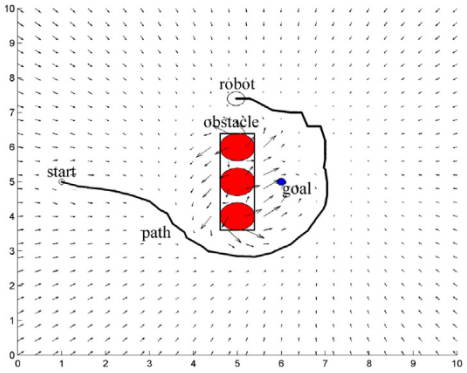


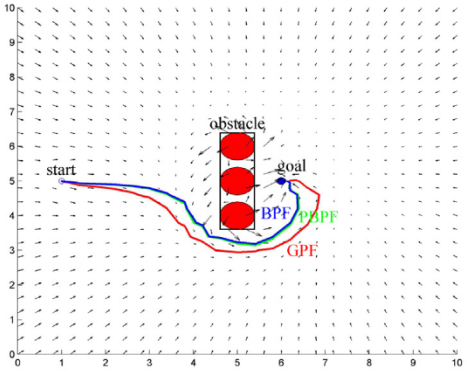
Fig. 5. Plot of convergence of the algorithms: GPF, PBPF, and BPF for the test problem 2.

Table 5
Computation time to solve the test problem 2 with BPF.

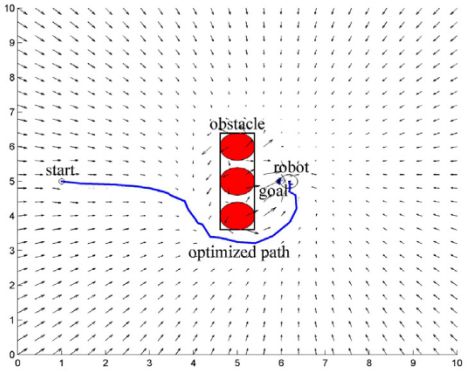
BPF	Sequential		Parallel
	Matlab	C/C++	CPU
Mean (μ)	5.33 s	2.64 s	0.87 s
Std. Dev. (σ)	1.64	0.34	0.21



(a) Goals non reachable problem



(b) Found suitable paths using different algorithms



(c) Optimized path planning

Fig. 6. Path planning for the test problem 3: Goals Non-Reachable with Obstacle Nearby.

module contains a Genetic Algorithm (GA) implementation to optimize the gains of the APF (Genetic Potential Field, GPF). (5) The last module is to simulate navigation, here the virtual MR mimics to follow the path with the capacity to sense the environment to detect any change on it, and avoiding dynamic obstacles.

The software was tailored to take advantage of the best of C/C++ and Matlab programming languages, as well as all the other capabilities that the Matlab provides. The algorithms BPF, PBPF, GPF and APF were programmed using C/C++. The graphical interface and plots were programmed using the Matlab, hence, the implementation is a mixed up of C/C++ and Matlab; the programs were integrated through the Matlab MEX-file tool. Fig. 1 shows the main screen of the experimental platform; it is assumed that the motion is achieved in the x–y plane.

4. Experiments and analysis of results

Considering that the aim of this work is to demonstrate that the BPF can effectively solve the problematic and non-solvable sceneries by other proposals based on the APF, five test problems were carefully selected. A large number of experiments were conducted to obtain statistical values to support the results. The chosen test problems are: (1) The local minima problem, which is the most common problem in classic APF method. (2) The trajectory prediction problem to show how the BPF proposal tackles the inherent limitations of the classic APF method, and how it outperforms the GPF and PBPF methods. (3) The case when the MR target (goal) is close to an obstacle; this problem in literature is known as "Goals Non-Reachable with Obstacle Nearby" (GNRON problem) (Ge & Cui (2000)). (4) Working online in unknown environments. (5) Navigating in unknown environments with dynamic obstacles; here it is necessary to perform the path planning in on-line mode, which for most current methods, local and global controllability is a big-problem, most of the times, for large sceneries, it is impossible to fulfill controllability criteria.

These first three test problems illustrate how well performs the BPF proposal beating the inherent limitations of the classic APF method, which have been documented in Koren and Borenstein (1991); Lee (2004) based on mathematical analysis, and also documented in He et al. (2011), Lee (2004), Weijun et al. (2010) and Zhang et al. (2006). The last two problems demonstrate that the BPF is capable of solving complex dynamic sceneries, which is almost impossible for the APF and the GPF in small-time.

To achieve the comparisons of the BPF proposal with respect to the APF, PBPF and the GPF methods, we have applied the next criteria:

1. The BPF proposal, the PBPF and GPF methods allow to use different values for the repulsive gains k_r's. However, because the common practice with the classic APF is to use only one value of k_r for all the obstacles to reduce problem complexity at the decision time, hence to make fair comparisons; only one k_r value for testing the algorithms was used.

Table 6
Results for test problem 3.

