Received September 18, 2019, accepted October 19, 2019, date of publication October 28, 2019, date of current version November 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2949835

Hybrid Path Planning Algorithm Based on Membrane Pseudo-Bacterial Potential Field for Autonomous Mobile Robots

ULISES OROZCO-ROSAS[1], **KENIA PICOS**[1], **AND OSCAR MONTIEL**[2]
[1]CETYS Universidad, Centro de Innovación y Diseño (CEID), Tijuana 22210, México
[2]Instituto Politécnico Nacional, CITEDI-IPN, Tijuana 22435, México

Corresponding author: Ulises Orozco-Rosas (ulises.orozco@cetys.mx)

This work was supported in part by the Coordinación Institucional de Investigación of CETYS Universidad, and in part by the Consejo Nacional de Ciencia y Tecnología (CONACYT).

ABSTRACT A hybrid path planning algorithm based on membrane pseudo-bacterial potential field (MemPBPF) is proposed. Membrane-inspired algorithms can reach an evolutionary behavior based on biochemical processes to find the best parameters for generating a feasible and safe path. The proposed MemPBPF algorithm uses a combination of the structure and rules of membrane computing. In that sense, the proposed MemPBPF algorithm contains dynamic membranes that include a pseudo-bacterial genetic algorithm for evolving the required parameters in the artificial potential field method. This hybridization between membrane computing, the pseudo-bacterial genetic algorithm, and the artificial potential field method provides an outperforming path planning algorithm for autonomous mobile robots. Computer simulation results demonstrate the effectiveness of the proposed MemPBPF algorithm in terms of path length considering collision avoidance and smoothness. Comparisons with two different versions employing a different number of elementary membranes and with other artificial potential field based algorithms are presented. The proposed MemPBPF algorithm yields improved performance in terms of time execution by using a parallel implementation on a multi-core computer. Therefore, the MemPBPF algorithm achieves high performance yielding competitive results for autonomous mobile robot navigation in complex and real scenarios.

INDEX TERMS Artificial potential field, autonomous mobile robots, membrane computing, path planning, pseudo-bacterial genetic algorithm.

I. INTRODUCTION

Autonomous mobile robots are automatic systems with locomotion ability [1]; the level of autonomy can be determined using a rank. The lower rank corresponds to teleoperated robots where humans decide and perform all the control activities. The highest rank of autonomy is for those robots that decide all the aspects of their locomotion without human intervention [2].

An autonomous mobile robot (AMR) needs to overcome many challenges to become autonomous; for example, in its essential task of moving from one point to another one [3], [4], an AMR needs to have efficient sensing system capable of determining routes while avoiding obstacles and building or updating its environment map. To move in the desired direction, the AMR needs to select an appropriated sequence of actions; i.e., it needs to plan the route. This task is very complex for many reasons. In real-world scenarios, considering an ideal situation with a known environment, workable maps for real applications are big; this makes the task of calculating the route extremely hard because the time of problems yields to deal with NP-hard computational complexities. In non-ideal situations, complexity increases since not only the path planner needs to calculate feasible routes but also needs to fulfill hard real-time constrains to navigate safely; missing deadlines may produce disastrous consequences. Onboard computers of AMRs should perform several tasks without missing deadlines.

Path planning can be applied only when the environment map is known; hence, AMRs should be capable of performing simultaneous localization and mapping (SLAM) [5]. Both tasks can become a bottleneck, and the reduction of computational or time complexities is very important.

The associate editor coordinating the review of this manuscript and approving it for publication was Yongming Li.

This proposal named MemPBPF contributes to state-of-the-art with a highly efficient path planning method for AMRs with the following three characteristics: 1) It reduces time complexity by integrating membrane computing [6], the pseudo-bacterial genetic algorithm [7], and the artificial potential field method [8]. 2) Better feasible solutions considering minimum path length, collision avoidance, and path smoothness are achieved. 3) The method is highly scalable and provides a solution that will improve with technological advances; for example, if it is implemented on graphics processing units (GPUs), and because this technology is advancing very fast, providing more processing units, memory, and processing speed due to architecture improvements, it is expected that with every advance a considerable speedup can be achieved.

This work introduces the MemPBPF algorithm as a computational framework that deals with two different goals that complement each other, path planning and trajectory planning [9]. Path planning is in charge of generating a continuous set of feasible positions called the path on the environment that can drive the AMR from its actual position to a target position. Trajectory planning focuses on solving how to move the AMR through the feasible already calculated path, considering the AMR type.

The remainder of this paper contains the next topics. Section II presents a basic review of the theoretical fundamentals of the main components of the MemPBPF algorithm: path planning, artificial potential fields, pseudo-bacterial genetic algorithms, and membrane computing. Section III explains the proposed MemPBPF algorithm for path planning in AMRs. Section IV describes the test specifications, MemPBPF path planning results considering static and dynamic obstacles, computational performance results, and the robotic platform implementation. Section V summarize some important conclusions of this work.

II. FUNDAMENTALS

In this section, we review the fundamentals of the proposed MemPBPF algorithm for path planning in AMRs. We start with a general description of the path-planning problem. Next, we explain in detail the artificial potential field method for path planning in AMRs. Then, we review the pseudo-bacterial genetic algorithm. Last, we conclude this section with the fundamentals of membrane computing and its integration in the scheme of the proposed MemPBPF algorithm.

A. PATH PLANNING PROBLEM

The path-planning problem essentially involves finding a collision-free path concerning a given start and target position, for an AMR subject to several constraints. Under this general definition, the simplified path planning problem formulation is presented as follows. The workspace Q is comprehended as an environment of two dimensions. Which contains several obstacles $O_j = [O_0, O_1, \ldots, O_n]$, where n denotes the number of obstacles presented in the environment. There is

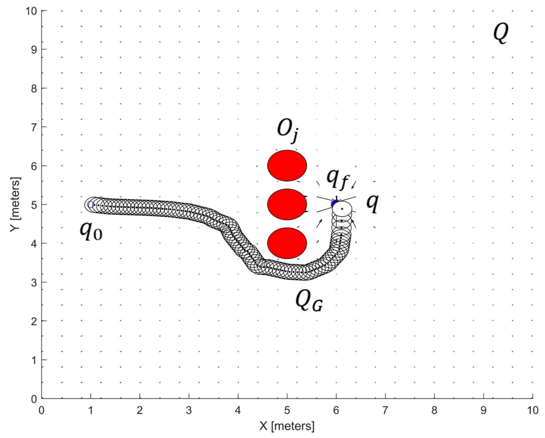


FIGURE 1. Path planning problem representation.

a wide variety of approaches and techniques for geometric modeling of the environment, and the choice usually depends on the application and the difficulty of the problem [3]. In this work, we have employed a circular representation of the obstacles due to its practical implementation and high compatibility with the artificial potential field method. Moreover, the circular representation can enclose and form any kind of obstacle that the AMR can expect to find. The AMR position or configuration is denoted by $q = (x, y, \theta)$ [10]. The Cartesian coordinates x and y represent the center point of the AMR. The term θ denotes the orientation, i.e., θ indicates the angular difference between the environment and the AMR reference frame [11]. The main path planning objective is to generate a viable sequence of positions $Q_G = [q_0, q_1, \ldots, q_f]$ to drive the AMR safely from a given start position q_0 to a target position q_f with the minimum path length [3]. Fig. 1 depicts these concepts.

