Applied Soft Computing Journal 77 (2019) 236–251

Contents lists available at ScienceDirect

Applied Soft Computing Journal

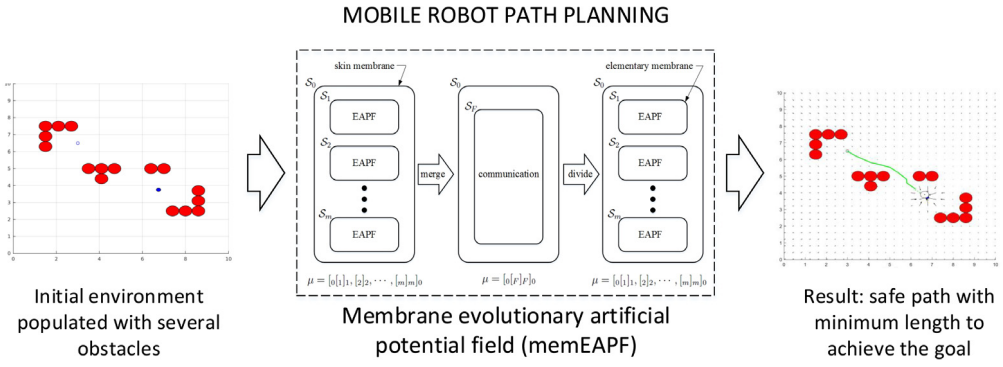
journal homepage: www.elsevier.com/locate/asoc

Mobile robot path planning using membrane evolutionary artificial potential field

Ulises Orozco-Rosas, Oscar Montiel *, Roberto Sepúlveda

Instituto Politécnico Nacional - CITEDI, Av. Instituto Politécnico Nacional 1310, Nueva Tijuana, C.P. 22435, Tijuana, Baja California, México

GRAPHICAL ABSTRACT



HIGHLIGHTS

- A membrane evolutionary artificial potential field method is proposed.
- The method was developed for mobile robot path planning.
- It can take advantage of multiprocessors systems obtaining solutions in less time.
- Path planning considering the length, safety, and smoothness in complex environments.

ARTICLE INFO

Article history:
Received 18 October 2016
Received in revised form 25 January 2019
Accepted 26 January 2019
Available online 31 January 2019

Keywords:
Path planning
Membrane computing
Membrane-inspired evolutionary algorithm
Evolutionary computation
Mobile robots

ABSTRACT

In this paper, a membrane evolutionary artificial potential field (memEAPF) approach for solving the mobile robot path planning problem is proposed, which combines membrane computing with a genetic algorithm (membrane-inspired evolutionary algorithm with one-level membrane structure) and the artificial potential field method to find the parameters to generate a feasible and safe path. The memEAPF proposal consists of delimited compartments where multisets of parameters evolve according to rules of biochemical inspiration to minimize the path length. The proposed approach is compared with artificial potential field based path planning methods concerning to their planning performance on a set of twelve benchmark test environments, and it exhibits a better performance regarding path length. Experiments to demonstrate the statistical significance of the improvements achieved by the proposed approach in static and dynamic environments are shown. Moreover, the implementation results using parallel architectures proved the effectiveness and practicality of the proposal to obtain solutions in considerably less time.

1. Introduction

In motion planning, the main problem is to calculate the path that allows a robot to move from a starting point to a goal point

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

https://doi.org/10.1016/j.asoc.2019.01.036
1568-4946/

of the environment while avoiding obstacles. This well-known problem is addressed in literature as *path planning* problem [1,2], and entails high computational complexity since it is an NP-hard problem [1,3,4]. Finding feasible solutions in critical applications of autonomous mobile robots in real-life requires solving path planning problems efficiently.

A goal in mobile robotics is to enable the robot to navigate successfully through the environment, hence the mobile robot (MR) needs to be equipped at least with sensors, the on-board computer, and the locomotion system [5]. Modern autonomous MRs are equipped with high-performance on-board computers to achieve multiple tasks with high processing demands, such as sensing, learning, reasoning, path planning in complex static and dynamic environments, and motion. For example, the development of MRs with moral – to take care our children crossing the street – is a present goal, and this requires path planning algorithms that allow implementing real-time schedules to achieve a safe, smooth and stable control of the MR [6].

This novel work contributes to state of the art with a high-performance soft computing methodology focused on solving real-life motion planning problems for MRs. This original proposal was evaluated using several scenarios and contrasted against up to date approaches and reported results, in the same research line referent to potential field methods – i.e., methods based on the artificial potential field (APF) – outperforming them. To the best of our knowledge, there is no work in the literature which considers the use of membrane-inspired evolutionary algorithms (MIEA) [7,8] with potential field approach to solve the MR path planning problem.

We named the proposal memEAPF (*membrane evolutionary artificial potential field*), because it is a hybrid approach based on an appropriate combination of three methodologies: membrane computing (cell-like P systems) [9,10], evolutionary computation (in specific a genetic algorithm) [11] and the APF method [12], used for solving the path planning problem. In specific a MIEA with one-level membrane structure (OLMS) [7,13,14] is employed to evolve a set of parameters required to generate the solution (path) that will drive the MR to its goal. The memEAPF can work efficiently in static and dynamic environments. It takes advantage of novel computer architectures to speed up the computation, providing high-quality solutions in considerable less time, in comparison of similar software versions whose algorithms do not exploit such advantages. We present an extensive simulation study to evaluate and demonstrate the performance of the proposal.

In this work, the memEAPF approach is proposed to solve MR path planning problems. The main contributions of this paper can be summarized as follows:

1. The memEAPF is a novel proposal based upon membrane evolutionary algorithm with one-level membrane structure that can find feasible paths, outperforming motion planning proposals based on the APF method.
2. The memEAPF is capable to achieve results (feasible paths) considering distance (minimum path length), safety (avoiding collisions) and smoothness (due to the use of the APF).
3. The memEAPF takes advantage of recent computer technology by making practical its implementation in parallel architectures to speed up the computation time obtaining results in considerable less time.
4. Extensive experiments are carried out by considering various complex static and dynamic environments to verify the effectiveness and practicality of memEAPF to perform path planning in off-line and on-line mode.

The field of path planning is a vast area of study, and there have been published hundreds of papers with proposals to solve path planning problems. Therefore, in Section 2 we present a broad classification of diverse techniques, we mentioned some related work that cover classical and approximation algorithms. The remainder of the paper is organized as follows. In Section 3, the path planning problem is stated. In Section 4, the description of the memEAPF approach is presented. In Section 5, the conducted experiments supported with a complete comparative study of twelve test environments as well as the experimental results are provided. Finally, conclusions are drawn in Section 6.

2. Related work

Robot motion planning can be broadly classified into two main approaches: classical, and approximation algorithms.

Classical approaches include: *roadmap*, *cell decomposition*, *mathematical programming*, and *potential field method* [15].

- *Roadmap* is a computational geometry-based approach to path planning [16] which mainly is subdivided in: *visibility graph* [17], *Voronoi diagram* [16], *subgoal network* [18], and *silhouette approach* [19].
- In *cell decomposition* the idea is to decompose the C-space into a set of simple cells and then compute the adjacency among cells. In [20] the cell decomposition is used to construct a connectivity graph with observation cells; the graph is pruned and transformed into a decision tree from which an optimal sensing strategy can be computed to perform the path planning.
- 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 the goal point [15].
- *Potential field method* was introduced by Khatib [12] for a configured space. The main idea of the method is to establish an attractive potential field around the goal position, as well as to establish a repulsive potential field force around obstacles, by this idea the *potential field method* uses attractive and repulsive forces to guide a robot to its goal while keeping it away from obstacles. The MR is considered as a particle under the influence of a potential field, and the local variation reflects the free space structure.

Approximation algorithms are methods based on heuristics or meta-heuristics, they can yield good solutions, but not necessarily the optimum; this category mainly includes: *probabilistic methods*, *single/multiple objective bio-inspired algorithms*, and *fuzzy logic* among others.

- *Probabilistic methods* are mainly subdivided in: *probabilistic roadmaps* [21], *rapidly-exploring random trees* [22], *level set* [23], and *linguistic geometry* [24].
- *Bio-inspired algorithms* include evolution based algorithms such as *genetic algorithms* [25], *evolutionary strategy*, *differential evolution*, swarm-based algorithms, and algorithms inspired by ecology systems such as the air and the water [26]. Examples of swarm-based intelligence paradigms employed in MR path planning are *ant colony optimization* [27], *particle swarm optimization* [28], *artificial bee colony* [29] and *bee colony optimization* [30,31].
- Multi-objective proposals to solve the MR path planning have been addressed with *multi-objective evolutionary algorithms* [32] such as *non-dominated sorting genetic algorithm II* [33], other proposals include *variable neighborhood search* [5], and *membrane-inspired algorithms* [34].

- *Stigmergy* [35], *fuzzy logic* [36], *wavelets* [37], *tabu search* [38] and *simulated annealing* [39] are other algorithms that have served as core methodology to develop path planning proposals. In control for trajectory tracking in MR navigation, type-1 and type-2 fuzzy logic systems have been developed [40,41] as well as membrane based controllers [42].

All the mentioned methods have their strengths and drawbacks; they are deeply connected to one another, and in many applications, some of them were combined to derive the desired MR motion planner in the most effective and most efficient manner [15]. In the literature, it has been stated that classical approaches suffer from numerous disadvantages, such as trapping in local minima as well as high time complexity in large search spaces. With the intention to overcome such drawbacks, many proposals have emerged that combine one or several approximation algorithms.

In that sense, the memEAPF is a hybrid algorithm that synergistically combines membrane computing with a genetic algorithm and the APF method to solve path planning problems. As a branch of natural computing, membrane computing, initiated by Gheorghe Păun in 1998 [9], aims to abstract distributed and parallel computing models, also called P systems or membrane systems, from the compartmentalized structure and interactions of living cells [34]. The obtained computing models (P systems) are distributed parallel devices that evolve through rules and process the multisets of objects into the compartments that are hierarchically defined [10]. Regarding the genetic algorithm employed in the memEAPF, the main reason is due to its usefulness and efficiency in large and complex search spaces, it can rapidly locate good solutions in difficult search spaces, and it is relatively easy to implement [26].

The APF method is used in path planning because of its simplicity, mathematical elegance and effectiveness in providing smooth and safe planning; unfortunately, it has limitations in many real-world applications where the environment is dynamic, and therefore APF method becomes impractical, producing inefficient path planning [43]. With the aim to overcome these limitations, the memEAPF approach through the membrane-inspired evolutionary algorithm optimizes the parameters required for the APF method to generate a feasible and safe path in static and dynamic environments. The memEAPF can work with one or several processors, which is important because to date there are many MRs with a single processor on-board computer system. On the other hand, new MRs have multiprocessor on-board computer systems.

3. Problem formulation

Path planning is a problem that requires finding a continuous path between the MR current state (start position) and goal states for a system, subject to a variety of constraints [44]. Under this broad definition, in this paper, we have formulated the problem in a simplified form, described as follows.

In Fig. 1(a), the MR environment Q is conceived as a two-dimensional map which includes a set of obstacles $O_j, j = 1, 2, \ldots, n, n \in \mathbb{N}$, where n is the number of the obstacles in the environment Q. Fig. 1(b) shows the instantaneous position, where the physical space occupancy of the MR and orientation are represented by q. The x and y coordinates provide the position, the radius r determines the circular occupancy area of the MR, and the orientation θ is the angular difference between the global and local reference frames [45]. Therefore, one configuration of the MR is given by $q(x, y, r, \theta)$, or equivalently by $q(c, r, \theta)$, where $c = (x, y)$ is the center of the robot. The path planning goal is to find a feasible sequence of configurations Q_G that can drive the MR from a start position q_0 to a goal position q_f in the x–y plane [46].

Particularly, this work aims to demonstrate how the novel memEAPF proposal based upon membrane-inspired evolutionary algorithms with one-level membrane structure can successfully find Q_G, outperforming other motion planning proposals based on the APF methodology by providing better solutions concerning path length.