Factor	APF	GPF	PBFP	BPF
Gain k_a	4.00	1.00	2.88	2.51
Gain k_r	25.00	5.71	7.96	6.65
Last position x	4.97	6.14	6.17	6.17
Last position y	7.40	5.03	5.00	5.02
Distance d_a	2.609	0.175	0.174	0.174
Path length	12.439	8.406	7.584	7.515
Num of config m	2000	672	552	546
Reached goal	No	Yes	Yes	Yes
Collision	No	No	No	No

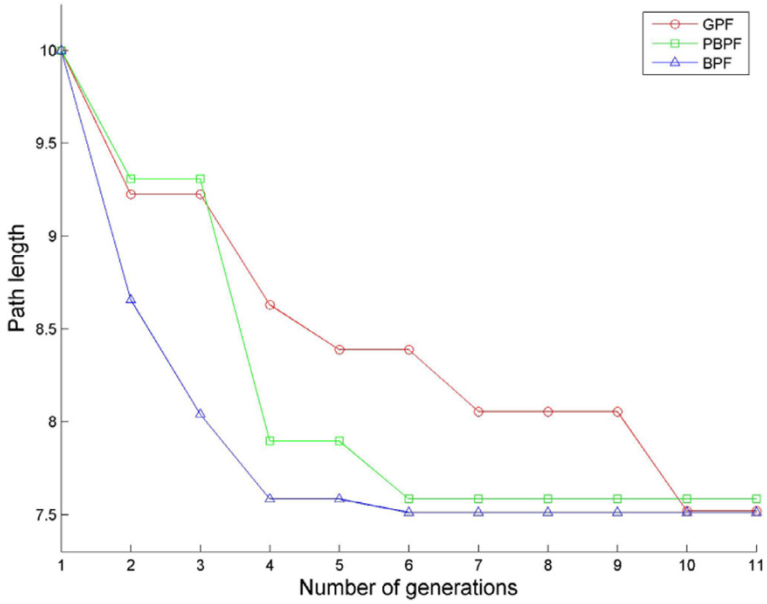


Fig. 7. Plot of convergence of the algorithms: GPF, PBPF, and BPF for the test problem 3.

Table 7
Computation time to solve the test Problem 3 with BPF.

BPF	Sequential		Parallel
	Matlab	C/C++	CPU
Mean (μ)	27.43 s	7.30 s	2.36 s
Std. Dev. (σ)	18.06	0.51	1.27

2. All the experiments were achieved on a computer with the Intel Core i7-4770, it is a fast quad core processor, and it offers Hyperthreading to handle eight threads at once. The base clock speed is 3.40 GHz but due to Turbo Boost, it can reach 3.90 GHz.
3. The parameters of the BPF, the PBPF and GPF methods are the same in all the experiments for consistency between them, we have used:
   (a) A population size of 32 chromosomes/bacteria (individuals) using 8-bit codification for each gain parameter where k_a and $k_r \in \mathbb{N}^1 \in \mathbb{N}$.
   (b) Constrained parameters, so $k_a,\ k_r \in (0, 49]$ for satisfying the criteria established in Ge and Cui (2000), the proportional gains can be any positive number and they have to be chosen properly to determine the relative intensity of potential force.
   (c) Elitist selection method with a selection rate of 0.5 (50%).
4. For the BPF, PBPF and GPF methods, we set $M = 2000$ to indicate that the maximum number of steps (robot configuration position) allowed is 2000.
5. The convergence radius (ϵ) was set to 0.175, we defined this value as an acceptable distance range from the MR position to the target point coordinates. The convergence radius has the intention to stop the robot in front of the goal and not precisely over the goal, so any position of q whose distance d_a be smaller than ϵ indicates that the robot has reached the target.

4.1. Test problem 1: local minima

Fig. 2(a) shows a typical local minima trap. It is the best-known and most referenced problematic situation in the APF (He et al., 2011; Lee, 2004; Zhang et al., 2006). Local minima can be caused by either one obstacle or combination of them, For this test, the start position is at coordinates (9.00, 8.00), the goal point at (4.00, 3.00), and there are three obstacles forming a single L-shaped obstacle, they have a radius of 0.4 units, and they are located at coordinates (6.80, 5.00), (5.5, 5.00), and (5.50, 6.30). We show experiments and results for the APF, GPF, PBPF and BPF methods. For the case of:

APF: The user has to set the gain parameters k_a and k_r using heuristic knowledge; however, to obtain a valid successful path, it is necessary to provide the adequate combination of gains, being common to give unappropriated values, falling in local minima traps. In this example the APF (Algorithm 3) iterated 2000 times, which means that the virtual MR computed 2000 different q-positions to try to reach the goal, but finally, it was trapped in a local minimum and stopped at (6.36, 5.94) without complete its mission (obtaining an optimal path).

GPF: In this case, a GA was used to optimize the gain parameters. Table 2 shows that it was possible to find an viable suboptimal path using a $k_a = 6.46$ and a $k_r = 43.35$. The path needs 466 q-positions to reach the goal, the final position of the last q-position in the path is in the limit of the convergence radius, $d_a = 0.173$ units.

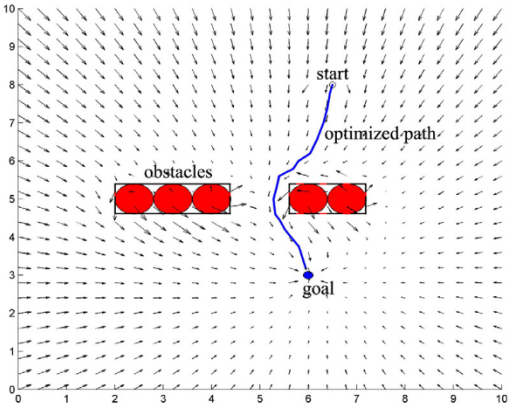
PBPF: Here, the algorithm also found a suboptimal path, the path length is smaller than the one obtained with the GPF. The path needs 438 q-positions to reach the goal. The last q-position is inside the convergence radius $d_a = 0.173$ units.