In the literature, there are several proposals to address path-planning problems. Hence, the existing research can be divided into two wide categories: classical methods and heuristic-based algorithms [12]. Classical methods consist of cell decomposition [13], roadmap [14], artificial potential field method [8], and mathematical programming [15]. On the other side, heuristic-based algorithms consist of single or multiple objective bio-inspired algorithms [16], probabilistic methods [17], adaptive neuro-fuzzy inference systems [18], among others. The literature indicates that heuristic-based algorithms can yield good results, but not necessarily the optimum [19].

The classical methods and heuristic-based algorithms have their strong points and disadvantages. In some cases, they have a strong relation between them, and sometimes they are joint to develop an effective and efficient AMR path planning algorithm [15]. In this regard, the proposed MemPBPF algorithm combines the evolution rules and the framework of membrane computing with dynamic membranes that contain a pseudo-bacterial genetic algorithm to evolve the gain parameters employed by the artificial potential field method to obtain viable paths taking into account minimum length, collision avoidance, and smoothness for an efficient and effective AMR navigation.

B. ARTIFICIAL POTENTIAL FIELD METHOD

The artificial potential field approach was introduced by Khatib [8], where for a configured space, the AMR is presented as a particle under the effect of an artificial potential field, whose local variation returns the free space structure. The core idea of the artificial potential field method is to create an attractive potential field around the target position, as well as to create a repulsive potential field force around the obstacles [20]. Therefore, the artificial potential field method employs attractive and repulsive components to drive the AMR to its target while avoiding collisions with the obstacles.

The total artificial potential field denoted by $U(q)$ and described by (1) is composed by two potential functions, the attractive potential $U_a(q)$, and the repulsive potential $U_r(q)$. Therefore, the artificial potential field $U(q)$ is the superposition of these two functions.

$$U(q) = U_a(q) + U_r(q) \tag{1}$$

The attractive potential function $U_a(q)$ is described by (2). Where q represents the current AMR configuration, q_f denotes the target position, and k_a is a positive scalar that denotes the attractive proportional gain of the potential function.

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

The repulsive potential function $U_r(q)$ is described by (3). Where ρ_0 denotes the limit distance of effect of the repulsive potential field, and ρ denotes the Euclidean distance from the AMR configuration to the nearest obstacle. The selection of the value of ρ_0 is subject to the AMR maximum speed. In (3), k_r is a positive scalar that denotes the repulsive proportional gain of the potential function. Therefore, the repulsive function has a limited effect, and it prevents the movement of the AMR will be affected by far obstacles. [11].

$$U_r(q) = \begin{cases} \frac{1}{2}k_r(\frac{1}{\rho} - \frac{1}{\rho_0})^2 & \text{if } \rho \leq \rho_0 \\ 0 & \text{if } \rho > \rho_0 \end{cases} \tag{3}$$

The total potential field force $F(q)$ is described by (4). Hence, $F(q)$ is obtained by the negative gradient of the total artificial potential field $U(q)$.

$$F(q) = -\nabla U(q) \tag{4}$$

In this context, now we present some recent work where the artificial potential field approach was employed for path planning in AMRs. Bayat et al. assigned a potential function for each obstacle in the environment, the interaction of all dispersed obstacles is incorporated into a potential surface which is subject to the physical features of the AMR and the obstacles [21]. In that work, the optimal path is found from a cost function optimization by achieving a trade-off between pass through the shortest path and avoiding collisions. In another example, Kovács et al. extended the artificial potential field approach using motion characteristics, including velocity and orientation information for path planning of AMRs in dynamic household environments [22]. Lastly, we can mention the work proposed by Rostami et al., where a modified artificial potential field method is employed for path planning and collision avoidance for AMRs in environments with fixed obstacles [23].

The artificial potential field method is broadly employed in path planning for AMRs due to its mathematical elegance, simplicity, and effectiveness to provide smooth and safe paths. On the other hand, it presents some disadvantages like the local minima problem [24], or the goal non-reachable with obstacles nearby [25], among others. For this reason, the artificial potential field approach is enhanced with several techniques to overcome or mitigate these disadvantages. In this work, the proposed MemPBPF algorithm combines the evolution rules and the framework of membrane computing with dynamic membranes that contain a pseudo-bacterial genetic algorithm. With the main purpose of evolving the attractive and repulsive proportional gain parameters, k_a and k_r, respectively, and the step size η required by the artificial potential field method to perform the path planning in AMRs.

C. PSEUDO-BACTERIAL GENETIC ALGORITHM

Nawa et al. presented a new kind of evolutionary algorithm named pseudo-bacterial genetic algorithm [26], which was applied for fuzzy rule base extraction from input-output data sets [27]. The pseudo-bacterial genetic algorithm presented a new genetic operator named bacterial mutation that has proved to be suitable in environments with a weak relationship between the parameters of a system [7].

Essentially, the pseudo-bacterial genetic algorithm is a global optimization approach derived from genetic algorithms [28]. Therefore, the pseudo-bacterial genetic algorithm has most of the advantages of genetic algorithms and contains other structures that serve to obtain better results. The fundamental of the pseudo-bacterial genetic algorithm holds the bacterium, which can transmit a duplicate of a gene from a host cell and insert it into an infected cell. Through this procedure, named bacterial mutation, the features of a single bacterium can spread to the population. In that sense, this procedure emulates the process of microbial evolution [7].

The pseudo-bacterial genetic algorithm employs the bacterial mutation process as follows: 1) In the beginning, N_c clones or copies of each chromosome are formed. 2) Next, a randomly selected segment with a determined length suffers a mutation in each clone. However, the first clone is preserved without the mutation. 3) Subsequently to the mutation, the clones are evaluated employing a fitness function. The clone that presents the best fitness value is selected to transfer the undergone mutation segment to the rest of the clones. The three steps, the mutation of the clones, the selection of the best clone, and the transfer of the mutated segment are repeated until each segment of the chromosome has been mutated once [29]. Finally, the best chromosome is preserved, and all the clones are removed. The entire bacterial mutation process is performed for all the chromosomes in the population [24].

The pseudo-bacterial genetic algorithm is a practical evolutionary algorithm that offers fast convergence and improvement in the results [30], without detrimental effects in the exploration. In the most elementary form, the pseudo-bacterial genetic algorithm is modeled for computer simulation employing the difference equation presented in (5). Where, t represents the generation counter, and $P(t+1)$ is the new population of chromosomes attained from the current population of chromosomes $P(t)$ after being improved by random variation v and selection s [31].

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

Related work in the area of path planning for AMRs employing the pseudo-bacterial genetic algorithm can be found in [32], where a bacterial foraging method is presented for path planning in robotics. In that work, the bacterial chemotaxis process enables the AMR to discover the environment for locating the target position without hitting any obstacle. In [24], a pseudo-bacterial genetic algorithm employed as a global optimization method and a fitness function based on the artificial potential field method are employed to generate feasible paths in AMR navigation. In [33] is presented a high-performance path planning algorithm based on a parallel pseudo-bacterial potential field and implemented on a GPU as an improvement to accelerate the path planning computation for AMR navigation.