4. memEAPF for path planning

In this section, the proposed memEAPF for path planning is presented. We provide and explain the pseudocode of the algorithms that constitute the memEAPF; as well as we give the pseudocode that allows the implementation of a simulation platform to test the memEAPF. All the experiments shown in this paper were achieved using this simulation platform.

4.1. memEAPF approach

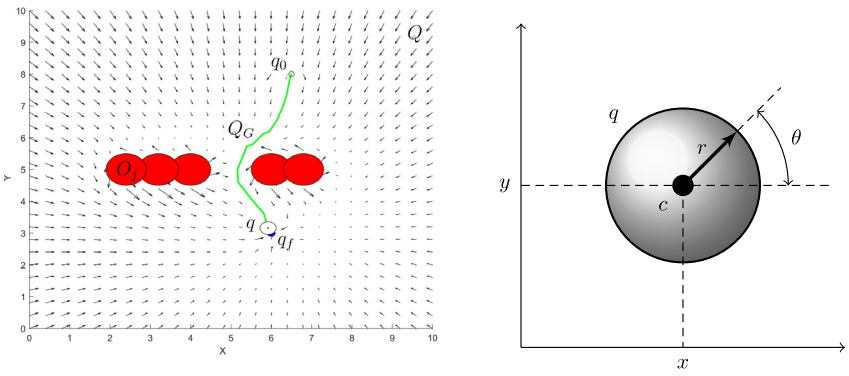
P systems employ various features to specify the structure and functionality of the living cells. In general, P systems are membrane structures with objects into their membranes, which have specific evolution rules like transformation and communication to merge and divide membranes [10]. There are three main types of P systems: cell-like P systems which contain one-membrane cell, tissue-like P systems that consist of several one-membrane cells in a common environment, and neural-like P systems that consider neurons as their cell [47]. In this work, we have selected a cell-like P system as a core component of the memEAPF approach due to its simple and easy realization in parallel. Cell-like P systems present a hierarchical structure of membranes, type of rules (e.g., transformation and communication), and intrinsic parallelism; strengths that are very effective from a computational point of view and attractive for modeling complex problems [7,13].

The memEAPF for path planning is a framework that consists of a cell-like P system, in this case, an MIEA that evolves a set of parameters $[k_a, k_r, \eta]$ required for the APF. These parameters are the attractive proportional gain k_a, the repulsive proportional gain k_r, and η which is the MR's step size. The memEAPF employs a dynamic structure with active membranes, in specific one-level membrane structure with rules, such as membrane merger and division [7], see Fig. 2. The membrane merger is helpful to enhance the information communication among individuals (set of parameters), and the membrane division is beneficial to improve the search capability [13,48].

Fig. 2 shows the membrane structure μ for the memEAPF. It is composed of a skin membrane $\mathcal{S}_0$ and a set of elementary membranes $\mathcal{S}_i$, $1 \leq i \leq m$, $m \in \mathbb{N}$, where m is the number of elementary membranes that are embedded in the skin membrane. All membranes are labeled; the number of membranes is the *degree* of the membrane structure μ, while the height of the tree associated in the usual way with the structure is the *depth* [49]. For the memEAPF, we have a membrane structure of degree $m + 1$ and depth 2.

The membrane structure μ for the memEAPF consists of delimited compartments where multisets of objects are evolved. In membrane computing, the multisets are composed of objects that can be of various types, not only characterized by letters from a given alphabet, as in the primary class of P systems. For instance, the objects can be described by strings, or even more by complex data structures (arrays) [50]. In the memEAPF, each elementary membrane $\mathcal{S}_i$ contains arrays of multisets of objects consisting of sets of parameters (proportional gains and step size), and evolution rules composed of merge, communication and divide stages.

The computational process outlined by Fig. 2 consists of several steps: First, each elementary membrane $\mathcal{S}_i$, composed of an



(a) MR environment (b) MR instantaneous position

Fig. 1. Path planning problem formulation.

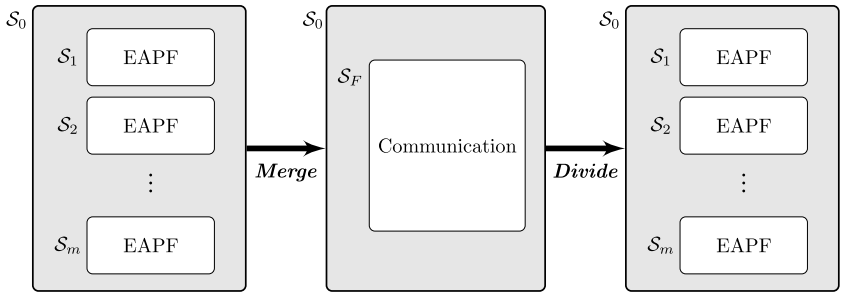


Fig. 2. Membrane structure for the memEAPF approach.

evolutionary artificial potential field (EAPF) evolves the individuals. Each individual is codified with a set of parameters. The purpose of this first stage is to find the best individual in each elementary membrane $\mathcal{S}_i$. Second, all the elementary membranes $\mathcal{S}_i$, merge into one membrane $\mathcal{S}_F$, containing all the individuals, this merge process is illustrated in Fig. 2. Communication rules are applied, which first separate the best individual of each elementary membrane with the purpose of finding the global best individual $[k_a, k_r, \eta]_{best}$, and further send out into the skin membrane a copy of this global best individual to preserve the current global best solution. The communication continues in the merged membrane $\mathcal{S}_F$ to exchange information among the elementary membranes $\mathcal{S}_i$ that will be formed in the next step. During the merge process, each subpopulation is maintained, and the worst individuals (a portion of the subpopulation) will be replaced by a copy of the best individuals to improve the subpopulation in each elementary membrane. Third, the process is repeated over and over again to refine the sets of parameters to be able to perform an optimal or close-to-optimal path planning.

4.1.1. memEAPF algorithm

Algorithm 1 shows the memEAPF pseudocode for MR path planning. The algorithm uses three input parameters: the MR start and goal positions q_0 and q_f respectively, as well as the environment layout *map*. The purpose of the Algorithm 1 is to obtain a sequence of objective points Q_G (path) that the MR must attain to reach the goal. Hence, the memEAPF achieves the task of path planning generation, with the characteristic that it provides an optimal or nearly optimal reachable set of configurations Q_G if it exists.

The backbone of the memEAPF is the use of MIEA that provides the path planner with better capabilities to find the global best set of parameters $[k_a, k_r, \eta]_{best}$ for the APF. This characteristic is critical since it allows the MR to navigate without being trapped in local minimum, making the memEAPF suitable to work in static and dynamic environments, which is crucial in real-world applications. In Algorithm 1, a function handler *FitFunc* is used to facilitate the pseudocode writing, and it is equivalent to write the APF fitness function (Algorithm 3), including all its parameters. At the beginning, the population $\mathcal{P}(t)$ is randomly generated. $\mathcal{P}(t)$ is composed of m subpopulations $P_i(t)$, forming multisets with unique individuals $p_{(i,j)}$, $1 \leq i \leq m$ and $1 \leq j \leq \ell$, where ℓ is

Algorithm 1 memEAPF pseudocode

```
1: procedure MEMEAPF(q_0, q_f, map)
2: FitFunc ← @ APF(q_0, q_f, map, k_a, k_r, η, ϵ, M)
3: t ← 0
4: N_gen ← number of generations
5: m ← number of elementary membranes
6: initialize population P(t)
7: initialize membranes S_0 and S_i, 1 ≤ i ≤ m
8: initialize EAPF_param
9: while t < N_gen do
10:    for each elementary membrane S_i in parallel do
11:       [k_a, k_r, η] ← EAPF(EAPF_param, FitFunc)
12:    end for
13:    merge all the elementary membranes S_i, into S_F
14:    apply communication rules in S_F
15:    find global best individual [k_a, k_r, η]_best
16:    divide S_F in m elementary membranes S_i
17:    t ← t + 1
18: end while
19: [Q_G, fitValue, nConf, goal] ← FitFunc using [k_a, k_r, η]_best
20: return [Q_G, fitValue, nConf, goal]
21: end procedure
```

the number of individuals in the subpopulation,

$$\mathcal{P}(t)\begin{cases} P_1(t) = \overbrace{[k_a, k_r, \eta]_{(1,1)}}^{p_{(1,1)}}, \ldots, \overbrace{[k_a, k_r, \eta]_{(1,\ell)}}^{p_{(1,\ell)}} \\ P_2(t) = [k_a, k_r, \eta]_{(2,1)}, \ldots, [k_a, k_r, \eta]_{(2,\ell)} \\ \quad\vdots \\ P_m(t) = [k_a, k_r, \eta]_{(m,1)}, \ldots, [k_a, k_r, \eta]_{(m,\ell)}. \end{cases} \tag{1}$$

The membrane structure $\mu = [_0[_1]_1, [_2]_2, \ldots, [_m]_m]_0$ is initialized in accordance with the number of membranes m, i.e., the skin membrane $\mathcal{S}_0$ is composed of m elementary membranes $\mathcal{S}_i$, and each elementary membrane is initialized with a subpopulation $P_i(t)$ containing ℓ individuals.

The initial multisets are composed of the individuals that are distributed among the elementary membranes S_i, as follows:

$$\begin{aligned} \mathcal{S}_0 &= \{\lambda\}, \\ \mathcal{S}_1 &= \{p_{(1,1)}, p_{(1,2)}, \ldots, p_{(1,\ell)}\}, \\ \mathcal{S}_2 &= \{p_{(2,1)}, p_{(2,2)}, \ldots, p_{(2,\ell)}\}, \\ &\vdots \\ \mathcal{S}_m &= \{p_{(m,1)}, p_{(m,2)}, \ldots, p_{(m,\ell)}\}, \end{aligned} \tag{2}$$

where λ is an empty string, and $p_{(i,j)}$ is a solution represented as

$$p_{(i,j)} = [k_a, k_r, \eta]_{(i,j)}, \tag{3}$$

where the chromosome $p_{(i,j)}$ consists of three genes (bit string with a fixed length). The first gene corresponds to k_a, the second gene to k_r, and the third contains η. The data structure named $EAPF_{param}$ contains all the parameter values needed by the EAPF (Algorithm 2), and the subpopulation P_i corresponding to each elementary membrane $\mathcal{S}_i$. In specific, the $EAPF_{param}$ contains the mutation rate, selection rate, and the maximum number of generations allowed for the EAPF algorithm, which is utilized in each elementary membrane S_i.

In Algorithm 1, the main purpose of the iterative process is to obtain the global best individual $[k_a, k_r, \eta]_{best}$. For each elementary membrane S_i, the EAPF procedure is launched to obtain the best individual of the generation t. Once that each elementary membrane has finished its task, all the membranes $\mathcal{S}_i$, except the skin membrane $\mathcal{S}_0$, merge into $\mathcal{S}_F$ where the communication rules are applied. These rules consist in including a copy of the best-selected individual (from each elementary membrane S_i) into the merged membrane $\mathcal{S}_F$. The purpose is to find the global best individual from the best-selected group of individuals and send out into the skin membrane a copy of this global best individual to maintain the current global best solution. Through the communication rules, the merged membranes exchange information that will be evolved in the next generation. During the merge process, each subpopulation is maintained, and the worst individuals are replaced by a copy of the best-selected individuals to improve the subpopulation in each elementary membrane. The best path Q_G (solution) is generated using the function handler *FitFunc* and the global best individual ($[k_a, k_r, \eta]_{best}$) that contains the solution parameters. The memEAPF algorithm returns the path Q_G, the path length *fitValue*, the number of MR configurations *nConf* required to achieve the goal, and a flag *goal* indicating if the goal was achieved.

4.1.2. EAPF algorithm

Algorithm 2 describes the EAPF procedure used by each elementary membrane $\mathcal{S}_i$ to evolve the sets of parameters. It is a genetic algorithm using the APF procedure (Algorithm 3) as the fitness function (*FitFunc*). The EAPF procedure has two input parameters, the $EAPF_{param}$ and the *FitFunc*. The output parameter is the evolved solution $[k_a, k_r, \eta]$; i.e., the best set of parameters required to generate the path.