BPF: Similar to GPF and the PBPF, the BPF was capable of finding an optimal path from the practical point of view. This path is better than the other ones founded by the GPF and the PBPF. It offers the shortest distance to the goal, it only needs 391 q-configurations, and it will let the MR closer to the goal since $d_a = 0.173$ units.

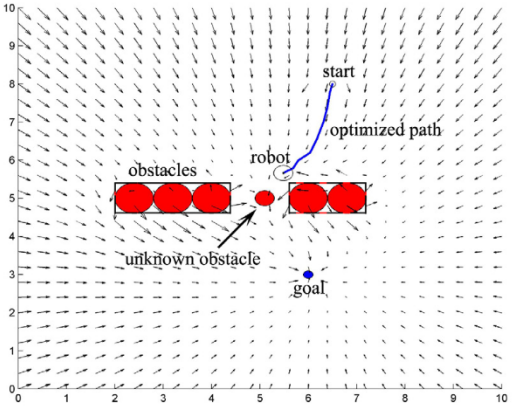
Table 2 provides a summary of the obtained results with the four methods. Fig. 2(b) shows that the GPF, PBPF and BPF methods could find an optimal path. Fig. 2(c) illustrates that the algorithms consider the physical size of the MR in spite of its architecture. In the last q-position the perimeter of the circle indicates that the MR

size contains the goal position ($d_a \leqslant \epsilon$ in Algorithm 3). The goal position is the center of the radius of convergence ϵ, note, that we are not looking for overlapping the last q-position with the goal position.

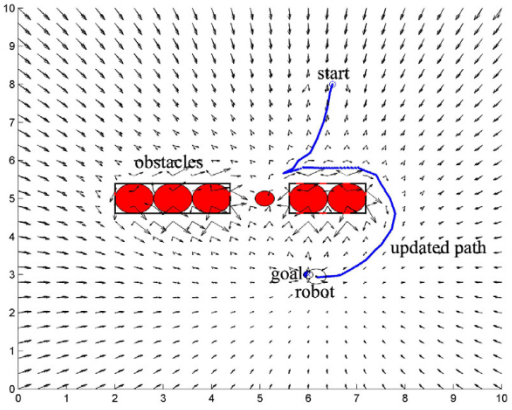
Fig. 3 helps to illustrate the convergence characteristics of the GPF, PBPF and the BPF algorithms. It reveals that the BPF is quite faster to find the optimal values leading to reduction in computation burden.



(a) Off-line path planning



(b) A unknown obstacle is detected during the navigation



(c) On-line path planning

Fig. 8. Path planning for test problem 4: On-line path planning for unknown environments.

Table 3 shows the amount of time that BPF method lasted to compute the optimal gain values (solution), hence using sequential programming ($N_p = 1$) and the algorithms programmed in C/C++, the BPF algorithm last 2.30 s to find the gain values. With the parallel version of BPF, the time that the algorithm last to compute the optimal gains is 0.74 s. For statistical significance, we run the Algorithm 30 times. As it was expected, the path planning algorithms programmed with the Matlab programming language were the slower.

4.2. Test problem 2: trajectory prediction

The trajectory prediction problem occurs when the gain parameters k_r and k_a are chosen wrong, see Fig. 4(a). This situation usually happens because heuristics based on the fact that "just because something worked in the past, it has to work again" at most times fail. In Chen et al. (2009a), Dozier, Homaifar, Bryson, and Moore (1998) and Zhang et al. (2012) some examples of this topic are illustrated. For this test problem, the start point position of the MR is at coordinate (5.00, 8.00), and the goal point is at (5.00, 2.00); moreover, there is a block of five obstacles forming a Z-shaped obstacle as it is shown in Fig. 4(b). The radius of each obstacle is 0.4 and center position at (5.00, 5.10), (4.00, 5.10), (6.00, 5.10), (4.00, 6.10), and (6.00, 4.10). Table 4 summarizes all the results of the achieve experiments, which are:

- **APF**: As it was mentioned before, choosing the right parameters is hard-work, and it is very common to use wrong gain parameters, which make impossible to find a feasible path to the goal, Fig. 4(a) illustrates this case. After 2000 step positions the algorithm stopped because the impossibility to reach the target, the last q-position was at coordinates (4.09, 0.98) which is very far from the target. In this case, this set of position will face the MR to a non-controllable situation.
- **GPF**: An optimal path was obtained using 555 q-positions. Considering the path length as a criteria, we can consider that this algorithm is as good as the PBPF and the BPF, however, consistently, this algorithm needed more q-positions than the other two.
- **PBPF**: For this test problem, this algorithm behaves in the middle of performance.
- **BPF**: Consistently, the BPF obtained a path that makes that the MR subsequently can reach the goal. It obtained the shortest paths using less q-positions. See Fig. 4(c)

Fig. 5 shows a plot of convergence to the optimal values for the three studied methods that can reach effectively the goal; i.e., the GPF, PBPF and the BPF. It is noticeable that the BPF is the faster algorithm to converge; therefore, we provide in Table 5 the computing time to solve the problems for different implementations.

Table 8
Results for the test problem 4.