D. MEMBRANE COMPUTING

Membrane computing belongs to natural computing, and it was introduced by Gheorghe Păun in 1998 [6]. The core purpose of membrane computing is to abstract parallel and distributed computing models, also called membrane systems or P systems, from the compartmentalized structure and interactions of living cells [34]. In that sense, the obtained models are parallel and distributed schemes that evolve through rules and process the multisets of objects into the sections that are hierarchically established [35].

Membrane computing is a variation on the P system model. P systems consist of membrane structures that contain objects into their membranes; the membranes have evolution rules like communication and transformation to divide and merge membranes [6]. The classification for the P systems consists of three types. The first type named as cell-like P systems; this type contains one membrane cell. The second type, called tissue-like P systems, consists of several one membrane cells in a common environment. The third type, known as neural-like P systems considers neurons as their cell [36]. In this work, a cell-like P system is the main component of the proposed MemPBPF algorithm because it is simple and practical to implement with parallel computing. The cell-like P system contains a hierarchical structure of membranes, the type of rules employed are communication and transformation, and it presents inherent parallelism. These strengths and features are attractive and desirable for modeling complex problems considering a computational point of view [37].

The proposed MemPBPF algorithm for path planning in AMRs consists of a cell-like P system that evolves the set of parameters $[k_a, k_r, \eta]$ required for the artificial potential field method. Therefore, the MemPBPF algorithm uses an active structure through dynamic membranes, in concrete terms, a one-level membrane structure [38] with rules, such as membrane division and merger [39]. The membrane division rules are helpful to increase the exploration ability [40], and the membrane merger rules are useful to improve the information communication among chromosomes, in specific the set of parameters $[k_a, k_r, \eta]$.

The membrane structure for the proposed MemPBPF algorithm for path planning in AMRs is denoted by $\mu = [_0[_1]_1, [_2]_2, \cdots, [_m]_m]_0$. Hence, the membrane structure μ 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. Therefore, for the proposed MemPBPF algorithm, we have a membrane structure of degree $m+1$ whereas the height of the tree associated is the *depth* [41]. In that sense, we have a membrane structure μ of depth 2 for the MemPBPF algorithm.

The one-level membrane structure μ for the proposed MemPBPF algorithm entails delimited sections to evolve the multisets of objects. In membrane computing, the multisets are formed with several types of objects. These objects not only are represented by letters from a particular alphabet, as in the earliest classes of P systems. Consequently, the objects can be represented by strings, or by complex data structures, e.g., arrays [42]. In the proposed MemPBPF algorithm, every single elementary membrane $\mathcal{S}_i$ has arrays of multisets of objects containing sets of parameters codified in the form $[k_a, k_r, \eta]$, and evolution rules that consist of the communication, divide, and merge phases.

III. MEMBRANE PSEUDO-BACTERIAL POTENTIAL FIELD

The proposed MemPBPF algorithm combines a cell-like P system, a pseudo-bacterial genetic algorithm, and the artificial potential field (APF) method to perform the AMR path planning. The objective of this section is to explain in detail the proposed MemPBPF algorithm. Hence, firstly the MemPBPF pseudocode (Algorithm 1) is explained, and then, an explanation of the APF pseudocode (Algorithm 2) is provided. In this work, the Algorithm 2 serves as a fitness function and as a local planning method for the MemPBPF algorithm. All the experiments were achieved using the Algorithm 1.

A. MemPBPF ALGORITHM

Algorithm 1 presents the MemPBPF pseudocode for AMR path planning. The MemPBPF algorithm employs the next input parameters: the AMR's start position q_0, the target

Algorithm 1 MemPBPF

```
    Input: start position $q_0$, target position $q_f$, and
           environment information $O_j$
    Output: set of configurations $Q_G$, path length $C$,
            number of configurations $\tau$, and flag $target$
1   $APF \leftarrow$ @ APF$(q_0, q_f, O_j, k_a, k_r, \eta)$
2   $t_0 \leftarrow 0$
3   set $N_g$, $m$, and $\ell$
4   initialize $P_0(t_0)$
5   initialize $\mathcal{S}_0$ and $\mathcal{S}_i$, $1 \leq i \leq m$
6   initialize $PBPF_{par}$
7   while $t_0 < N_g$ do
8       for each elementary membrane $\mathcal{S}_i$ in parallel do
9           $t_i \leftarrow 0$
10          evaluate each chromosome in $P_i(t_i)$ using $APF$
11          while not termination do
12              copy the best chromosomes $R \leftarrow P_i(t_i)$
13              select parents $P' \leftarrow P_i(t_i)$
14              perform crossover $P'$
15              perform bacterial mutation $P'$
16              evaluate each chromosome in $P'$ using $APF$
17              $P_i(t_i + 1) \leftarrow P' \cup R$
18              $t_i \leftarrow t_i + 1$
19          end
20      end
21      merge all the elementary membranes $\mathcal{S}_i$, into $\mathcal{S}_F$
22      apply communication rules and send out
        $[k_a, k_r, \eta]_{opt}$
23      divide $\mathcal{S}_F$ in $m$ elementary membranes $\mathcal{S}_i$
24      $t_0 \leftarrow t_0 + 1$
25  end
26  $[Q_G, C, \tau, target] \leftarrow APF$ using $[k_a, k_r, \eta]_{opt}$
27  return $[Q_G, C, \tau, target]$
```

Algorithm 2 APF

```
    Input: start position $q_0$, target position $q_f$, environment
           information $O_j$, proportional gains $k_a$ and $k_r$,
           and step size $\eta$
    Output: set of configurations $Q_G$, path length $C$,
            number of configurations $\tau$, and flag $target$
1   set $N$ and $r$
2   $\tau \leftarrow 0$
3   $C \leftarrow 0$
4   $safe \leftarrow$ True
5   $G \leftarrow \|q_f - (q_\tau \leftarrow q_0)\|$
6   while $G > 0$ and $\tau < N$ and $safe$ do
7       $U(q_\tau) \leftarrow \frac{1}{2}\left[k_a(q_\tau - q_f)^2 + \sum_{j=1}^{n} k_r\left(\frac{1}{\rho} - \frac{1}{\rho_0}\right)^2\right]$
8       $F(q_\tau) \leftarrow -\nabla U(q_\tau)$
9       $q_{\tau+1} \leftarrow q_\tau + \eta * F(q_\tau) / \parallel F(q_\tau) \parallel$
10      $G \leftarrow \|q_f - q_{\tau+1}\|$
11      $C \leftarrow C + \parallel q_{\tau+1} - q_\tau \parallel$
12      if $\rho \leq r + r_o$ then
13          $safe \leftarrow$ False
14      end
15      $\tau \leftarrow \tau + 1$
16  end
17  if $q_\tau$ is equal to $q_f$ then
18      $target \leftarrow$ True
19      $Q_G \leftarrow [q_0, q_1, \ldots, q_\tau]$
20  else
21      $target \leftarrow$ False
22      $C \leftarrow \infty$
23  end
24  return $[Q_G, C, \tau, target]$
```

position q_f that the AMR must attain, and the environment information O_j which contains the obstacles position and its corresponding radius in (x, y, r_o) format. The main objective of the proposed MemPBPF algorithm is to obtain a sequence of objective positions $Q_G = [q_0, q_1, \ldots, q_f]$, i.e., a collision-free path that will drive the AMR to reach the target position with the minimum path length. Therefore, Algorithm 1 performs the path planning generation, with the distinctive attribute that provides an optimal or nearly optimal set of configurations Q_G. Other parameters that Algorithm 1 returns are the path length C, the number of configurations τ, and a Boolean parameter *target* that indicates if the target position was achieved.