Algorithm 2 EAPF pseudocode

```
1: procedure EAPF(EAPF_param, FitFunc)
2: τ ← 0
3: evaluate each individual in P_i(τ) using FitFunc
4: while not termination do
5:     copy of best individuals R(τ) ← P_i(τ)
6:     select parents P'_i(τ) ← P_i(τ)
7:     perform crossover P'_i(τ)
8:     perform mutation P'_i(τ)
9:     evaluate each individual in P'_i(τ) using FitFunc
10:    P_i(τ + 1) ← P'_i(τ) ∪ R(τ)
11:    τ ← τ + 1
12: end while
13: return [k_a, k_r, η]
14: end procedure
```

In Algorithm 2, every individual of the population $P_i(\tau)$ is evaluated using the APF procedure through the function handler *FitFunc*, where the best individual has the shortest feasible path length to reach the goal. The selection process drives the EAPF to improve the population fitness over the successive generations. In this process, the individuals in $P_i(\tau)$ are sorted from the best to the worst based on fitness values. Where, $R(\tau)$ is a special set that contains a copy of the best individuals with a guaranteed place in the next generation without undergoing variation, and $P'_i(\tau)$ is a set of the best individuals selected as parents. The size of $R(\tau)$ is $1 - selection\ rate$ and the size of $P'_i(\tau)$ is indicated by the *selection rate*. E.g., if *selection rate* $=$ 0.8 then the 20% of $P_i(\tau)$ is selected and copied to $R(\tau)$ starting from the best individuals and 80% of $P_i(\tau)$ is selected and assigned to $P'_i(\tau)$ also starting from the best individuals. The crossover is performed using a single point crossover; where a random point in the two chromosomes (parents) is chosen. The offspring is formed by joining the genetic material to the right of the crossover point of one parent, and the genetic material to the left of the crossover point of the other parent. The mutation process is performed using random mutations that alter a certain percentage of the bits in the list of chromosomes. The mutation process tends to distract the evolution from converging on a popular solution.

4.1.3. APF algorithm

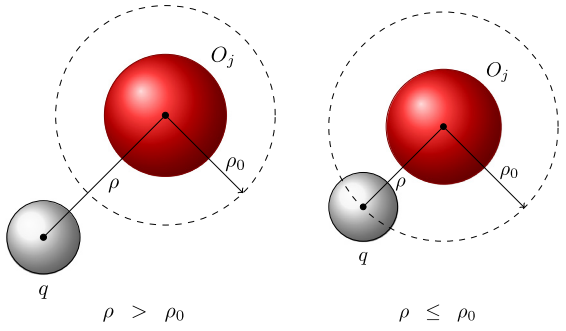
The soft computing solution of the APF procedure described by Algorithm 3 is based on the original APF hard-computing mathematical method proposed by Khatib [12] and modified by Montiel et al. [51]. The memEAPF algorithm computes the global best set of parameters $[k_a, k_r, \eta]_{best}$ through its elementary membranes $\mathcal{S}_i$, composed of EAPF procedures, as a result, the APF algorithm can provide paths tagged as Q_G to reach the goal without colliding obstacles that lie between the start and the goal point, which is a distinctive advantage over the original APF method [12] and further state-of-the-art innovations.

In the memEAPF, the APF algorithm has two different tasks: First, it is used as the fitness function in the EAPF procedure at each elementary membrane $\mathcal{S}_i$; and second, once that the memEAPF has computed the global best set of parameters, it provides the best path Q_G (solution).

The input parameters of Algorithm 3 are: the start position q_0 of the MR, the goal position q_f that the MR must achieve, a data structure called *map* that contains the composition of the environment, the attractive proportional gain k_a, the repulsive proportional gain k_r, the MR's step size η, the goal approximation radius ϵ, and M that is the maximum number of allowed configurations.

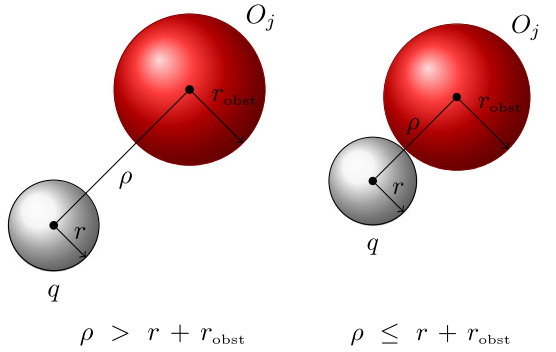
Algorithm 3 APF pseudocode

```
1: procedure APF(q_0, q_f, map, k_a, k_r, η, ϵ, M)
2: safe ← True
3: i ← 0
4: fitValue ← 0
5: d_a ← ‖q_f − q_0‖
6: while d_a > ϵ and i < M and safe do
7:     U_total(q(i)) ← U_att(q(i)) + Σ_{j=1}^{n} U_rep(q(i))_j
8:     F(q(i)) ← −∇U_total(q(i))
9:     q(i + 1) ← q(i) + η ∗ F(q(i))/‖ F(q(i)) ‖
10:    d_a ← ‖q_f − q(i + 1)‖
11:    fitValue ← fitValue+ ‖ q(i + 1) − q(i) ‖
12:    if ρ ≤ r + r_obst then
13:        safe ← False
14:    end if
15:    i ← i + 1
16: end while
17: Q_G ← [q(0), q(1), q(2), . . . , q(i)]
18: nConf ← i
19: if d_a ≤ ϵ then
20:    goal ← True
21: else
22:    goal ← False
23: end if
24: return [Q_G, fitValue, nConf, goal]
25: end procedure
```



(a) The MR is free of the influence of U_{rep}

(b) The MR is under the influence of U_{rep}



(c) Safe condition for MR navigation

(d) Unsafe condition for MR navigation

Fig. 3. The distance ρ and its relationship between MR and obstacles.

Algorithm 3 employs an iterative process to generate all the required configurations to form the path that the MR should follow to reach the goal. During and before this iterative process, the distance d_a is calculated to determine the Euclidean distance between the MR (current position) and the goal. The total potential field $U_{total}(q)$ proposed by Khatib [12] is calculated using the sum of the attraction potential field

$$U_{att}(q) = \frac{1}{2}k_a(q - q_f)^2, \tag{4}$$

and the repulsion potential field,

$$U_{rep}(q) = \begin{cases} \frac{1}{2}k_r\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho \leq \rho_0 \\ 0 & \text{if } \rho > \rho_0 \end{cases} \tag{5}$$

where, ρ_0 is the limit distance of influence of the potential field, and ρ is the distance between the center of the MR q and the center of the obstacle O_j, see Figs. 3(a) and 3(b). In Algorithm 3, the gradient descent operation is applied to the total potential field $U_{total}(q)$ to obtain the total force $F(q)$. Every new MR configuration $q(i + 1)$ in the path is calculated using the current position $q(i)$, the step size η, and the total force $F(q)$. And in every new MR configuration, the *fitValue* is updated to know the path length until the current position. At the end, the algorithm will return the path length contained in *fitValue* that is the sum of distances between the configurations from the start to the goal point [34,52], i.e.,

$$fitValue = \sum_{i=0}^{nConf-1} L(i, i + 1), \tag{6}$$

where, $L(i, i + 1) = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$ is the distance between configurations $i = (x_i, y_i)$ and $i + 1 = (x_{i+1}, y_{i+1})$.

In line 12 of Algorithm 3, the safety condition is evaluated. If the sum of r (radius of the MR) and r_{obst} (radius of the nearest obstacle to the MR) is greater than or equal to ρ there will be an unsafe condition (e.g., collision, see Fig. 3(d)). Therefore, the flag *safe* becomes *False* and the loop will be terminated. Otherwise, if the sum of r and r_{obst} is less than ρ there will be a safe condition for the navigation (see Fig. 3(c)) and the process will continue. The parameters ϵ, M, and the flag *safe* are the stop conditions for the algorithm; i.e., the loop will finish if the MR has reached the goal with a precision $d_a \leq \epsilon$ or the maximal number of configurations M has been accomplished or a collision has occurred. Once the loop has reached the stop condition, the path Q_G has been built using all the MR configurations calculated, from $q(0)$ to $q(i)$, i.e., from q_0 to q_f.

4.2. Implementation

In this section, we present the simulation platform to validate the memEAPF using experimental data. Algorithm 4 provides the pseudocode that allows implementing the simulation platform. Fig. 4 illustrates our software implementation, particularly we used Matlab/C++ programming languages. The simulation platform can work in off-line and on-line test modes. It contains a set of environments, which are commonly used as benchmark test problems to evaluate MR path planning algorithms; since these environments present path planning problems that are particularly challenging or even impossible to solve by many MR path planning algorithms [15,53,54].

The MR uses Algorithm 4 to simulate the navigation from the start position q_0 to the goal position q_f under a known, partially known, or unknown environment. Algorithm 4 uses the memEAPF (Algorithm 1) to obtain the best path in the off-line and on-line modes, as it can be observed in lines 3 and 11 of Algorithm 4. For off-line mode, before the navigation starts, the memEAPF algorithm is called to get the path that will drive the MR to the goal. If there is a feasible path (if the memEAPF set the flag *goal*

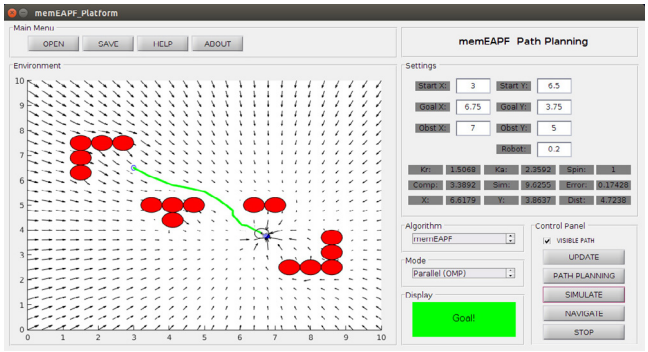


Fig. 4. Experimental simulation platform.

as *True*) the navigation will be performed. During the navigation process, an environment verification is performed to see if new obstacles not considered at the beginning were added or dynamic obstacles have changed their position. If the environment change, the data structure *map* is updated, and the current configuration becomes in the new start configuration q_0, then the memEAPF algorithm is called once again to update the path in the current environment (path planning in on-line mode). The counter *count* is an index destined to tour all the $Q_G(count)$ configurations in sequential ascending order.

Algorithm 4 Experimental simulation platform pseudocode

```
1:  procedure NAVIGATION(q_0, q_f, map)
2:  count ← 0
3:  [Q_G, fitValue, nConf, goal] ← MEMEAPF(q_0, q_f, map)
4:  if goal then
5:      navigation ← True
6:      while navigation do
7:          verification of the environment
8:          if environment has changed then
9:              update map
10:             q_0 ← Q_G(count)
11:             [Q_G, fitValue, nConf, goal] ←MEMEAPF(q_0, q_f, map)
12:             if goal then
13:                 count ← 0
14:             else
15:                 display "Unreachable goal!"
16:                 navigation ← False
17:             end if
18:         else
19:             navigate from Q_G(count) to Q_G(count + 1)
20:             count ← count + 1
21:         end if
22:         if count ≥ nConf then
23:             navigation ← False
24:             display "Goal!"
25:         end if
26:     end while
27: else
28:     display "Unreachable goal!"
29: end if
30: end procedure
```

The experimental simulation platform consists of four modules based on the algorithms previously described. It was tailored to take advantage of the Matlab numerical computing environment and the C++ programming language. The algorithms memEAPF, EAPF, and APF were programmed using Matlab/C++. The navigation algorithm, user interface, and plots were programmed using Matlab; hence, the implementation is a mixed up of Matlab and C++. The modules of the experimental simulation platform were integrated through the Matlab MEX-file tool. The main screen of the experimental simulation platform is shown in Fig. 4.