Factor	GPF	PBPF	BPF
Gain k_a	15.31	14.93	12.11
Gain k_r	43.73	37.71	29.61
Last position x	6.17	6.17	6.17
Last position y	2.99	3.00	2.99
Distance d_a	0.174	0.174	0.174
Path length	5.413	5.272	5.256
Num of config m	404	397	393
Reached goal	Yes	Yes	Yes
Collision	No	No	No

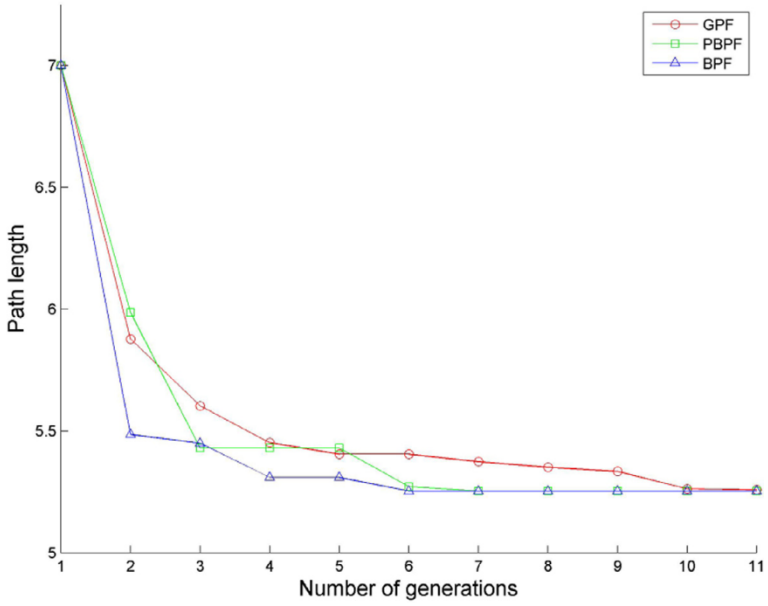


Fig. 9. Plot of convergence of the algorithms: GPF, PBPF, and BPF for the test problem 4.

Table 9
Computation time to solve the test Problem 4 with BPF.

BPF	Sequential		Parallel
	Matlab	C/C++	CPU
Mean (μ)	4.51 s	2.82 s	0.91 s
Std. Dev. (σ)	3.42	0.50	0.15

Similar to the first test problem, the best times were obtained with a C/C++ parallel implementation of the algorithm.

4.3. Test problem 3: goals non reachable

This scenery, is different from the local minima problem, in that the global minimum is being "pushed" away from where it is supposed to be (i.e., the goal point), to another position. As a result, the MR cannot converge to the correct point, which is the goal. This problem, in literature is known as "Goals Non-Reachable with Obstacle Nearby" (GNRON problem) Ge and Cui (2000), Lee (2004) and Volpe and Khosla (1990). For this test problem, the start point position of the MR is at coordinate (1.00, 5.00), and the goal point is at (6.00, 5.00); moreover, there are three obstacles with coordinate position in (5.00, 4.00), (5.00, 5.00), and (5.00, 6.00) to recreate a real-world wall, as it is illustrated in Fig. 6(a). Table 6 shows a summary of results for the four methods. For the GPF, PBPF and BPF, the results are taken from the sixth generation when the BPF converges as it is shown in Fig. 7. Some remarks about them are:

APF: In the APF, using heuristic knowledge, we choose the unlucky values $k_a = 4.00$ and $k_r = 25.00$, hence the MR gets involved in the GNRON problem. Here, the Algorithm 3 after trying 2000 ($m = 2000$) steps declines to continue, and it stops; so, the MR is non-controllable because the time T to reach the goal tends to infinity. Fig. 6(a) shows this situation.

GPF: This method can effectively reach the goal. Fig. 6(b) shows this situation.

PBPF: In this problem, the PBPF can also provide an optimal path, but it requires more q-positions than the BPF; moreover, it provides a shorter path than the GPF.

BPF: This is the best method to solve this problem, it needs few q-positions and it provides the shortest path. Fig. 6(c) illustrates how the BPF by choosing the appropriated gains can effectively avoid to fall in the GNRON problem.

The comparison of convergence characteristics of GPF, PBPF, and BPF algorithms on test Problem 3 in Fig. 7 reveals that BPF is quite faster in optimization, leading to a reduction in computation burden.

Table 7 shows the required computation time to solve this problem for different programming languages. The parallel version of BPF outperforms the sequential version by a factor of 3.09 (i.e., $\approx 209\%$), both programmed in C/C++.

4.4. Test problem 4: on-line path planning for unknown environments

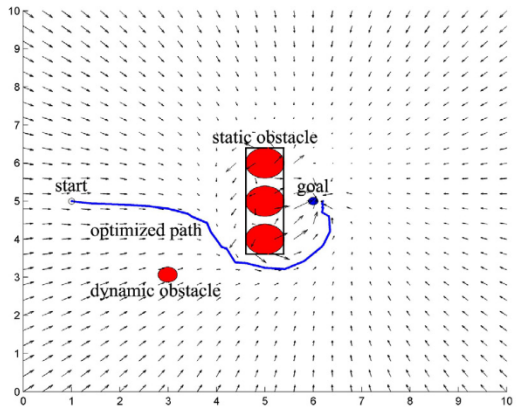
An advantage of the off-line mode is that the environment is known; however, real-world applications frequently face the MR to unknown or partially known environments, where the already planned path needs to be adjusted for reaching the goal fulfilling controllability criteria, this operating mode is referred as on-line.

Definitely, the classic APF method cannot work well in the on-line mode of path planning, where random positioning of obstacles as well as with moving obstacles (dynamic obstacles) exist, just because the APF method needs that the gain parameters are set in the "Manual" form of operation. The existence of a non-static world configuration requires the use of the "Automatic" form for updating the gain parameters' values.