Algorithm 1 employs in line 1 a function handler *APF* (artificial potential field method) to facilitate the pseudocode writing, it is equivalent to write the APF pseudocode (Algorithm 2), including all its parameters. In line 2 and 3, a generation counter t_0 is initialized to zero and the parameters: number of generations N_g, number of elementary membranes m and subpopulation size ℓ are established, respectively. In line 4, an initial random population $P_0 = \{P_1, P_2, \ldots, P_m\}$ of chromosomes is created. Where, P_i is a subpopulation, $P_i = \{p_1, \ldots, p_\ell\}$, $1 \leq i \leq m$, and $p_j = [k_a, k_r, \eta]$ is a chromosome codified with the information of the proportional gains k_a and k_r and the step size η, i.e., a potential solution for the path planning problem presented. Therefore, the population will be divided into m subpopulations P_i that will be distributed among the elementary membranes $\mathcal{S}_i$.

In line 5, the Algorithm 1 initializes the membrane structure $\mu = [_0[_1]_1, [_2]_2, \ldots, [_m]_m]_0$. Where, μ represents the membrane structure that is composed of an outer membrane or skin membrane denoted by $\mathcal{S}_0$ and m number of inner membranes or elementary membranes denoted by $\mathcal{S}_i$. Therefore, the MemPBPF algorithm employs the hierarchical structure of a cell-like P system (formally, a rooted tree), in specific a one-level membrane structure μ expressed in the form $[_0[_1]_1, [_2]_2, \ldots, [_m]_m]_0$ with m number of elementary membranes contained in the skin membrane, denoted by the subindex 0. Moreover, each elementary membrane $\mathcal{S}_i$

will be initialized with a subpopulation P_i composed of ℓ chromosomes.

In line 6, the data structure $PBPF_{par}$ is initialized, which is a group of data elements or members of a different type. The $PBPF_{par}$ contains the selection rate (floating point), the crossover type (integer), the bacterial mutation rate (floating point), and the number of clones (integer) per chromosome.

From line 7 to 25, there is the core iterative process of the proposed MemPBPF algorithm to find the global best chromosome $[k_a, k_r, \eta]_{opt}$ through the evolution of the population. Then, from line 8 to 20, each elementary membrane $\mathcal{S}_i$ evolves its corresponding subpopulation of chromosomes through a pseudo-bacterial genetic algorithm indicated from line 11 to 19 of Algorithm 1. For the sequential implementation of the MemPBPF algorithm, the evolution of each elementary membrane (loop from line 8 to 20) is executed in sequential form, i.e., first, the elementary membrane $\mathcal{S}_1$ is evolved, after that, the elementary membrane $\mathcal{S}_2$ is evolved and so on until the last elementary membrane $\mathcal{S}_m$. On the other hand, for the parallel implementation of the MemPBPF algorithm, the evolution of the elementary membranes is performed in a concurrent form regarding the number of available cores in the CPU. Before the pseudo-bacterial genetic algorithm process, in line 10, each subpopulation P_i is evaluated using the APF function (see Algorithm 2) to know the path length C, i.e., the fitness value of each chromosome.

In line 12, due to the above evaluation mentioned. The subpopulation P_i is sorted from the best chromosome (with the strongest fitness value that is equal to the shortest path length) to the worst (with the weakest fitness value that is equal to the longest path length) to copy the best chromosomes in a subset R, with the aim to preserve the best chromosomes. Next, in line 13, the selection operator is applied to the subpopulation P_i to drive the algorithm to improve the subpopulation fitness over the successive generations. The selection operator chooses and stores in P' the best chromosomes according to their fitness value. The chromosomes with the stronger fitness values have the chance to be selected and retained as "parents" for being used by the crossover operator. Therefore, the selection operator tends to eliminate those chromosomes that present a weak fitness value.

Then, in line 14, the crossover operator is applied over the subset P'. This operator randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two new chromosomes called "offspring" that are included in P'. The improvement of the subpopulation through the crossover operator will be determined by the current chromosomes of the subset P'. In detail, the MemPBPF employs a single point crossover where the parent number one passes its binary code to the left of that crossover point to the offspring number one. In like manner, parent number two passes its binary code to the left of the same crossover point to the offspring number two. Then, the binary code to the right of the crossover point of the parent number one goes to the offspring number two, and the parent number two passes its code to the offspring number one. Therefore, the offspring contain portions of the binary codes of both parents.

Last, in line 15, the bacterial mutation operator is applied over the subset P' to explore different regions in the search space that have not been covered with the current subpopulation. First, some clones of each chromosome of the subset P' are created. Second, one segment randomly selected suffers a mutation in each clone; the mutation alters a certain percentage of the bits in the clone. Third, the clones are evaluated using the fitness function (Algorithm 2). The resulting clone with the best fitness value is selected to transfer the mutation segment to the other clones. In the end, the best clone is compared with the original chromosome, where the best in terms of path length will be maintained, and the remaining clones will be removed.

Finally, the chromosomes in P' are evaluated using Algorithm 2. The new subpopulation $P_i(t_i + 1)$ will be composed by the best chromosomes in P' and the chromosomes in R. The main purpose of the MemPBPF (Algorithm 1) from line 7 to 25 is to find the global best chromosome $[k_a, k_r, \eta]_{opt}$ to perform the path planning employing the APF function (Algorithm 2). Once that the elementary membranes have evolved their corresponding subpopulations from line 8 to 20, all the elementary membranes $\mathcal{S}_i$ are merged into a fusion membrane $\mathcal{S}_F$ in line 21 of Algorithm 1.

In line 22, through the communication rules a copy of the global best chromosome $[k_a, k_r, \eta]_{opt}$ in the fusion membrane $\mathcal{S}_F$ is sent out to the skin membrane $\mathcal{S}_0$ and the recombination operation is conducted in the fusion membrane to exchange the information among chromosomes. The current global best chromosome in the skin membrane is compared with the one from the fusion membrane, where the best in terms of path length will be stored in the skin membrane. Then, the fusion membrane $\mathcal{S}_F$ is divided into m elementary membranes $\mathcal{S}_i$. At the end of the loop, the counter of generations t_0 is incremented in one.

In line 26 of Algorithm 1, the global best chromosome $[k_a, k_r, \eta]_{opt}$ obtained from the membrane evolution process is employed in the APF function to obtain the resultant path Q_G. Furthermore, the path length C is also obtained, the number of configurations τ to reach the target position, and a Boolean flag *target* that indicates if the target position was achieved.

B. APF ALGORITHM

Algorithm 2 presents the APF pseudocode that serves as a fitness function and as a local planning method for the proposed MemPBPF (Algorithm 1). The APF pseudocode employs the input parameters: start position q_0, target position q_f, environment information O_j, attraction proportional gain k_a, repulsion proportional gain k_r, and the step size η. The output parameters of the Algorithm 2 are the set of configurations Q_G (path), the cost C (path length), the number of configurations τ, and the flag *target*.