5. Experimental results

In this section, we describe the test protocol and the results of the different implementations of the memEAPF working in off-line and on-line modes. A comparative study of the memEAPF against the parallel evolutionary artificial potential field (PEAPF) [51], the pseudo-bacterial potential field (PBPF) [55], and the bacterial potential field (BPF) [43] is given. Moreover, the performance of the memEAPF considering sequential and parallel implementations are also given.

The PEAPF, PBPF, and BPF are state-of-the-art potential field based algorithms that share with the memEAPF the characteristic of hybridization with a soft computing method to solve path planning problems. The PEAPF is a high-performance hybrid metaheuristic that makes use of the APF method and mathematical programming to solve efficiently path planning problems [51]. The PBPF algorithm uses a pseudo-bacterial genetic algorithm and a fitness function based on the potential field concepts to construct viable paths for autonomous mobile robot navigation [55]. The BPF algorithm makes use of the APF method with a bacterial evolutionary algorithm to obtain an enhanced flexible path planner method for MRs [43].

5.1. Test protocol

Considering that an MR configuration is defined as $q(x, y, r, \theta)$. In all the experiments, we considered that the MR in the initial position q_0 is centered over the starting point and oriented to the first coordinate to visit ($\theta = \theta_0$), hence we have a radius $r_0 = 0$, obtaining the initial configuration $q(x_0, y_0, r_0, \theta_0)$. For the goal position, we used the value $r_f = \epsilon$ to control how far from the goal coordinate (x_f, y_f) the MR must stop, moreover, in the penultimate configuration the MR is oriented to the goal coordinate to visit (x_f, y_f), hence we have ($\theta = \theta_f$) and the final configuration is $q(x_f, y_f, r_f, \theta_f)$. The aforementioned is an advantage since it is not always physically possible to start or finish the navigation precisely over these positions.

Table 1 shows the test environments that contain the MR's missions with a start and goal configuration, q_0 and q_f respectively; as well as the environment layout, the environment is formed by n obstacles $O_j(x_j, y_j, r_{obst(j)})$, for $1 \leq j \leq n$, which are placed at the coordinates (x_j, y_j) and they have a radius $r_{obst(j)}$. Each test environment was designed to evaluate the performance and accuracy of the memEAPF, these environments were named as M01, M02, …, M12.

The environments M01 to M12 cover well-known difficult problems, e.g., trap sites due to local minima, path-following prediction problems, problematic areas to reach because the goal is very close to an obstacle, and other situations described in [53,54]. These environments are challenging problems for testing path planning algorithms, so we used them to evaluate the memEAPF. They represent just a sample of the types of scenarios that the MR can expect to find in typical real-world indoor environments, which are composed of walls, corridors, barriers, U-shaped and L-shaped obstacles, among others.

In all the experiments, we set up the memEAPF as follows:

1. The MR model is given by $q(x, y, r, \theta)$, where the MR has a physical size of $r = 0.2$ meters, and θ is always oriented to the next coordinate position to visit.

Table 1
Test environments, M01 to M12. For each environment, we have the MR's mission (start q_0 and goal q_f), and the environment layout (obstacles O_j).

Environment	Start q_0 (x_0, y_0)	Goal q_f (x_f, y_f)	Obstacles $O_j(x_j, y_j, r_{obst(j)})$
M01	(6.5,8.0)	(6.0,3.0)	(6.0,5.0,0.5),(4.0,5.0,0.5),(3.2,5.0,0.5),(2.4,5.0,0.5),(6.8,5.0,0.5)
M02	(5.0,9.0)	(5.0,1.0)	(4.0,6.5,0.5),(2.5,6.5,0.5),(5.0,3.5,0.5),(6.5,3.5,0.5),(8.0,3.5,0.5)
M03	(5.0,9.0)	(5.0,1.0)	(4.0,5.1,0.5),(5.0,5.1,0.5),(6.0,5.1,0.5),(4.0,6.1,0.5),(4.0,7.1,0.5)
M04	(5.0,8.0)	(5.0,2.0)	(5.0,4.5,0.5),(3.5,4.5,0.5),(3.5,6.0,0.5),(6.5,4.5,0.5),(6.5,6.0,0.5)
M05	(2.0,3.8)	(8.0,6.3)	(4.0,3.8,0.5),(5.0,3.8,0.5),(6.0,3.8,0.5),(6.0,2.8,0.5),(6.0,1.8,0.5), (6.0,6.3,0.5),(5.0,6.3,0.5),(4.0,6.3,0.5),(4.0,7.3,0.5),(4.0,8.3,0.5)
M06	(5.0,9.0)	(5.0,1.0)	(2.0,7.5,0.5),(3.0,7.5,0.5),(4.0,7.5,0.5),(4.0,5.0,0.5),(5.0,5.0,0.5), (6.0,5.0,0.5),(6.0,2.5,0.5),(7.0,2.5,0.5),(8.0,2.5,0.5)
M07	(5.5,9.0)	(4.5,3.0)	(2.0,7.5,0.5),(3.0,7.5,0.5),(4.0,7.5,0.5),(4.0,5.0,0.5),(5.0,5.0,0.5), (6.0,5.0,0.5),(6.0,2.5,0.5),(7.0,2.5,0.5),(8.0,2.5,0.5),(2.0,5.5,0.5), (2.0,6.5,0.5),(8.0,3.5,0.5),(8.0,4.5,0.5)
M08	(2.0,7.0)	(8.0,3.0)	(3.8,1.8,0.5),(2.5,8.3,0.5),(3.5,8.3,0.5),(3.5,7.3,0.5),(3.5,6.3,0.5), (6.5,3.8,0.5),(6.5,2.8,0.5),(6.5,1.8,0.5),(7.5,1.8,0.5),(3.8,2.8,0.5), (1.5,8.3,0.5),(8.5,1.8,0.5)
M09	(5.0,8.0)	(6.0,2.0)	(4.3,5.0,1.0),(5.8,5.0,1.0)
M10	(3.0,6.5)	(6.8,3.8)	(7.4,2.5,0.3),(8.0,2.5,0.3),(8.6,2.5,0.3),(8.6,3.1,0.3),(8.6,3.7,0.3), (1.5,6.3,0.3),(1.5,6.9,0.3),(1.5,7.5,0.3),(2.1,7.5,0.3),(2.7,7.5,0.3), (3.5,5.0,0.3),(4.1,5.0,0.3),(4.7,5.0,0.3),(4.1,4.4,0.3),(6.4,5.0,0.3), (7.0,5.0,0.3)
M11	(5.0,9.0)	(5.0,1.0)	(4.0,6.5,0.5),(2.5,6.5,0.5),(5.0,3.5,0.5),(6.5,3.5,0.5),(8.0,3.5,0.5), (3.3,6.5,0.5),(5.8,3.5,0.5),(7.3,3.5,0.5)
M12	(1.5,7.5)	(8.5,3.0)	(5.0,5.0,0.3),(3.5,3.5,0.5),(3.5,6.5,0.5),(6.5,3.5,0.5),(6.5,6.5,0.5)

2. The number m of elementary membranes $\mathcal{S}_i$ has been varied to see the effect of adding elementary membranes, to contrast with the results obtained regarding path length in the set of test environments. We have used $m \in \{2, 4, 8, 16\}$.
3. The stop condition is $N_{gen} = 100$ generations.
4. Each elementary membrane $\mathcal{S}_i$ contains a subpopulation size of $\ell = 16$ individuals.
5. In the merge process, three-quarters of the subpopulation is maintained, and the rest is replaced by a copy of the best individuals to improve the subpopulation in each elementary membrane $\mathcal{S}_i$.
6. Each individual $p_{(i,j)}$ consists of three genes, k_a, k_r and η; where the gain parameters are constrained, $\{k_a, k_r \mid 0 < k_a, k_r < 10\}$. Note, in this criterion we are using just one repulsive proportional gain k_r. We know that depending on the number of obstacles $[O_1, \ldots, O_n]$ several $[k_{r1}, \ldots, k_{rn}]$ repulsive proportional gains may exist; however, because all the reported results in literature consider just one k_r, and with the aim of being compatible with these reference marks, we have used only one k_r value for all the obstacles.
7. The elitist parameter selection in $EAPF_{param}$ was set to 0.5. The strategy of elitist selection is performed over the population $P_i(\tau)$, where the 50% of individuals with the best fitness values are chosen as parents and assigned to $P_i'(\tau)$, and a copy of these individuals is assigned to $R(\tau)$ guaranteeing a place in the next generation without undergoing mutation. This strategy is employed to prevent the random destruction by mutation operator of individuals with good genetics.
8. The mutation parameter in $EAPF_{param}$ was set to 0.2. Random mutations alter the 20% of the bits in the list of chromosomes. The mutation process is employed to generate diversity and avoid the evolution from converging on a popular solution.
9. We set $M = 2000$ to indicate the maximum number of steps (MR configurations) allowed.

All the experiments were carried out on a quad-core Intel i7-4770 CPU (3.40 GHz) with 8 GB of RAM running the Ubuntu Trusty distribution of Linux with the GNU Compiler Collection (GCC 4.8), and Matlab (version 8.5).

5.2. Off-line path planning

In off-line path planning, all the objects on the environment are in known positions [56]. The aim is to find a feasible collision-free path between two points, start q_0 and goal q_f, in a static environment composed of walls and obstacles. For this test, the cost of the path was calculated using the path length (MR travel distance) [44,57], that is computed by *fitValue* in line 11 of Algorithm 3 using (6).

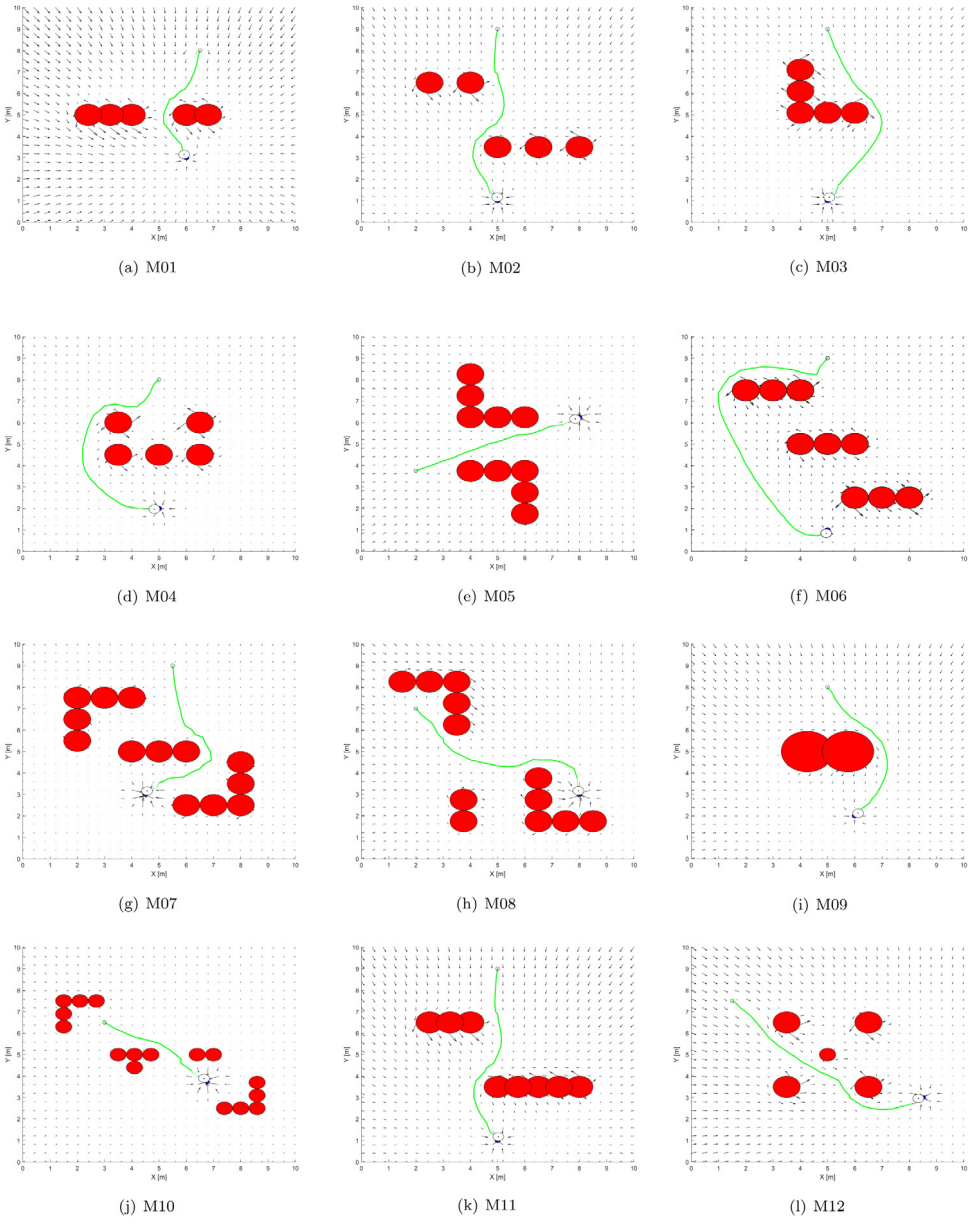
The environments described in Table 1 were used to test the memEAPF algorithm in off-line mode. Fig. 5 shows the best (suboptimal or optimal in the best of cases) path Q_G, these paths have the shortest path lengths obtained with the memEAPF in off-line mode for each test environment. Table 2 shows the results obtained by the memEAPF in off-line mode with $m = 2, 4, 8$, and 16 elementary membranes; the parameters $k_{a(best)}$ and $k_{r(best)}$ obtained by the memEAPF with $m = 16$ to generate the best path (see Fig. 5), its path length *fitValue* in meters, and the number of configurations *nConf* required to reach the goal q_f for each test environment.