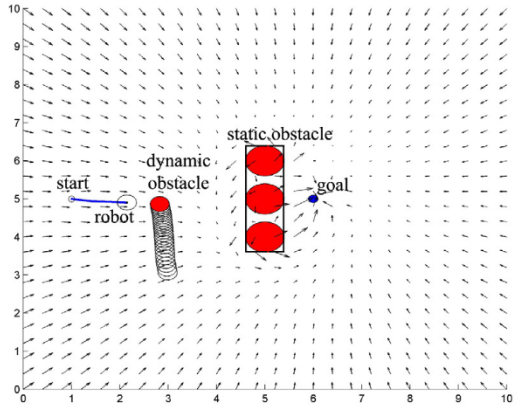
In this experiment, we are going to place an obstacle at random, in such a way that it blocks the already found optimal path, interfering with the MR trajectory. In this case, the random obstacle must remain static once it was put. At the beginning of the experiment the environment is known, the start point is at (6.50, 8.00), the goal point is at (6.00, 3.00), and there are five obstacles placed at positions: (6.00, 5.00), (4.00, 5.00), (3.20, 5.00), (2.40, 5.00), and (6.80, 5.00) forming two wall obstacles to recreate a real-world

corridor; the proposal methods were tested using the above world configuration, see Fig. 8(a). The experiments were conducted as follows:

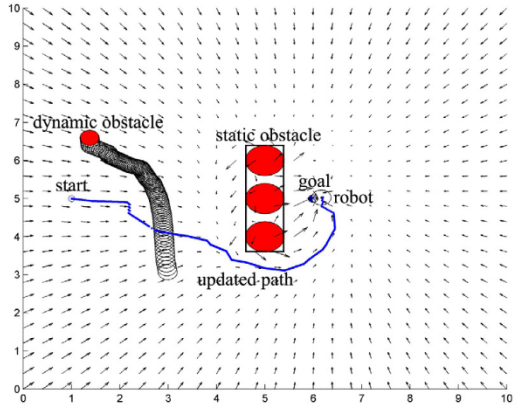
1. The path planning is achieved in the off-line mode because the environment is known, see Fig. 8(a).
2. We switch to the on-line mode, and the navigation starts. At the beginning the MR follows the path planned in off-line mode, see Fig. 8(b).



(a) Off-line path planning



(b) A unknown obstacle is detected during the navigation



(c) On-line path planning

Fig. 10. Path planning for test problem 5: On-line path planning for unknown environments with dynamic obstacles.

3. Afterwards, an obstacle is added at position (5.10, 5.00) to make a change in the environment (world configuration) for the MR, see Fig. 8(b).
4. The MR at the position (5.49, 5.66) senses the obstacle, and it calculates the obstacle position to update the world configuration.
5. The path planner of the MR now has a different world configuration; hence, it is necessary to update the path by recalculating the set of gain parameters.
6. The MR follows the new path to reach the goal, see Fig. 8(c).

Table 8 shows a summary of the obtained results from the sixth generation of Fig. 9. Some important remarks about the feasible methods are given below:

GPF: This method can find optimal paths avoiding collisions. The path length is comparatively as good as with the PBPF and BPF. It requires more q-positions than the other methods. It is the slower method, as it can be seen in Fig. 9.

PBPF: Almost same comments than in the GPF, but it will require less q-positions than the GPF. It practically is as faster as the BPF.

BPF: This method provided the best results, it needs less q-positions than the other two methods. Consistently with the other experiments, it is still being the faster. Table 9 shows computation times for the sequential and parallel implementation, note the BPF in parallel outperforms the sequential implementation by a factor of 3.1 (i.e., $\approx 210\%$) in on-line mode, avoiding obstacles.

4.5. Test problem 5: on-line path planning for unknown environments with dynamic obstacles

The interaction with the real world requires the ability to respond and take decisions over the changes in the environment. In Goerzen et al. (2010), Qixing, Yanwen, and Jingliang (2006), Vadakkepat et al. (2001) and Vadakkepat et al. (2000) are presented some implementations with a changing environment (world configuration). In this test problem, differently to the test problem 4, where the random obstacle remained static once it appeared in the environment; here, once the optimal path was planned, the obstacle will appear in the environment blocking the MR trajectory, but the obstacle will not remain static, it will be moving with a predefined trajectory. This problem is more complex because it faces the path planning algorithms to make their optimization work over a noisy surface; hence, at every iteration, the searching zone might be different. The experiment was conducted as follows:

1. The start position of the MR is at coordinate (1.00, 5.00), and the goal position is at (6.00, 5.00), there are three obstacles with coordinate position in (5.00, 4.00), (5.00, 5.00), and (5.00, 6.00) to recreate a real-world wall.
2. The dynamic obstacle in the position (2.99, 3.07) is unknown for the MR, as a consequence of this condition, the dynamic obstacle is not considered for the initial path planning as is shown in Fig. 10(a), at this point the path planning is considered off-line.
3. Once the optimal path was obtained (off-line mode), the MR starts to navigate (on-line mode).
4. When the MR is at the position (2.15, 4.91), it detects a "new" obstacle, which was not considered at the off-line path planning stage; see Fig. 10(b).
5. The new obstacle is moving, and eventually, it will obstruct the path that the MR is following; at this point, the BPF proposal

starts to recalculate a set of parameters to avoid colliding the mobile by updating the optimal path to reach the goal (on-line mode).