From line 1 to 5 of Algorithm 2 several parameters of the APF pseudocode are initialized. First, in line 1, the maximum number of allowed configurations N and the physical radius r of the AMR are set. Next, in lines 2 and 3, the AMR configuration counter τ and the cost-to-come C are set to zero, respectively. Then, in line 4, the Boolean flag *safe* is set to True. The main objective of the flag *safe* is serving as a stop condition if a collision is presented during the path planning. Finally, in line 5, the cost-to-go G is initialized with the Euclidean distance that is calculated from the start position q_0 to the target position q_f. The main objective of the cost G is to serve as a stop condition and to indicate numerically through the Euclidean distance if the AMR achieves the target position.

The core component of the Algorithm 2 is the APF hard-computing mathematical method to perform the path planning employing the set of proportional gains given by the MemPBPF algorithm. The path is generated through an iterative process described from line 6 to 16 of Algorithm 2. The parameters G, τ, and *safe* are the stop conditions for the loop, as can be seen in line 6 of Algorithm 2. Therefore, the iterative process will be performed if all of the following conditions are met.

1) If the cost G is greater than zero. The cost G is a distance measure about how far the AMR from the target position q_f is, hence, any position q_τ of the AMR whose distance G is greater than zero indicates that the AMR has not reached the target position and the loop must continue otherwise will stop.
2) If the configurations counter τ is less than the maximum number of configurations N allowed. This condition is established to avoid an endless loop.
3) If the flag *safe* is true. This flag indicates if the path is free of a collision, i.e., if it is safe for the AMR navigation. If a collision is detected during the path generation process, the loop will stop.

Hence, while the above conditions are met, the APF method is executed inside the loop to generate and evaluate the path. In line 7, the potential field $U(q_\tau)$ of the current position of the AMR is computed by the sum of the attraction potential field and the repulsive potential field.

In line 8 of Algorithm 2, the force $F(q_\tau)$ of the potential field is computed by the application of the gradient descent operation to the potential field. The force of the potential field will be employed to compute the next position of the AMR. In line 9, the next position $q_{\tau+1}$ of the AMR is computed by employing the information of the current position q_τ, the step size η, and the force of the potential field $F(q_\tau)$.

In line 10 and 11, the cost G and C are updated, respectively. The cost-to-go G represents the Euclidean distance from the new position $q_{\tau+1}$ to the target position q_f. The cost of G is employed to know if the AMR has reached the target position. The cost-to-come C is computed by the sum of its current value and the Euclidean distance from the current position of the AMR q_τ to the new position $q_{\tau+1}$. The cost C is employed to know the path length, i.e., the fitness value that will be employed for the MemPBPF algorithm to sort the solutions (chromosomes) to the given path planning problem.

From line 12 to 14, a safety condition is verified by the sum of the AMR radius r and the radius of the nearest obstacle r_o to the AMR. If the value of the sum is greater than or equal to the limit distance of effect of the repulsive potential field ρ_0 there will be a possible collision. Therefore, under this unsafe condition, the Boolean flag *safe* takes a false state, and the loop will stop. At the end of each cycle, the position/configuration counter is updated by incrementing its current value in one.

From line 17 to 23 of Algorithm 2, a target condition is verified. If the current (last) position q_τ is equal to the target position q_f the flag *target* takes a true state and the resultant path Q_G is composed by the set of configurations from q_0 to q_τ. Otherwise, the flag *target* takes a false state, and the cost C takes an infinite value (extremely high value). In the end, the Algorithm 2 returns the resultant path Q_G, the path length C, the number of configurations τ, and the flag *target*.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we describe the test specifications and we present the path planning results obtained with the MemPBPF algorithm considering static and dynamic environments. A comparative study of the MemPBPF algorithm against the parallel evolutionary artificial potential field (PEAPF) algorithm exposed in [20] and the pseudo bacterial potential field (PBPF) algorithm exposed in [24] is presented. Furthermore, the computational performance of the MemPBPF algorithm considering sequential, as well as parallel implementations, are given. Lastly, we implement the proposed MemPBPF algorithm in a robotic platform to test the path planning in a real AMR.

A. TEST SPECIFICATIONS

In Table 1, the AMR's mission, and the environment information are presented. The mission information is composed by the start position q_0 and the target position q_f coordinates in (x, y) format. The environment information O_j is given by the obstacles position and its corresponding radius in (x, y, r_o) format. Each environment configuration is a map instance designed to evaluate the performance and accuracy of the proposed MemPBPF algorithm; these instances were labeled as Env1, Env2, ..., Env8.

The test environments Env1 to Env8 employed to test the MemPBPF algorithm present challenging situations for testing path planning algorithms and cover well-known difficult problems, e.g., trap sites due to local minima, path-following prediction problems, problematic areas to reach because the target position is very close to an obstacle, and other situations described in [20], [24], [25]. These test environments represent just a sample of the types of scenarios that the AMR can expect to find in typical real-world indoor environments, which are composed of corridors, barriers, walls, L-shaped and U-shaped obstacles, among others.

TABLE 1. Set of test environments, the AMR's mission and the environment information are presented.

Test environment	Start $q_0(x,y)$	Target $q_f(x,y)$	Obstacles $O_j(x,y,r_o)$
Env1	(2.0,5.0)	(8.0,5.0)	(5.0,5.0,0.9)
Env2	(5.0,9.0)	(7.0,1.0)	(5.0,6.0,0.3),(7.0,7.0,0.3), (7.0,3.0,0.3)
Env3	(5.0,9.0)	(5.0,1.0)	(4.0,6.5,0.3),(2.5,6.5,0.3), (5.0,3.5,0.3),(6.5,3.5,0.3), (3.5,3.5,0.3)
Env4	(1.0,5.0)	(6.0,5.0)	(5.0,4.0,0.4),(5.0,5.0,0.4), (5.0,6.0,0.4)
Env5	(9.0,8.0)	(4.0,3.0)	(6.8,5.0,0.4),(5.5,5.0,0.4), (5.5,6.3,0.4)
Env6	(5.0,8.0)	(5.0,2.0)	(5.0,5.1,0.4),(4.0,5.1,0.4), (6.0,5.1,0.4),(4.0,6.1,0.4), (6.0,4.1,0.4)
Env7	(5.0,8.0)	(5.0,2.0)	(3.4,6.1,0.4),(3.4,5.3,0.4), (3.4,4.5,0.4),(4.2,4.5,0.4), (5.0,4.5,0.4),(5.8,4.5,0.4), (6.6,4.5,0.4),(6.6,5.3,0.4), (6.6,6.1,0.4)
Env8	(4.0,5.0)	(9.5,5.0)	(2.5,3.5,0.4),(2.5,4.5,0.4), (2.5,5.5,0.4),(7.5,3.5,0.4), (7.5,4.5,0.4),(7.5,5.5,0.4), (2.5,6.5,0.4),(3.5,6.5,0.4), (4.5,6.5,0.4),(5.5,6.5,0.4), (6.5,6.5,0.4),(7.5,6.5,0.4), (4.5,3.5,0.4),(5.5,3.5,0.4)