In Table 2, the results show that the best paths were found by the memEAPF with $m = 16$. It can be observed that the quality of solutions (shorter paths) increase as the number of elementary membranes $\mathcal{S}_i$ grew up. A point to consider is that above from $m = 16$ the improvement starts to be small and the computational time is increased in a significant way. On the other hand, in performance terms (Section 5.4) the use of $m = 2$ means that at least we can have two processes running in parallel. The upper bound of $m = 16$ is because we can handle 16 threads in our machine without any interference among them.

Table 3 shows a summary of statistical results for the different implementations. To compare the memEAPF with the PEAPF, the PBPF and the BPF, all in off-line mode, we ran each algorithm 30 times for each test environment (M01, M02, …, M12). Table 3 shows the best, mean, worst, and standard deviation representing the best solution for each environment tested. It can be observed from the best and mean solutions that memEAPF is superior in all test environments. The statistical technique in [52,58] is used to analyze the behavior of PEAPF, PBPF, BPF and memEAPF over the 12 test environments. To check whether the algorithms are

Table 2
Path planning results (path length in meters) obtained by the memEAPF in off-line mode with $m = 2$, 4, 8, and 16 elementary membranes. For the memEAPF with $m = 16$ the best proportional gains ($k_{a(best)}$ and $k_{r(best)}$) and the MR's number of configurations ($nConf$) are shown.

Environment	$m = 2$	$m = 4$	$m = 8$	$m = 16$	$k_{a(best)}$	$k_{r(best)}$	$nConf$
M01	5.4635	5.4627	5.4609	5.4600	4.5909	9.8700	400
M02	8.5799	8.5664	8.5627	8.5558	1.0599	3.0111	671
M03	8.9435	8.9409	8.9386	8.9352	0.7625	2.9056	590
M04	9.3238	9.3221	9.3127	9.3122	0.8753	5.1589	714
M05	6.3895	6.3845	6.3768	6.3761	0.6053	0.2072	526
M06	11.3821	11.2328	11.2115	11.1761	0.6969	4.9608	1682
M07	7.7552	7.7544	7.7460	7.7252	0.3589	0.9639	855
M08	8.3021	8.2899	8.2862	8.2837	0.7068	1.8189	749
M09	6.9649	6.9622	6.9596	6.9588	2.5875	9.4633	463
M10	4.7174	4.7023	4.6855	4.6746	0.5029	0.2977	453
M11	8.4456	8.4455	8.4416	8.4388	1.7842	3.5018	623
M12	9.2489	9.2461	9.2440	9.2406	1.7422	5.4788	545



(a) M01 (b) M02 (c) M03

(d) M04 (e) M05 (f) M06

(g) M07 (h) M08 (i) M09

(j) M10 (k) M11 (l) M12

Fig. 5. Path planning (solution) for the different test environments. Each map shows the best path Q_G obtained by the memEAPF algorithm in off-line mode.

significantly different or not, the t-test is performed. The level 0.05 of significance is considered. Table 3 shows the results of t-test, where the symbol '+' denotes significant difference between memEAPF versus PEAPF, memEAPF versus PBPF, and memEAPF versus BPF, with respect to t-test. Moreover, the results indicate that memEAPF outperforms the other methods in all the cases

because the p-values are far smaller than the level 0.05 of significance.

5.3. On-line path planning

Real-world applications frequently face the MR with unknown or partially known environments, which requires the ability to respond and taking decisions without detrimental of controllability, this action is referred as on-line path planning. Some works perform off-line path planning to know the initial conditions of the environment, and once the navigation starts they change to the on-line mode in order to adjust the path already known, considering the modified environment [51].

The purpose of the on-line path planning experiments is to show the advantage of the memEAPF in path planning in partially known environments, where random positioning of static obstacles and dynamic obstacles exist.

Next, we describe the experiments and provide the results for on-line path planning with obstacles random placement, and for on-line path planning with non-static obstacles.

5.3.1. On-line path planning with random static obstacles placement

We start with a known environment; hence the position of the static obstacles was established in advance, in the test environment. Once the best path was obtained, a random interfering obstacle is placed on the path, afterward remaining static.

Fig. 6 illustrates an on-line path planning experiment that was achieved using the test environment M04 of Table 1, which is found in many real-life scenarios. To achieve the experiment, we set up the path planner and follow the next steps:

1. The experiment starts with a known environment, consisting of five obstacles forming a U-shape trap, as is shown in Fig. 6(a). The memEAPF was configured for $m = 16$ (sixteen elementary membranes).
2. The path planning is performed in off-line mode because we just know the environment information described by the original environment M04. The minimum path length found to reach the goal is 9.3122 m, using the best set of gain parameters ($k_{a(best)} = 0.8753$ and $k_{r(best)} = 5.1589$), as it is described in Table 2. The resultant path is shown in Fig. 6(b).
3. At position (3.5, 2.5), a new obstacle is added to change the environment configuration. After a while, when the MR has traveled a distance of 6.7619 m, it reaches the position (2.6889, 3.0928). The MR senses the new obstacle, it calculates the obstacle position to update the environment layout *map*, as shown in Fig. 6(c).
4. The path planner of the MR (memEAPF algorithm) now has a different environment layout; hence, it is necessary to update the path by recalculating the set of gain parameters. The new best gain parameters are $k_{a(best)} = 3.8594$ and $k_{r(best)} = 9.2830$, and the new minimum path length to reach the goal from the new position at (3.5, 2.5) is 2.8947 m, as is described in Table 4. Finally, the MR follows the new path to reach the goal, the total path length from the original start point to the goal is 6.7619 + 2.8947 = 9.6566 m, the complete path is shown in Fig. 6(d).

The above procedure was repeated for the PEAPF, PBPF, BPF with the aim to compare with the memEAPF. Table 4, which is a summary of the results of executing thirty times these experiments. In all the cases, the memEAPF performed better.

5.3.2. On-line path planning with non-static obstacles

This case study considers non-static obstacles which can be humans or other robots moving through the environment. Differently to the case of random static obstacle placement, here the obstacle will not remain static, it will be moving with a defined trajectory that is unknown to the MR. This problem is more complicated than those mentioned above because it faces the planner to find a feasible path in a noisy surface because at every iteration the searching zone is different [59].

Fig. 7 illustrates an on-line path planning experiment that was achieved using the test environment M03 of Table 1. To achieve the experiment, we set up the path planner and followed the next steps:

1. The start position q_0 of the MR is at coordinate (5.0, 9.0), and the goal position q_f is at (5.0, 1.0), at the beginning of the experiment. This initial scenario presents five obstacles forming an L-shaped barrier, as shown in Fig. 7(a).
2. The path planning is performed in off-line mode. The minimum path length found to reach the goal was 8.9352 m, using the best set of gain parameters ($k_{a(best)} = 0.7625$ and $k_{r(best)} = 2.9056$) found by the memEAPF algorithm with $m = 16$, as described in Table 2.
3. A non-static obstacle is placed at the position (4.0, 2.0); at this time the non-static obstacle is unknown for the MR, so the non-static obstacle was not considered in the initial path planning, as shown in Fig. 7(b).
4. Once the path was obtained (off-line mode), the MR starts to navigate (on-line mode). The non-static obstacle will be moving, and eventually, it will obstruct the path that the MR must follow. During the navigation of the MR, when it has traveled a distance of 6.5924 m. At the position (6.3395, 3.0726) the MR senses the new obstacle, which was not considered at the off-line path planning stage, see Fig. 7(c). At this point, the environment is updated, and the memEAPF algorithm starts to recalculate the set of gain parameters (on-line mode) to avoid colliding with the new obstacle.
5. Fig. 7(d) shows the updated path for avoiding the non-static obstacle. The new best gain parameters are $k_{a(best)} = 0.4174$ and $k_{r(best)} = 0.4206$. They are used to obtain the shortest path to reach the goal from the new start point (2.5215 m), as described in Table 4. The total path length from the original start point to the goal is 6.5924 + 2.5215 = 9.1139 m.

Fig. 8 shows a second test with several non-static obstacles. For this experiment the test environment M03 (Fig. 7(a)) described in Table 1 was employed. After the off-line path planning (Fig. 7(b)) the MR starts to navigate, and the on-line path planning will be employed. When the MR has traveled 1.9826 m, it detects the first non-static obstacle (Fig. 8(a)). Then, after 2.2472 m, the MR detects the second non-static obstacle (Fig. 8(b)). Last, when the MR has traveled 3.9610 m, it detects the third non-static obstacle (Fig. 8(c)). Finally, the MR follows the new path to reach the goal (Fig. 8(d)), the total path length from the original start point to the goal is 1.9826 + 2.2472 + 3.9610 + 1.9579 = 10.1487 m.

The aim of the results shown in Table 4 is to determine which algorithm provides the shortest path for the on-line implementations in static environments with random static obstacle placement, and non-static obstacles. To compare the memEAPF with the PEAPF, the PBPF, and the BPF, we ran each algorithm 30 times for each test. It can be observed from the best and mean solutions that memEAPF is superior in the test environments. To check whether the algorithms are statistically different or not, the t-test is performed. The level 0.05 of significance is considered. The results indicate that memEAPF outperforms in all test environments because the p-values are far smaller than the level 0.05 of significance.