6. The Fig. 10(c) shows the updated path for avoiding the moving obstacle.

Table 10 shows the results taken from the fifth generation of Fig. 11, the BPF has converged while the GPF and PBPF will do later. It is worthwhile to note that eventually the results will tend to be very similarly due to convergence of the algorithms. However, the strength of the BPF is that the BEA in this application, usually need fewer generations to provide optimal values; therefore, it is the fastest method leading to a reduction in computation burden.

Some important remarks about the feasible methods are given below:

GPF: This method was capable to obtain an optimal path in on-line mode. The path to reach the goal is longer than the obtained by the PBPF and BPF, but is comprehensively good.

PBPF: The PBPF performs well working on-line and it requires less q-positions than the GPF.

BPF: BPF approach provide the best results, it finds a shorter path to reach the goal in less q-positions and consistently to previous experiments, the parallel version of the BPF outperforms the sequential version by a factor of 3.04 ($\approx 204\%$) for the on-line mode as it is shown in Table 11.

Table 10
Results for test Problem 5.

Factor	GPF	PBPF	BPF
Gain k_a	2.32	2.51	3.07
Gain k_r	9.66	8.53	8.72
Last position x	6.17	6.17	6.18
Last position y	4.96	5.00	5.00
Distance d_a	0.175	0.175	0.175
Path length	6.965	6.775	6.482
Num of config m	476	426	417
Reached goal	Yes	Yes	Yes
Collision	No	No	No

Finally, in Table 12 a summary of the execution times to reach the optimal value of every test problem is provided. In this table, we let run the algorithms until they found a feasible path and they fulfilled the optimization criteria; i.e., finding a distance $d_a \leqslant \epsilon$. Hence, the BPF last fewer generations than the PBPF and GPF, although there were only two or four generations less, it is significant since the fitness function (Algorithm 3) requires to perform many operations making its calculus computationally expensive. The average time for each implementation (sequential or parallel) of each algorithm was computed, with the aim of calculating the average speedup between the BPF and the PBPF, as well as with the GPF, in all the cases, the speedup was significantly good. For example, in the BPF vs. the GPF the speedup is 2.16 for the sequential implementation, and 2.05 for the parallel implementation, favoring the BPF.

Note, that the speedup of the BPF running in parallel mode using the system described in Section 4, vs. the sequential mode, is 3.08. This is important since it indicates that the BPF is an

Table 11
Computation time to solve the test Problem 5 with BPF.

BPF	Sequential		Parallel
	Matlab	C/C++	CPU
Mean (μ)	14.50 s	8.50 s	2.79 s
Std. Dev. (σ)	3.65	0.29	0.15

Table 12
Algorithm convergence to the optimal value. Time comparison for the different test problems. The time was measured in seconds. Seq. = Sequential, Par. = Parallel.

Test prob.	GPF		PBPF		BPF	
Num.	Seq.	Par.	Seq.	Par.	Seq.	Par.
1	4.11	1.33	3.53	1.08	**2.30**	**0.74**
2	8.53	2.54	3.48	1.13	**2.64**	**0.87**
3	14.91	4.72	11.71	3.79	**7.30**	**2.36**
4	6.81	1.82	3.07	1.00	**2.82**	**0.91**
5	16.88	5.26	15.85	5.15	**8.50**	**2.79**
Avg. time	10.19	3.13	7.53	2.43	**4.71**	**1.53**
Average speedup	2.16	2.05	1.60	1.59		

For each test problem, for the sequential and parallel implementation, the best obtained values were written in bold.

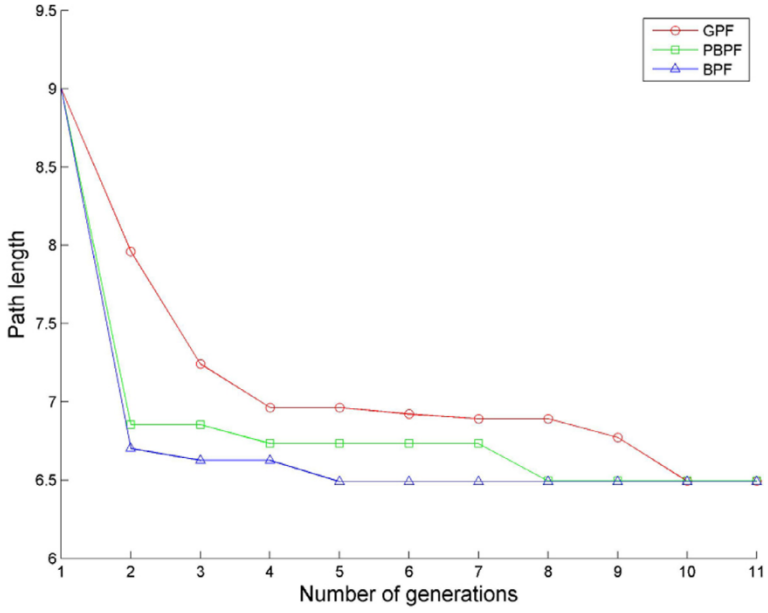


Fig. 11. Plot of convergence of the algorithms: GPF, PBPF, and BPF for test problem 5.

scalable method that can take advantage of new technologies, such as the use of graphics processing units (GPUs).