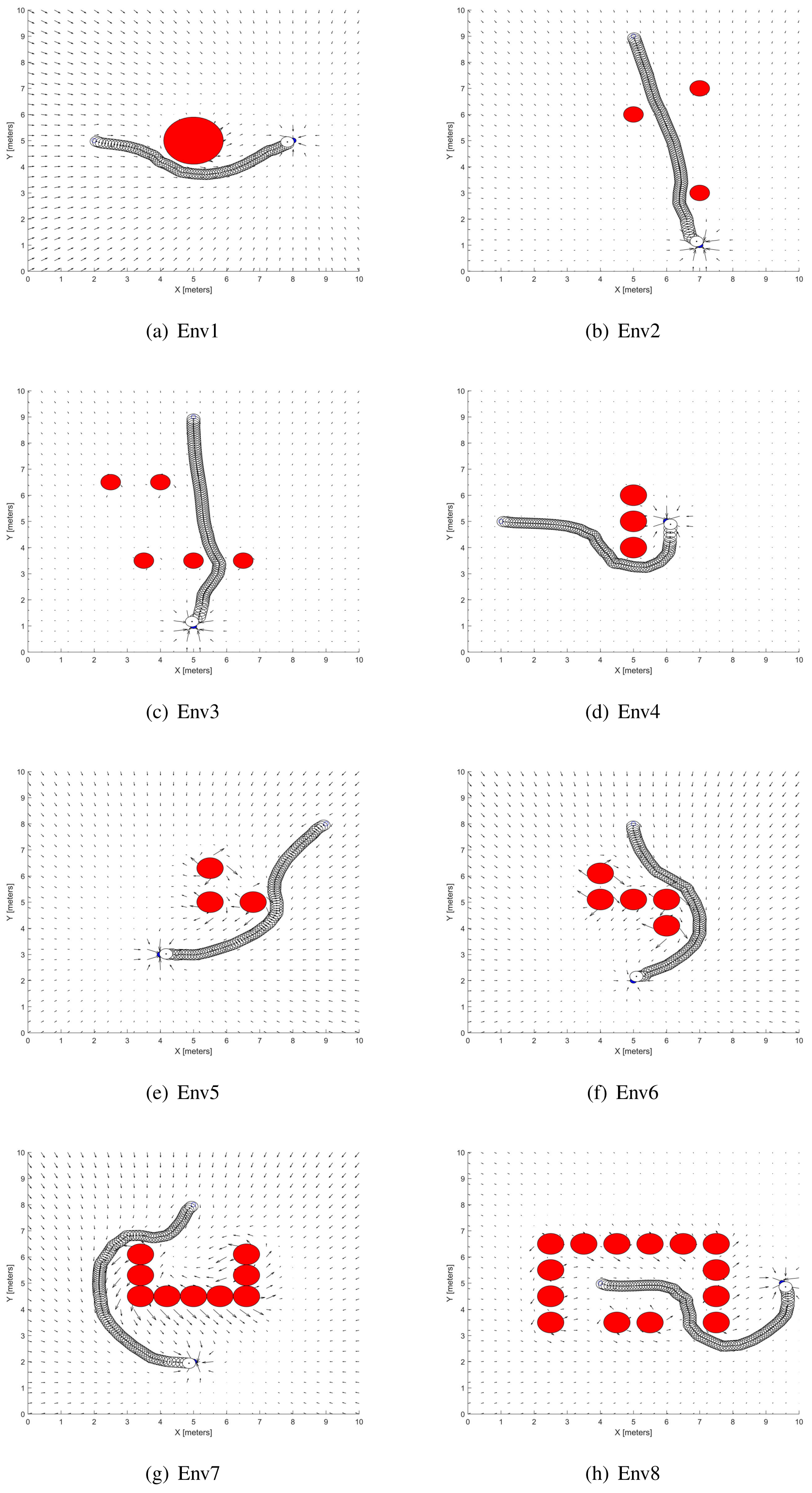
To make a comparison between the proposed MemPBPF algorithm and the PEAPF and PBPF algorithms, we considered the following points:

- All the results were achieved using a quad-core personal computer equipped with the Intel i7-7700HQ CPU@2.80 GHz; the installed memory (RAM) is of 8.0 GB. The computer runs with the Ubuntu 18.04 version of Linux, and Matlab R2018a installed.
- The stop condition N_g was fixed to ten generations. The main reason is due to the MemPBPF algorithm normally converges in ten generations. Adding more generations hardly ever will improve the solutions in a significant manner and the computation time is increased.
- To see the advantage of adding elementary membranes m, we varied them from two to four. Increasing the number of elementary membranes will improve the possibility of finding better solutions, but the computation time will also increase. To balance between good solutions and a moderate computation time we employed two and four elementary membranes.
- Each elementary membrane contains a subpopulation of 16 chromosomes (candidate solutions). Population sizing has been one of the important topics to consider in evolutionary computation. Researchers usually argue that a small population size could guide the algorithm to poor solutions and that a large population size could make the algorithm expend more computation time in finding a solution [33]. In this work, making a compromise between solution quality and computation time, we have found the best results with a subpopulation of 16 chromosomes.
- Each chromosome $p_j = [k_a, k_r, \eta]$ codifies the information of the proportional gains k_a and k_r and the step size η. Hence, each chromosome p_j contains three genes, where each gene is a bit string with a fixed length. The first gene corresponds to the attractive proportional gain k_a, the second gene corresponds to the repulsive proportional gain k_r, and the third gene entails the step size η. Once that the MemPBPF algorithm has evolved the chromosomes, the global best chromosome $[k_a, k_r, \eta]_{opt}$ is sent to Algorithm 2 to generate the path Q_G.
- Algorithm 1 and 2 allow us to use one repulsion gain k_r for each obstacle presented in the environment; however, the common practice in the artificial potential method is to use only one k_r for all obstacles [24]. To reduce the computational load and to make a fair comparison, we have also used just one k_r for testing the proposed MemPBPF algorithm, and the PEAPF and PBPF algorithms.
- The $PBPF_{par}$ is composed of the selection rate fixed at 50%; a single point crossover was employed; the bacterial mutation rate was fixed at 20%, and the number of clones per chromosome employed was four.
- The AMR configuration is given by $q = (x, y, \theta)$; which means that the center of the AMR is at coordinates (x, y), and its orientation is given by θ. In all cases, it is considered that the AMR is oriented to the next point to visit.
- The AMR has a physical size representation given by its radius of $r = 0.2$ meters.
- The limit distance of effect ρ_0 of the repulsive potential field $U_r(q)$ for each obstacle O_j will be determined by its radius multiplied by two.
- The stop condition N was fixed to 2,000 to indicate the maximum number of AMR configurations allowed to reach the target position.

B. PATH PLANNING RESULTS

Table 1 presents the set of test environments employed for testing the proposed MemPBPF algorithm. Each test environment is structured, and all the obstacles on it are in a known position. Thereby, it is assumed that the environment is known and given in advance. The objective is to find an optimal, collision-free path between the start position q_0 and the target position q_f, in a static environment composed of obstacles.