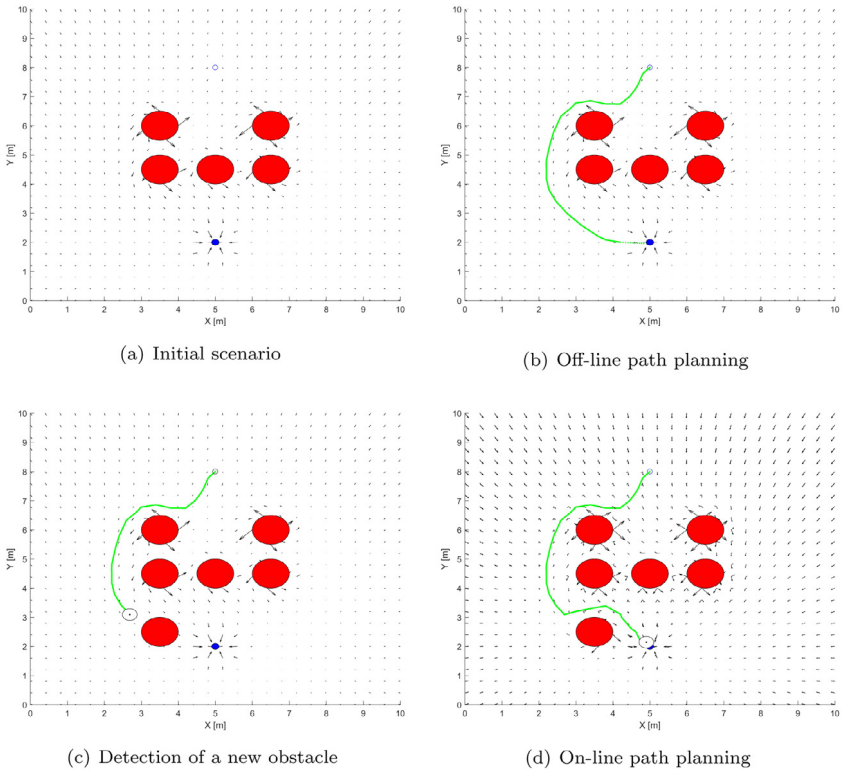
Table 3
A comparison between PEAPF, PBPF, BPF, and memEAPF with $m = 16$. Best, mean, worst and standard deviation represent best solution (shortest path length) over independent 30 runs, respectively in each test environment. Path length units are in meters. Significant difference is represented by '+'.

Environment	Statistics	PEAPF	PBPF	BPF	memEAPF
M01	**Best**	**5.7839**	**5.6461**	**5.6033**	**5.4600**
	Mean	6.3828	6.1133	5.7992	5.4661
	Worst	7.6062	7.1003	6.6651	5.4779
	Std. Dev.	0.4946	0.3946	0.2814	0.0035
	t-test	6.89e−11(+)	1.99e−9(+)	3.58e−4(+)	–
M02	**Best**	**9.1234**	**8.8565**	**8.8253**	**8.5558**
	Mean	9.5349	9.6823	9.2346	8.5735
	Worst	12.6986	12.5651	12.1282	8.5875
	Std. Dev.	0.6020	1.2269	0.8156	0.0084
	t-test	1.80e−9(+)	9.45e−5(+)	1.19e−2(+)	–
M03	**Best**	**9.4492**	**9.2910**	**9.2325**	**8.9352**
	Mean	9.9902	9.6850	9.7293	8.9432
	Worst	10.8700	10.6865	10.6025	8.9504
	Std. Dev.	0.4360	0.3812	0.4861	0.0032
	t-test	1.44e−13(+)	2.07e−5(+)	4.55e−4(+)	–
M04	**Best**	**10.0683**	**10.0133**	**9.7815**	**9.3122**
	Mean	11.1489	10.6486	10.3231	9.3309
	Worst	12.3318	12.1611	12.0800	9.3599
	Std. Dev.	0.7162	0.5119	0.5706	0.0101
	t-test	3.57e−14(+)	5.56e−13(+)	2.86e−10(+)	–
M05	**Best**	**6.7931**	**6.5934**	**6.5721**	**6.3761**
	Mean	7.1459	6.6420	6.6202	6.3917
	Worst	8.1966	6.7374	6.6592	6.4168
	Std. Dev.	0.3459	0.0320	0.0239	0.0102
	t-test	1.52e−12(+)	2.20e−16(+)	2.20e−16(+)	–
M06	**Best**	**13.4315**	**13.1992**	**13.2194**	**11.1761**
	Mean	13.5494	13.5564	13.4285	12.9316
	Worst	16.5236	14.5773	13.9072	13.2331
	Std. Dev.	0.5538	0.2928	0.1922	0.6800
	t-test	3.68e−4(+)	5.13e−5(+)	6.02e−4(+)	–
M07	**Best**	**8.1361**	**8.0234**	**7.9446**	**7.7252**
	Mean	8.5174	8.3421	8.2504	7.7665
	Worst	9.6456	9.3778	9.8828	7.7782
	Std. Dev.	0.4882	0.4392	0.5292	0.0135
	t-test	3.92e−9(+)	3.47e−6(+)	3.14e−5(+)	–
M08	**Best**	**8.9660**	**9.0150**	**8.7588**	**8.2837**
	Mean	9.3611	9.5172	9.1060	8.2951
	Worst	10.2599	10.5228	9.8451	8.3104
	Std. Dev.	0.3630	0.4422	0.3181	0.0063
	t-test	8.43e−16(+)	8.88e−12(+)	6.96e−13(+)	–
M09	**Best**	**7.3781**	**7.2646**	**7.1973**	**6.9588**
	Mean	8.0013	7.7854	7.5615	6.9653
	Worst	10.1659	9.5631	9.8351	6.9714
	Std. Dev.	0.5672	0.6064	0.5122	0.0036
	t-test	9.58e−11(+)	5.06e−8(+)	7.65e−7(+)	–
M10	**Best**	**4.9546**	**4.8519**	**4.8103**	**4.6746**
	Mean	5.1132	4.9184	4.8326	4.7212
	Worst	5.8227	5.0591	4.9076	4.7537
	Std. Dev.	0.2210	0.0587	0.0254	0.0199
	t-test	1.72e−10(+)	3.06e−11(+)	2.20e−16(+)	–
M11	**Best**	**8.9952**	**8.7750**	**8.6808**	**8.4388**
	Mean	9.4128	8.9326	8.8702	8.4477
	Worst	12.0849	9.5338	9.2432	8.4556
	Std. Dev.	0.7266	0.2030	0.1972	0.0043
	t-test	7.14e−8(+)	7.17e−6(+)	1.92e−3(+)	–
M12	**Best**	**9.8120**	**9.6016**	**9.5519**	**9.2406**
	Mean	10.3427	10.1014	9.8560	9.2517
	Worst	12.4670	11.7129	10.7806	9.2588
	Std. Dev.	0.5622	0.6017	0.3368	0.0042
	t-test	2.40e−11(+)	1.15e−5(+)	4.48e−5(+)	–

5.4. Performance evaluation for the memEAPF

To evaluate the memEAPF, all the test environments of Table 1 were considered. The aim was to provide the speedup among the two different programming implementations in C++; the sequential programming and the parallel programming using the application programming interface OpenMP (open multi-processing). In the memEAPF, the execution of each elementary membrane $\mathcal{S}_i$ (line 11 of Algorithm 1) was achieved in the sequential form using one processor and in the parallel form using several processors.

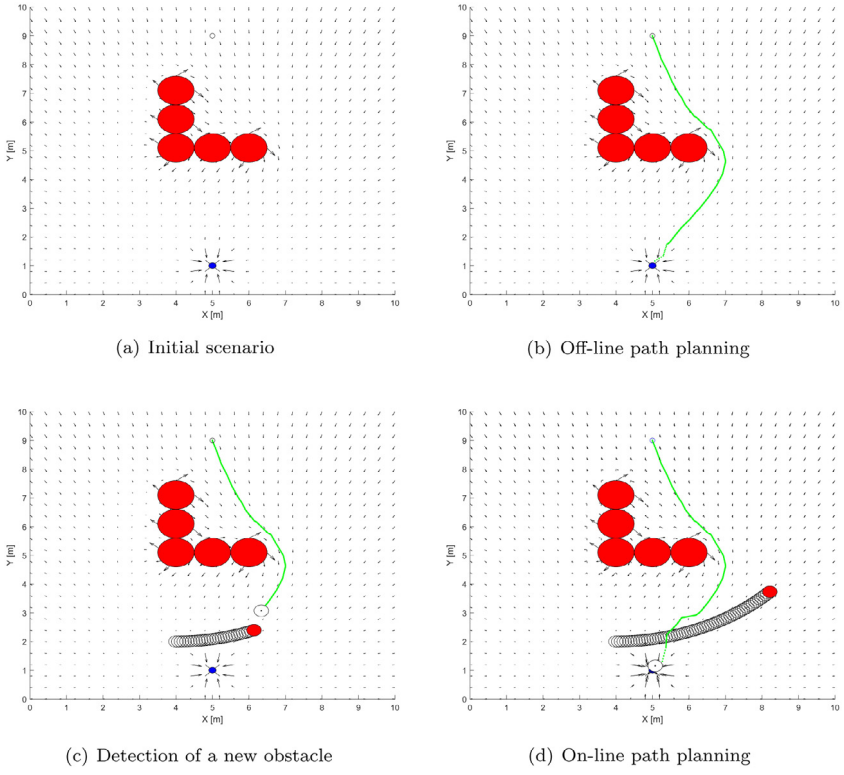
In parallel computing, speedup refers to how much a parallel algorithm is faster than its equivalent in the sequential algorithm. The speedup is defined by a relation of T_1/T_p, where T_1 is the execution time in a processor and T_p is the runtime on N_p processors. In this work, we have considered T_1 as the execution time taken by the memEAPF using the C++ sequential programming, and



(a) Initial scenario (b) Off-line path planning

(c) Detection of a new obstacle (d) On-line path planning

Fig. 6. On-line path planning with random static obstacles placement.



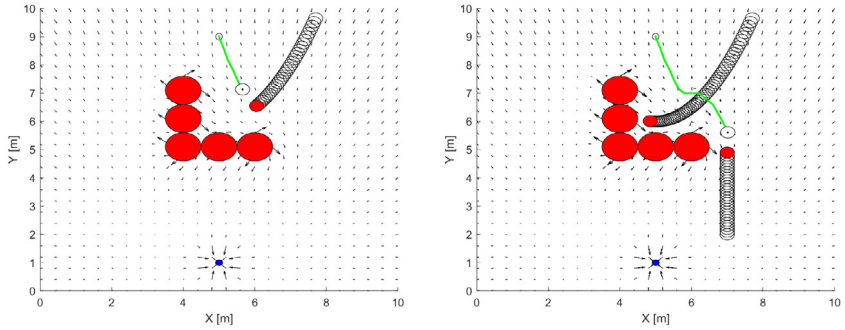
(a) Initial scenario (b) Off-line path planning

(c) Detection of a new obstacle (d) On-line path planning

Fig. 7. On-line path planning with non-static obstacles.

T_p as the execution time taken by the memEAPF in C++ parallel programming with $N_p = 4$.

For the performance evaluation, we ran the Algorithm 1 thirty times for the three different programming implementations to obtain the mean, the standard deviation, and the speedup among the different test environments. Table 5 shows a summary of the obtained results. For the sequential implementation, we used two different programming platforms, Matlab and C++, to evaluate



(a) First obstacle detected

(b) Second obstacle detected

(c) Third obstacle detected

(d) The entire path to reach the goal

Fig. 8. On-line path planning with several non-static obstacles.

Table 4
A comparison between PEAPF, PBPF, BPF, and memEAPF with $m = 16$ for the on-line implementations in static environments with random static obstacle placement, and non-static obstacles. Best, mean, worst and standard deviation represent best solution (the shortest path length from the last new starting point to the goal) over independent 30 runs, respectively. Path length units are meters. Significant difference is represented by '+'.

Environment	Statistics	PEAPF	PBPF	BPF	memEAPF
Static environment	**Best**	**3.1042**	**3.0158**	**2.9801**	**2.8947**
with random static	Mean	3.4658	3.1799	3.1130	2.9035
obstacle placement	Worst	3.6790	3.5134	3.5160	2.9102
in M04	Std. Dev.	0.4384	0.1117	0.1427	0.0038
	t-test	1.37e−7(+)	6.59e−14(+)	1.01e−8(+)	–
Non-static obstacle	**Best**	**2.6406**	**2.6103**	**2.6009**	**2.5215**
in M03	Mean	2.7703	2.6155	2.6070	2.5709
	Worst	3.8586	2.6333	2.6211	2.5907
	Std. Dev.	0.2921	0.0047	0.0057	0.0254
	t-test	9.76e−4(+)	1.76e−10(+)	1.66e−8(+)	–
Several non-static	**Best**	**2.0379**	**2.0168**	**2.0072**	**1.9579**
obstacles in M03	Mean	2.0651	2.0614	2.0508	1.9832
	Worst	2.1865	2.1685	2.1365	2.0096
	Std. Dev.	0.0298	0.0281	0.0251	0.0132
	t-test	1.10e−14(+)	2.20e−16(+)	2.19e−16(+)	–

all the test environments of Table 1 using a different number of elementary membranes ($m = 2, 4, 8$ and 16). The number of elementary membranes in the experiments was chosen according to the following criteria: The use of two elementary membranes means that at least we can have two processes running in parallel. The upper bound of 16 elementary membranes is because we can handle 16 threads in our machine without any interference among them. The number 4 and 8 is to analyze the effect of the performance and variability of solutions when we duplicate the number of elementary membranes.