5. Conclusions and future work

In this paper, we proposed a novel method for path planning called Bacterial Potential Field (BPF), which ensures to find optimal paths in complex real-world sceneries with static and dynamic obstacles. The method is based on the APF, and it uses a BEA to find the shortest path (optimal path) according to the designer criteria; hence, it is not necessary finding the global optimal path. The experimental platform is based on the BPF, and it uses a generic but realistic MR model which considers the MR physical size and orientation in the plane.

This work has been focused to path planning since it is the computational challenge; however, Algorithm 4 provides an MR navigation method that can be used for offline, online and for a virtual MR. The algorithm considers following a path and trajectory planning if it is necessary. Trajectory planning depends on the MR hardware architecture, hence it is necessary to know the kind of MR, motor speeds, etc., to achieve such planning.

For non-critical mission, we say that the MR is controllable if it can successfully reach the goal from the start position, in a reasonable time T. On the other hand, in critical missions, the MR must reach the goal before the time T has elapsed. This time depends directly on the number of MR configurations in the path, being very important the consumed time to perform the trajectory planning to go from one point to the next point in the path. Unknown environments with dynamic obstacles in the path of an MR confront the path planning algorithm to very complex computational problems that demand high-performance computing to calculate online new paths and then calculate the trajectory, fulfilling the small-time local controllability requirements. In general terms, we have demonstrated that the BPF outperforms the GPF and PBPF methods by reducing the computational time to find the optimal path at least by a factor of 1.59, obtaining the shortest distances using fewer configurations. These results definitely impact positively the ability of the BPF path planning method to satisfy local and global controllability.

The BPF method can be used in sequential mode for low cost onboard computers, or in parallel mode ($N_p > 1$) to take advantage of novel computing architectures that provides multicore CPUs in the same board, even the use of GPUs at low cost. The BPF is a high-performance scalable method that can help to solve the problem of local and global controllability because it is capable of providing solutions at a very low computational time.

In addition to the MR navigation in dynamic complex environments, the path planning with the BPF method can be used in different applications, for example, in robotic surgery, video game artificial intelligence, architectural design, and optimizing paths in computer-assisted manufacturing applications (CAM).

There are several directions for the future work, for example: (1) The work can be expanded to very complex sceneries in 3D (massive data environment) containing multiple static and dynamic obstacles of many sizes and forms. (2) The evaluation of the BPF in parallel mode demonstrated that it is a high-performance highly scalable method, the approach can be implemented using new technologies such as the Compute Unified Device Architecture (CUDA) that takes advantage of the GPUs, providing the algorithm with thousands of programming threads; this is enough programming power that enables the MR to navigate in practically any complex real-world scenery. (3) The BPF method can be extended and applied to the challenging research area of driverless cars, better known as autonomous cars. The BPF can be part or can assist the navigation system in an automated vehicle capable of fulfilling the main transportation capabilities of a traditional car. (4) The BPF method can be extended to MR formations, where the main activity of the BPF method can be planning and coordinating the motion of multiple MRs, an application for a network of robots can be a reconnaissance mission, where the number of networked robots would permit the site to be mapped in a much shorter time span, than using only a single robot.

Acknowledgments

We thank to Instituto Politécnico Nacional (IPN), to the Comisión de Fomento y Apoyo Académico del IPN (COFAA), and to the Mexican National Council of Science and Technology (CONACYT) for supporting our research activities.

References

Barnes, L., Fields, M., & Valavanis, K. (2009). Swarm formation control utilizing elliptical surfaces and limiting functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 39*(6), 1434–1445.

Baudry, B., Fleurey, F., Jézéquel, J.-M., & Traon, Y. L. (2005). From genetic to bacteriological algorithms for mutation-based testing. *Wiley InterScience – Software Testing, Verification and Reliability*, 73–96.

Berger, J., Jabeur, K., Boukhtouta, A., Guitouni, A., & Ghanmi, A. (2010). A hybrid genetic algorithm for rescue path planning in uncertain adversarial environment. *Evolutionary computation*. IEEE [pp. 1–8].

Bhattacharyya, A., Singla, E., & Dasgputa, B., 2007. Robot path planning using silhouette method. In *13th National conference on mechanisms and machines (NaCoMM07)*. Bangalore, India.

Botzheim, J., Gál, L., & Kóczy, L. T. (2009). Fuzzy rule base model identification by bacterial memetic algorithms. In E. Rakus-Andersson, R. R. Yager, N. Ichalkaranje, L. C. Jain, E. Rakus-Andersson, & R. R. Yager, et al. (Eds.), *Recent advances in decision making. Studies in computational intelligence* (pp. 21–43). Springer.

Botzheim, J., & Kóczy, L. T. (2004). *Model identification by bacterial optimization*. Budapest, Hungary: National Scientific Research Fund.

Botzheim, J., Toda, Y., & Kubota, N. (2010). *Path planning in probabilistic environment by bacterial memetic algorithm*. Szchenyi Istvn University and Tokio Metropolitan University.

Botzheim, J., Toda, Y., & Kubota, N. (2011). *Path planning for mobile robots by bacterial memetic algorithm*. IEEE.

Cantú-Paz, E. (2001). *Efficient and accurate parallel genetic algorithms*. Norwell, Massachusetts, USA: Kluwer Academic Publishers.

Chen, F., Di, P., Huang, J., Sasaki, H., & Fukuda, T. (2009a). *Evolutionary artificial potential field method based manipulator path planning for safe robotic assembly*. IEEE [pp. 92–97].

Chen, K. Y., Lindsay, P. A., Robinson, P. J., & Abbass, H. A. (2009b). A hierarchical conflict resolution method for multi-agent path planning. *Evolutionary computation*. IEEE [pp. 1169–1176].