In Fig. 2 is shown the resultant path Q_G followed by the AMR in each test environment described in Table 1. The resultant path Q_G is the shortest collision-free path generated with the optimal set of gain parameters found by the proposed MemPBPF (Algorithm 1) with four elementary membranes, $m = 4$. The optimal set of gain parameters k_a and k_r, the path length C in meters, and the number of configurations τ to achieve the target position q_f in each test environment are shown in Table 2.



(a) Env1 (b) Env2 (c) Env3 (d) Env4 (e) Env5 (f) Env6 (g) Env7 (h) Env8

FIGURE 2. Path planning results for the different test environments, the shortest path followed by the robot was generated with the optimal set of gain parameters computed by the MemPBPF algorithm with $m = 4$.

Fig. 3 shows graphically a summary of the path planning results in terms of path length for four different implementations, the PEAPF algorithm, the PBPF algorithm, the MemPBPF algorithm with $m = 2$, and the MemPBPF algorithm with $m = 4$ in each test environment, respectively. Each test was independently executed thirty times for

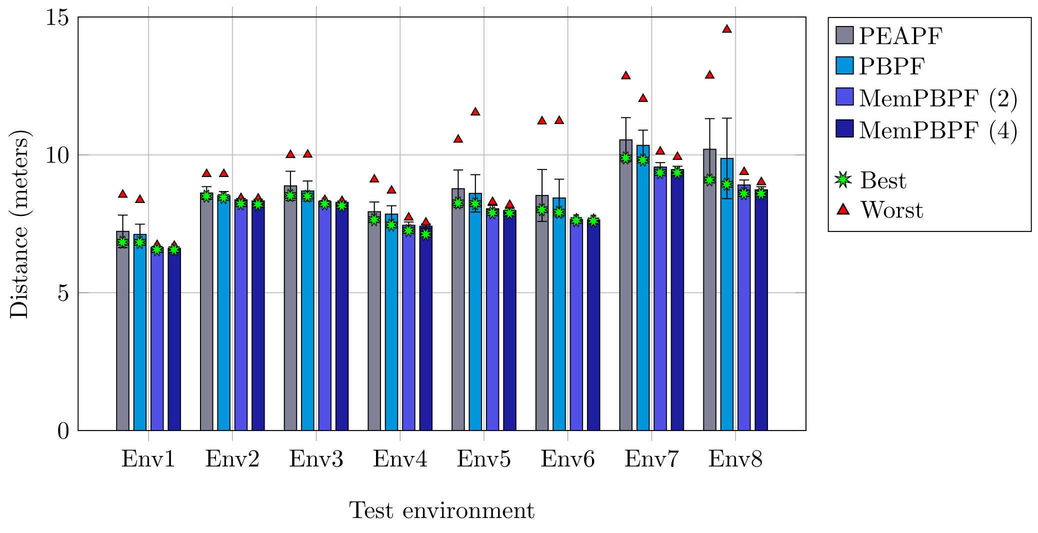


FIGURE 3. Path length results for the four different implementations: PEAPF, PBPF, MemPBPF with $m = 2$, MemPBPF with $m = 4$.

TABLE 2. Path planning results for the MemPBPF algorithm with four elementary membranes, the path length is expressed in meters.

Test environment	k_a	k_r	Path length	Number of configurations
Env1	2.118	8.039	6.558	80
Env2	0.392	0.353	8.204	100
Env3	0.314	0.745	8.155	99
Env4	0.235	0.628	7.121	88
Env5	1.020	5.137	7.886	92
Env6	3.255	9.451	7.602	91
Env7	1.412	6.431	9.349	110
Env8	0.471	2.824	8.588	106

each implementation of each test environment. The results show the arithmetic mean, worst, best, and standard deviation for the thirty independent tests on each environment (Env1, Env2, . . . , Env8) to account for the randomness of the implementations.

The path planning results in Fig. 3 shows that the proposed MemPBPF algorithm provides better solutions than those obtained with the PEAPF and the PBPF algorithms, even when the MemPBPF algorithm is implemented with just two elementary membranes $m = 2$. Also, in the results it can be observed how the increase in the number of elementary membranes m in the MemPBPF algorithm allows minimizing the path length, being the main objective to obtain the shortest path for achieving the target position without colliding with the obstacles presented in the environment.

An example of the best performance of the proposed MemPBPF algorithm over the PEAPF and PBPF algorithms in terms of the average path length, it was found in the test environment Env8. Here, the PEAPF algorithm (■ Mean) obtained an average path length of 10.203 meters and the PBPF algorithm (■ Mean) obtained an average path length of 9.871 meters from the thirty independent test runs, the MemPBPF algorithm with two elementary membranes (■ Mean) obtained an average of 8.909 meters, and the MemPBPF algorithm with four elementary membranes (■ Mean) yielded 8.733 meters; giving a difference of 1.470 and 1.138 meters between the average path length found by the PEAPF and PBPF algorithms and the MemPBPF algorithm.

In terms of the worst results in path length (▲ Worst), also the best performance of the MemPBPF algorithm compared to the PEAPF and PBPF algorithms was found in the test environment Env8. The PEAPF algorithm obtained the worst result with a path length of 12.872 meters, and the PBPF algorithm obtained the worst result with a path length of 14.536 meters, the proposed MemPBPF algorithm with two elementary membranes ($m = 2$) obtained the worst result with a path length of 9.377 meters, and the MemPBPF algorithm with $m = 4$ obtained the worst result with a path length of 9.009 meters. The above gives a difference of 3.863 and 5.527 meters between the worst path length found by the PEAPF and PBPF algorithms and the MemPBPF algorithm.

The best performance of the proposed MemPBPF algorithm over the PEAPF and PBPF algorithms in terms of the best path length (✳ Best), it was found in the test environment Env7, where the PEAPF and PBPF algorithms obtained a path with 9.889 and 9.812 meters in length, respectively, the MemPBPF algorithm with $m = 2$ and $m = 4$ yielded 9.354 meters and 9.349 meters, respectively. In this case, the difference between the best path found by the MemPBPF algorithm in comparison to the PEAPF and PBPF algorithms was 0.540 and 0.463 meters.

Also, Fig. 3 shows the standard deviation for the thirty independent tests in each test environment for each implementation. The results of the PEAPF and PBPF algorithms present a high standard deviation compared to the MemPBPF algorithm implementations, i.e., the resultant path length values are spread out over wider range values. On the

TABLE 3. Path planning average success rate.

Test environment	PEAPF	PBPF	MemPBPF $m = 2$	MemPBPF $m = 4$
Env1	0.900	0.933	1.000	1.000
Env2	0.933	0.967	1.000	1.000
Env3	1.000	1.000	1.000	1.000
Env4	0.900	0.933	1.000	1.000
Env5	0.867	0.900	1.000	1.000
Env6	0.900	0.933	1.000	1.000
Env7	0.867	0.833	1.000	1.000
Env8	0.833	0.767	1.000	1.000

other hand, the results of the MemPBPF algorithm implementations present a low standard deviation, i.e., the resultant path length values tend to be close to the mean, which is desirable. In terms of the standard deviation, the best performance it was found in the test environment Env8, where the PEAPF and PBPF algorithms obtained a standard deviation of 1.111 and 1.460 meters, respectively, the MemPBPF algorithm with $m = 2$ standard deviation was 0.181 meters, and the MemPBPF algorithm with $m = 4$ yielded 0.113 meters, giving a difference of 0.998 and 1.347 meters between the PEAPF and PBPF algorithms and the MemPBPF algorithm. The best standard deviation was 0.016; it was obtained on the tests executed on the environment Env6 using the MemPBPF algorithm with $m = 4$, and the worst was 1.460, which was obtained on the tests executed on the environment Env8 with the PBPF algorithm.