The performance evaluation of programming implementations was achieved through two sets of experiments:

(1) variability analysis of the solutions when using sequential programming, where the evaluation was performed using two different programming software platforms, Matlab and C++.

(2) variability analysis of the solutions and speedup when parallel programming is used.

For the first case, the standard deviation of the execution time using Matlab fluctuate from 0.51 to 72.09, the average value is 9.71; whereas using C++, the standard deviation of the execution time varies from 0.14 to 21.08, the average value is 2.66; therefore, statistics showed that C++ implementations provided better results concerning precision and accuracy.

For the second case, the standard deviation of execution time for the parallelized C++ implementation varies from 0.09 to 4.05, the average value is 0.95. The average speedup is 3.09, and the best speedup achieved was 3.41 for the test environment M02, using the memEAPF algorithm with eight elementary membranes ($m = 8$). Since eight execution threads is the maximal number of independent processes that can handle our computer system.

Table 5
Performance results of the memEAPF on the test environments. The memEAPF was tested with $m = 2, 4, 8$, and 16 elementary membranes $\mathcal{S}_i$. Mean, and standard deviation represent the time to obtain the best solution over 30 independent runs, respectively. Time is in seconds.

Environment	m	Sequential				Parallel		
		Matlab		C++		C++		
		Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.	Speedup
M01	2	22.24	0.51	6.42	0.19	2.38	0.09	**2.70**
	4	44.45	0.99	15.54	0.99	5.19	0.17	**3.00**
	8	88.28	1.37	32.84	0.61	10.89	0.24	**3.02**
	16	176.45	1.94	68.60	1.24	22.49	0.49	**3.05**
M02	2	38.84	1.10	10.91	0.35	3.45	0.17	**3.16**
	4	78.42	2.56	24.51	1.00	7.27	0.24	**3.37**
	8	158.68	6.92	51.55	1.95	15.11	0.55	**3.41**
	16	317.45	10.47	106.53	3.39	31.52	1.19	**3.38**
M03	2	26.17	2.13	7.26	0.37	2.50	0.20	**2.90**
	4	52.03	2.07	17.55	1.41	5.59	0.38	**3.14**
	8	106.79	8.80	38.12	1.81	12.07	0.77	**3.16**
	16	219.40	15.61	82.57	7.01	25.25	2.08	**3.27**
M04	2	41.06	3.03	11.16	0.79	3.96	0.61	**2.82**
	4	82.46	6.80	26.06	2.27	8.15	0.63	**3.20**
	8	165.55	10.01	54.88	2.66	17.03	1.21	**3.22**
	16	326.37	15.50	112.18	3.82	35.85	2.31	**3.13**
M05	2	40.69	2.67	10.82	0.58	3.48	0.26	**3.11**
	4	78.83	5.19	23.39	1.02	7.09	0.57	**3.30**
	8	155.37	9.58	48.09	2.31	15.24	1.23	**3.16**
	16	321.30	21.85	100.64	5.54	31.82	2.78	**3.16**
M06	2	65.82	6.39	17.81	1.46	5.64	0.47	**3.16**
	4	134.19	11.26	37.31	3.07	11.47	0.94	**3.25**
	8	266.84	28.68	76.80	5.06	23.08	1.32	**3.33**
	16	535.37	72.09	155.01	10.45	45.85	2.72	**3.38**
M07	2	45.06	2.47	12.18	1.52	4.05	0.35	**3.01**
	4	93.23	17.46	27.04	1.75	8.93	0.83	**3.03**
	8	178.32	13.38	58.24	5.03	18.90	2.83	**3.08**
	16	370.47	40.26	118.41	5.97	38.70	3.31	**3.06**
M08	2	55.02	4.43	14.29	1.98	4.88	0.19	**2.93**
	4	111.79	7.92	31.60	3.41	10.43	0.65	**3.03**
	8	226.90	15.53	64.53	3.57	21.70	1.05	**2.97**
	16	455.93	33.08	133.16	5.86	44.38	2.20	**3.00**
M09	2	21.06	1.20	5.98	0.24	2.18	0.29	**2.74**
	4	40.41	1.38	14.58	0.88	4.75	0.33	**3.07**
	8	79.44	1.70	32.47	1.40	10.43	0.93	**3.11**
	16	159.17	2.98	66.81	2.06	20.89	0.98	**3.20**
M10	2	30.70	3.01	8.36	0.88	3.05	0.27	**2.74**
	4	59.93	10.57	18.96	1.28	6.90	0.88	**2.75**
	8	122.30	16.41	39.84	4.05	14.77	1.63	**2.70**
	16	243.68	24.97	88.16	21.08	31.25	4.05	**2.82**
M11	2	38.77	1.47	10.49	0.14	3.36	0.11	**3.12**
	4	75.98	0.84	23.17	0.33	7.06	0.25	**3.28**
	8	152.53	2.83	48.71	0.76	14.60	0.52	**3.34**
	16	304.83	10.15	100.01	2.07	29.47	0.66	**3.39**
M12	2	21.13	1.06	5.82	0.15	2.08	0.12	**2.79**
	4	41.78	1.70	14.41	0.66	4.75	0.23	**3.04**
	8	82.87	1.36	31.24	1.02	10.04	0.47	**3.11**
	16	166.51	2.27	65.70	2.08	20.60	0.88	**3.19**

In both cases, the variability of the execution time increased when using more membranes because the algorithm became more exploratory while improving the balance with the exploitation of landscapes. The best paths were obtained when the number of membranes was increased; furthermore, the variability of the solutions always remains small enough to outperform the solutions provided by the other methods.

6. Conclusions

In this paper, the membrane evolutionary artificial potential field (memEAPF) method was proposed to solve MR path planning problems for static and dynamic environments. The proposal was evaluated in off-line mode in totally known static environments, and in on-line mode in partially known dynamic environments. The memEAPF blends a GA with the APF through a membrane structure, using active membranes with membrane merger and division operations to strengthen the information communication among individuals, providing feasible and efficient paths.

An experimental simulation platform using an MR model with physical size in the plane, position, and orientation was developed to achieve the experiments. The experiments were designed to challenge the memEAPF in complex path planning scenarios. To achieve a fair comparison between the memEAPF and the other methods exhaustive statistical experiments were conducted. The path planning experiments were divided into two groups, off-line mode and on-line mode.

For the off-line mode, the memEAPF produced the best results concerning path length. The mathematical analysis showed that there exists statistical independence among the results of the experiments, demonstrating in all the cases that the memEAPF obtained the shortest paths.

For the on-line mode, the memEAPF was tested using two different set of experiments with dynamic environments: (1) with random placement of static objects. (2) using dynamic objects. For both planning modes, the memEAPF not only always found the best solutions compared to the other methods, as the statistical analysis demonstrated it, but also it provided the best solutions in less time when it was executed in the parallel mode.

The evaluation of the memEAPF demonstrates that it is a high-performance highly scalable method that can be used in sequential mode for low cost on-board computers, or in parallel mode to take advantage of novel computing architectures that provides multicore CPUs in the same board, making the memEAPF suitable for path planning problems in complex real-world scenarios. The increase of the execution time when using more membranes is attained to two main factors. The first one has to do with the fact that the algorithm becomes more exploratory, which is a desirable characteristic. The second factor is attained with the number of cores of the computer system where the algorithm is implemented, in our case we used a system with four cores with a maximum of 8 execution threads; therefore, the maximum number of membranes that we could run without time-sharing was eight. However, in general, the best paths were obtained when the number of membranes was increased, and the variability of the solutions always remains small enough to outperform the solutions provided by the other methods. In real-life systems, according to system constraints, a trade-off analysis focused to finding a balance between quality of solutions and execution time is recommendable.

As future work, it could be interesting to focus on multi-robot coordinated path planning in complicated dynamic and uncertain situations since this work only considers one MR. Another future work is considering the use of the memEAPF through the one-level membrane structure to perform multi-objective path planning considering the time of the trip and the energy employed by the MR as new objectives. Finally, the memEAPF can be extended to work in a three-dimensional space that could be beneficial to many applications for gathering information (i.e., drones), disaster relief, and exploration.

Acknowledgments

This work was supported by the Commission of Operation and Promotion of Academic Activities of the Instituto Politécnico Nacional (IPN-COFAA), and by the Mexican National Council of Science and Technology (CONACYT), Mexico.