Dozier, G., Homaifar, A., Bryson, S., & Moore, L. (1998). *Artificial potential field based robot navigation, dynamic constrained optimization, and simple genetic hill-climbing*. IEEE [pp. 189–194].

Eppstein, D., (1996). Geometry in action: Voronoi diagrams. <http://www.ics.uci.edu/eppstein/gina/voronoi.html>. accessed January 20, 2015.

Fallah, M., Akbari, F., & Javan, A. (2010). *Implementation of common genetic and bacteriological algorithms in optimizing testing data in mutation testing*. IEEE.

Fogel, D., Bäck, T., Hammel, U., & Schwefel, H. P. (1998). *An introduction to evolutionary computation*. New York: IEEE Press, New York.

Ge, S. S., & Cui, Y. J. (2000). New potential functions for mobile robot path planning. *IEEE – Transactions on Robotics and Automation*, 615–620.

Goerzen, C., Kong, Z., & Mettler, B. (2010). A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Springer – Journal of Intelligent & Robotic Systems*, 65–100.

He, L. G., Gao, W. H., & Nan, L. Y. (2011). *A route planning method based on improved artificial potential field algorithm*. IEEE [pp. 550–554].

Hocaoglu, C., & Sanderson, A. (2001). Planning multiple paths with evolutionary speciation. *Evolutionary computation*. IEEE [pp. 169–191].

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotic Research*, 90–98.

Kim, M. H., Heo, J. H., Wei, Y., & Lee, M. C. (2011). A path planning algorithm using artificial potential field based on probability map. *Eighth international conference on ubiquitous robots and ambient intelligence*. IEEE [pp. 41–43].

Koren, Y., & Borenstein, J. (1991). Potential field methods and their inherent limitations for mobile robot navigation. *Conference on robotics and automation*. IEEE [pp. 1398–1404].

Lee, L. F. (2004). Decentralized motion planning within an artificial potential framework (apf) for cooperative payload transport by multi-robot collectives (Master's thesis). New York, USA: State University of New York.

Li, S., Ding, M., Cai, C., & Jiang, L. (2010). Efficient path planning method based on genetic algorithm combining path network. *Genetic and evolutionary computing*. IEEE [pp. 194–197].

Masehian, E., & Sedighizaddeh, D. (2007). Classic and heuristic approaches in robot motion planning – A chronological review. *Engineering and technology* (Vol. 23, pp. 101–106). Germany: World Academy of Science.

Mitchel, M. (2001). An introduction to genetic algorithms. Bradford, Massachusetts, USA.

Montiel, O., Sepúlveda, R., Castillo, O., & Melin, P. (2013). Ant colony test center for planning autonomous mobile robot navigation. *Computer Applications in Engineering Education, 21*(2), 214–229.

Nawa, N. E., & Furuhashi, T. (1999). Fuzzy system parameters discovery by bacterial evolutionary algorithm. *IEEE Transactions on Fuzzy Systems, 7*(5), 608–616.

Nawa, N. E., Hashiyama, T., Furuhashi, T., & Uchikawa, Y. (1997). A study on fuzzy rules discovery using pseudo-bacterial genetic algorithm with adaptive operator. *Evolutionary computation*. IEEE [pp. 589–593].

Park, M. G., & Lee, M. C. (2003). Artificial potential field based path planning for mobile robots using a virtual obstacle concept. *International conference on advanced intelligent mechatronics*. IEEE/ASME [pp. 735–740].

Qixing, C., Yanwen, H., & Jingliang, Z. (2006). An evolutionary artificial potential field algorithm for dynamic path planning of mobile robot. *International conference on intelligent robots and systems*. IEEE/RSJ [pp. 3331–3336].

Sedighi, K. H., Ashenayi, K., Manikas, T. W., Wainwright, R. L., & Tai, H. M. (2004). Autonomous local path planning for a mobile robot using a genetic algorithm. *Genetic and evolutionary computing*. IEEE [pp. 1338–1345].

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robot* (2nd ed.). London, England: The MIT Press.

Vadakkepat, P., Lee, T. H., & Xin, L. (2001). *Application of evolutionary artificial potential field in robot soccer system*. IEEE [pp. 2781–2785].

Vadakkepat, P., Tan, K. C., & Wang, M. L. (2000). Evolutionary artificial potential fields and their application in real time robot path planning. *Congress on evolutionary computation*. IEEE [pp. 256–263].

Volpe, R., & Khosla, P. (1990). *Manipulator control with superquadric artificial potential functions: Theory and experiments*. *Transactions on systems, man, and cybernetics* (20). IEEE [6].

Weijun, S., Rui, M., & Chongchong, Y. (2010). A study on soccer robot path planning with fuzzy artificial potential field. *International conference on computing, control and industrial engineering*. IEEE [pp. 386–390].

Zhang, Q., Chen, D., & Chen, T. (2012). An obstacle avoidance method of soccer robot based on evolutionary artificial potential field. *International conference on future energy, environment, and materials* [pp. 1792–1798].

Zhang, B., Chen, W., & Fei, M. (2006). An optimized method for path planning based on artificial potential field. *Proceedings of the sixth international conference on intelligent systems design and applications*. IEEE.

Zhang, K., Collins, E. G., & Barbu, A. (2013). An efficient stochastic clustering auction for heterogeneous robotic collaborative teams. *Journal of Intelligent & Robotic Systems, 72*(3-4), 541–558.