References

[1] S.M. LaValle, Planning Algorithms, Cambridge University Press, New York, NY, USA, 2006.
[2] J. Han, Y. Seo, Mobile robot path planning with surrounding point set and path improvement, Appl. Soft Comput. 57 (2017) 35–47, http://dx.doi.org/10.1016/j.asoc.2017.03.035.
[3] J.F. Canny, The Complexity of Robot Motion Planning, The MIT Press, Cambridge, Massachusetts, 1988.
[4] Y. Wang, D. Mulvaney, I. Sillitoe, E. Swere, Robot Navigation by Waypoints, J. Intell. Robot. Syst. 52 (2) (2008) 175–207, http://dx.doi.org/10.1007/s10846-008-9209-6.
[5] A. Hidalgo-Paniagua, M.A. Vega-Rodríguez, J. Ferruz, Applying the MOVNS (multi-objective variable neighborhood search) algorithm to solve the path planning problem in mobile robotics, Expert Syst. Appl. 58 (2016) 20–35, http://dx.doi.org/10.1016/j.eswa.2016.03.035.
[6] L. Huang, H. Qu, P. Ji, X. Liu, Z. Fan, A novel coordinated path planning method using k-degree smoothing for multi-UAVs, Appl. Soft Comput. 48 (2016) 182–192, http://dx.doi.org/10.1016/j.asoc.2016.06.046.
[7] G. Zhang, M. Gheorghe, L. Pan, M.J. Pérez-Jiménez, Evolutionary membrane computing: A comprehensive survey and new results, Inform. Sci. 279 (2014) 528–551, http://dx.doi.org/10.1016/j.ins.2014.04.007.
[8] G. Zhang, J. Cheng, M. Gheorghe, Dynamic Behavior Analysis of Membrane-Inspired Evolutionary Algorithms, Int. J. Comput. Commun. Control 9 (2) (2014) 227–242, http://dx.doi.org/10.15837/ijccc.2014.2.794.
[9] G. Păun, Computing with Membranes, J. Comput. System Sci. 61 (1) (2000) 108–143, http://dx.doi.org/10.1006/jcss.1999.1693.
[10] G. Păun, G. Rozenberg, A Guide to Membrane Computing, Theoret. Comput. Sci. 287 (1) (2002) 73–100, http://dx.doi.org/10.1016/S0304-3975(02)00136-6.
[11] D.B. Fogel, An Introduction to Evolutionary Computation, in: Evolutionary Computation: The Fossil Record, Wiley-IEEE Press, 1998, pp. 1–28.
[12] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, in: Proceedings, IEEE International Conference on Robotics and Automation, vol. 2, 1985, pp. 500–505.
[13] G. Zhang, M. Gheorghe, C. Wu, A Quantum-Inspired Evolutionary Algorithm Based on P systems for Knapsack Problem, Fund. Inform. 87 (1) (2008) 93–116.
[14] T.Y. Nishida, Membrane algorithms: Approximate algorithms for NP-complete optimization problems, in: Applications of Membrane Computing, Springer, Berlin, Heidelberg, 2006, pp. 303–314.
[15] E. Masehian, D. Sedighizadeh, Classic and Heuristic Approaches in Robot Motion Planning A Chronological Review, Int. J. Mech. Aerosp. Ind. Mechatron. Manuf. Eng. 1 (5) (2007) 228–233.
[16] P. Bhattacharya, M.L. Gavrilova, Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path, IEEE Robot. Autom. Mag. 15 (2) (2008) 58–66, http://dx.doi.org/10.1109/MRA.2008.921540.
[17] B. Oommen, S. Iyengar, N. Rao, R. Kashyap, Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case, IEEE J. Robot. Autom. 3 (6) (1987) 672–681, http://dx.doi.org/10.1109/JRA.1987.1087133.
[18] B. Faverjon, P. Tournassoud, A local based approach for path planning of manipulators with a high number of degrees of freedom, in: Proceedings, IEEE International Conference on Robotics and Automation, Vol. 4, 1987, pp. 1152–1159.
[19] A. Blake, R. Cipolla, R. Curwen, Z. Xie, A. Zisserman, Visual guidance for robot motion, in: IEEE Colloquium on Active and Passive Techniques for 3-D Vision, 1991, pp. 7/1–7/3.
[20] C. Cai, S. Ferrari, Information-Driven Sensor Path Planning by Approximate Cell Decomposition, IEEE Trans. Syst. Man Cybern. B 39 (3) (2009) 672–689, http://dx.doi.org/10.1109/TSMCB.2008.2008561.
[21] L.E. Kavraki, P. Svestka, J.C. Latombe, M.H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. 12 (4) (1996) 566–580, http://dx.doi.org/10.1109/70.508439.
[22] T. Ju, S. Liu, J. Yang, S. D., Rapidly Exploring Random Tree Algorithm-Based Path Planning for Robot-Aided Optical Manipulation of Biological Cells, IEEE Trans. Autom. Sci. Eng. 11 (3) (2014) 649–657, http://dx.doi.org/10.1109/TASE.2013.2289311.
[23] T. Lolla, M.P. Ueckermann, K. Yiğit, P.J. Haley, P.F.J. Lermusiaux, Path planning in time dependent flow fields using level set methods, in: IEEE International Conference on Robotics and Automation (ICRA), 2012, pp. 166–173.
[24] B. Stilman, Network Languages for Complex Systems, Comput. Math. Appl. 26 (8) (1993) 51–79, http://dx.doi.org/10.1016/0898-1221(93)90331-O.
[25] C.C. Tsai, H.C. Huang, C.K. Chan, Parallel Elite Genetic Algorithm and Its Application to Global Path Planning for Autonomous Robot Navigation, IEEE Trans. Ind. Electron. 58 (10) (2011) 4813–4821, http://dx.doi.org/10.1109/TIE.2011.2109332.
[26] S. Binitha, S.S. Sathya, A Survey of Bio inspired Optimization Algorithms, Int. J. Soft Comput. Eng. 2 (2) (2012) 137–151.
[27] M.A. Porta Garcia, O. Montiel, O. Castillo, R. Sepúlveda, P. Melin, Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation, Appl. Soft Comput. 9 (3) (2009) 1102–1110, http://dx.doi.org/10.1016/j.asoc.2009.02.014.
[28] V. Roberge, M. Tarbouchi, G. Labonte, Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning, IEEE Trans. Ind. Inf. 9 (1) (2013) 132–141, http://dx.doi.org/10.1109/TII.2012.2198665.
[29] M.A. Contreras-Cruz, V. Ayala-Ramirez, U.H. Hernandez-Belmonte, Mobile robot path planning using artificial bee colony and evolutionary programming, Appl. Soft Comput. 30 (2015) 319–328, http://dx.doi.org/10.1016/j.asoc.2015.01.067.
[30] L. Amador-Angulo, O. Mendoza, J.R. Castro, A. Rodriguez-Diaz, P. Melin, O. Castillo, Fuzzy Sets in Dynamic Adaptation of Parameters of a Bee Colony Optimization for Controlling the Trajectory of an Autonomous Mobile Robot, Sensors 16 (9) (2016) 1–27, http://dx.doi.org/10.3390/s16091458.
[31] L. Amador-Angulo, O. Castillo, J. Castro, A Generalized Type-2 Fuzzy Logic System for the dynamic adaptation the parameters in a Bee Colony Optimization algorithm applied in an autonomous mobile robot control, in: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2016, pp. 537–544, http://dx.doi.org/10.1109/FUZZ-IEEE.2016.7737733.
[32] J.H. Kim, J.H. Han, Y.H. Kim, S.H. Choi, E.S. Kim, Preference-Based Solution Selection Algorithm for Evolutionary Multiobjective Optimization, IEEE Trans. Evol. Comput. 16 (1) (2012) 20–34, http://dx.doi.org/10.1109/TEVC.2010.2098412.

[33] J. Conesa-Muñoz, A. Ribeiro, D. Andujar, C. Fernandez-Quintanilla, J. Dorado, Multi-path planning based on a NSGA-II for a fleet of robots to work on agricultural tasks, in: IEEE Congress on Evolutionary Computation, 2012, pp. 1–8.
[34] X. Wang, G. Zhang, J. Zhao, H. Rong, F. Ipate, R. Lefticaru, A Modified Membrane-Inspired Algorithm Based on Particle Swarm Optimization for Mobile Robot Path Planning, Int. J. Comput. Commun. Control 10 (5) (2015) 732–745, http://dx.doi.org/10.15837/ijccc.2015.5.2030.
[35] A.A. Khaliq, A. Saffiotti, Stigmergy at work: Planning and navigation for a service robot on an RFID floor, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015 pp. 1085–1092.
[36] X. Yang, M. Moallem, R.V. Patel, A layered goal-oriented fuzzy motion planning strategy for mobile robot navigation, IEEE Trans. Syst. Man Cybern. B 35 (6) (2005) 1214–1224, http://dx.doi.org/10.1109/TSMCB.2005.850177.
[37] D.K. Pai, L.M. Reissell, Multiresolution rough terrain motion planning, IEEE Trans. Robot. Autom. 14 (1) (1998) 19–33, http://dx.doi.org/10.1109/70.660835.
[38] E. Masehian, M.R. Amin-Naseri, Sensor-Based Robot Motion Planning - A Tabu Search Approach, IEEE Robot. Autom. Mag. 15 (2) (2008) 48–57, http://dx.doi.org/10.1109/MRA.2008.921543.
[39] M. Couillard, J. Fawcett, M. Davison, Optimizing Constrained Search Patterns for Remote Mine-Hunting Vehicles, IEEE J. Ocean. Eng. 37 (1) (2012) 75–84, http://dx.doi.org/10.1109/JOE.2011.2173833.
[40] O. Castillo, L. Amador-Angulo, J.R. Castro, M. Garcia-Valdez, A comparative study of type-1 fuzzy logic systems, interval type-2 fuzzy logic systems and generalized type-2 fuzzy logic systems in control problems, Inform. Sci. 354 (2016) 257–274, http://dx.doi.org/10.1016/j.ins.2016.03.026.
[41] M.A. Sanchez, O. Castillo, J.R. Castro, Generalized Type-2 Fuzzy Systems for controlling a mobile robot and a performance comparison with Interval Type-2 and Type-1 Fuzzy Systems, Expert Syst. Appl. 42 (14) (2015) 5904–5914, http://dx.doi.org/10.1016/j.eswa.2015.03.024.
[42] X. Wang, G. Zhang, F. Neri, T. Jiang, J. Zhao, M. Gheorghe, F. Ipate, R. Lefticaru, Design and implementation of membrane controllers for trajectory tracking of nonholonomic wheeled mobile robots, Integr. Comput.-Aided Eng. 23 (1) (2016) 15–30, http://dx.doi.org/10.3233/ICA-150503.
[43] O. Montiel, U. Orozco-Rosas, R. Sepúlveda, Path planning for mobile robots using Bacterial Potential Field for avoiding static and dynamic obstacles, Expert Syst. Appl. 42 (12) (2015) 5177–5191, http://dx.doi.org/10.1016/j.eswa.2015.02.033.
[44] D. Devaurs, T. Siméon, J. Cortés, Optimal Path Planning in Complex Cost Spaces With Sampling-Based Algorithms, IEEE Trans. Autom. Sci. Eng. 13 (2) (2016) 415–424, http://dx.doi.org/10.1109/TASE.2015.2487881.
[45] R. Siegwart, I.R. Nourbakhsh, D. Scaramuzza, Introduction to Autonomous Mobile Robots, second ed., The MIT Press, Cambridge, Mass., 2011.
[46] M. Davoodi, F. Panahi, A. Mohades, S.N. Hashemi, Clear and smooth path planning, Appl. Soft Comput. 32 (2015) 568–579, http://dx.doi.org/10.1016/j.asoc.2015.04.017.
[47] G. Zhang, J. Cheng, M. Gheorghe, Q. Meng, A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems, Appl. Soft Comput. 13 (3) (2013) 1528–1542, http://dx.doi.org/10.1016/j.asoc.2012.05.032.
[48] C. Liu, G. Zhang, H. Liu, M. Gheorghe, F. Ipate, An improved membrane algorithm for solving time- frequency atom decomposition, in: Membrane Computing: 10th International Workshop, WMC 2009, Curtea de Arges, Romania, August 24–27, 2009. Revised Selected and Invited Papers, Springer, Berlin, Heidelberg, 2010, pp. 371–384.
[49] G. Zhang, J. Cheng, M. Gheorghe, F. Ipate, X. Wang, QEAM: An Approximate Algorithm Using P Systems with Active Membranes, Int. J. Comput. Commun. Control 10 (2) (2015) 263–279, http://dx.doi.org/10.15837/ijccc.2015.2.1757.
[50] G. Păun, A quick introduction to membrane computing, J. Log. Algebr. Program. 79 (6) (2010) 291–294, http://dx.doi.org/10.1016/j.jlap.2010.04.002.
[51] O. Montiel, R. Sepúlveda, U. Orozco-Rosas, Optimal Path Planning Generation for Mobile Robots using Parallel Evolutionary Artificial Potential Field, J. Intell. Robot. Syst. 79 (2) (2015) 237–257, http://dx.doi.org/10.1007/s10846-014-0124-8.
[52] G. Zhang, M.J. Pérez-Jiménez, M. Gheorghe, Real-life Applications with Membrane Computing, Springer, ECC, volume 25, 2017, http://dx.doi.org/10.1007/978-3-319-55989-6.
[53] S.S. Ge, Y.J. Cui, New potential functions for mobile robot path planning, IEEE Trans. Robot. Autom. 16 (5) (2000) 615–620, http://dx.doi.org/10.1109/70.880813.
[54] Y. Koren, J. Borenstein, Potential field methods and their inherent limitations for mobile robot navigation, in: Proceedings, 1991 IEEE International Conference on Robotics and Automation, vol. 2, 1991, pp. 1398–1404.
[55] U. Orozco-Rosas, O. Montiel, R. Sepúlveda, Pseudo-bacterial Potential Field Based Path Planner for Autonomous Mobile Robot Navigation, Int. J. Adv. Robot. Syst. 12 (7) (2015) 81, http://dx.doi.org/10.5772/60715.
[56] M.P. Aghababa, 3D path planning for underwater vehicles using five evolutionary optimization algorithms avoiding static and energetic obstacles, Appl. Ocean Res. 38 (2012) 48–62, http://dx.doi.org/10.1016/j.apor.2012.06.002.
[57] J. Botzheim, Y. Toda, N. Kubota, Bacterial memetic algorithm for offline path planning of mobile robots, Memetic Comput. 4 (1) (2012) 73–86, http://dx.doi.org/10.1007/s12293-012-0076-0.
[58] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization, J. Heuristics 15 (6) (2008) 617–644, http://dx.doi.org/10.1007/s10732-008-9080-4.
[59] N. Morales, J. Toledo, L. Acosta, Path planning using a Multiclass Support Vector Machine, Appl. Soft Comput. 43 (2016) 498–509, http://dx.doi.org/10.1016/j.asoc.2016.02.037.