Table 3 shows the path planning average success rate, which is defined by the number of times that the AMR reached the target position without any collision divided by the number of the total run tests. In this work, it was performed thirty independent tests for each implementation in each test environment. In the results, it can be observed that the proposed MemPBPF algorithm with two and four elementary membranes achieved the target position successfully in the entire thirty independent tests over each test environment. On the other hand, the PEAPF and PBPF algorithms reach the target position successfully in all the tests only on the environment Env3. Moreover, the worst results by the PEAPF and PBPF algorithms were obtained over the tests executed on environment Env8, with an average success rate of 83.3% and 76.7%, respectively, i.e., the PEAPF algorithm from thirty independent tests successfully obtained a free-collision path to reach the target position in twenty-five executions, and it fails in five and the PBPF algorithm successfully obtained a free-collision path in twenty-three executions, and it fails in seven. As a general result, the obtained outcomes of Table 3 demonstrates the robustness of the proposed MemPBPF algorithm for path planning in terms of high success reliability to achieve the target position.

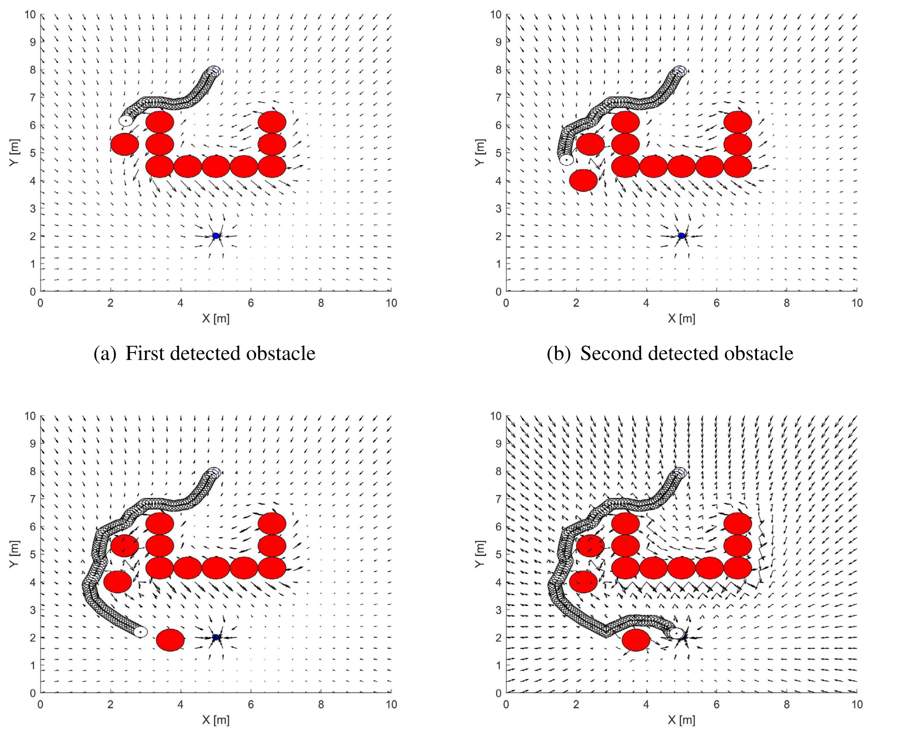
C. PATH PLANNING IN CHANGING ENVIRONMENTS

In real-world applications, the AMR frequently is faced with unknown or partially known environments that contain static and dynamic obstacles. These navigation conditions require the AMR to be able to respond and make decisions, a task that is known as online path planning. Some works perform offline path planning to know the initial condition of the environment, and once the AMR navigation starts, they change to the online mode to adjust the path already known, considering the modified environment. In this context, we are going to employ the MemPBPF algorithm to perform the online path planning in the test environment Env7 that will be modified with new random static obstacles; these new obstacles are unknown to the AMR.

Fig. 4 illustrates the online path planning experiment considering the new static obstacles. For this experiment, the test environment Env7 was employed, see Table 1. First, the path planning is performed in offline mode because we know the environment information described by the original test environment Env7, see Fig. 2(g). The minimum path length found to reach the target position is 9.349 meters, using the best set of gain parameters, as it is described in Table 2. At position (2.400, 5.300), a new static obstacle is added to change the environment configuration. After a while, when the AMR has traveled 3.544 meters, it reaches the position (2.411, 6.110). The AMR senses the new obstacle; it calculates the obstacle position to update the environment layout map, as shown in Fig. 4(a). The AMR path planning algorithm based on the MemPBPF now has a different environment layout. Therefore, it is necessary to update the path by recalculating the set of gain parameters to update the path to reach the target position.

Next, at position (2.200, 4.000), a second new static obstacle is added to change the environment configuration. After a while, when the AMR has traveled an additional distance of 1.761 meters, it reaches the position (1.727, 4.718). The AMR senses the second new obstacle; it calculates the obstacle position to update the environment layout, as shown in Fig. 4(b). Then, at position (3.700, 1.900), a third new static obstacle is added to change the environment configuration. After a while, when the AMR has traveled an additional distance of 3.242 meters, it reaches the position (2.850, 2.196). The AMR senses the third new obstacle; it calculates the obstacle position to update the environment layout, as shown in Fig. 4(c). Finally, the AMR follows the new path to reach the target position. The complete path to achieve the target position is shown in Fig. 4(d). The total path length from the original start point to the target position is $3.544 + 1.761 + 3.242 + 2.303 = 10.850$ meters.

Now, we are going to present an experiment where the MemPBPF algorithm will be employed for online path planning dealing with new dynamic obstacles. This experiment looks at the case of new dynamic obstacles that can be persons or other AMRs moving through the test environment. Differently to the case of the unknown static obstacles, here the new obstacle will not remain static, it will be moving with a defined trajectory that is unknown to the AMR. This problem is more complicated than the last scenario due to the MemPBPF algorithm must find a feasible path in a noisy surface because at every iteration the searching zone is different.



(a) First detected obstacle (b) Second detected obstacle

(c) Third detected obstacle (d) The complete path to achieve the target position

FIGURE 4. **Path planning results for the test environment Env7 with unknown static obstacles, the shortest path followed by the AMR was generated with the optimal set of gain parameters computed by the MemPBPF algorithm with $m = 4$.**

For this experiment, we are going to employ the test environment Env4, see Table 1. Fig. 5 illustrates the online path planning experiment considering the new dynamic obstacles. First, the path planning is performed in offline mode because we have the environment information described by the original test environment Env4, see Fig. 2(d). The minimum path length found to reach the target position is 7.121 meters, using the best set of gain parameters, see Table 2. Now the AMR navigation can start. After a while, when the AMR has traveled 1.471 meters, it reaches the position (2.504, 4.887). The AMR senses a new obstacle, which is located at (3.100, 4.800) at that moment, the AMR calculates the obstacle position to update the environment layout map, as shown in Fig. 5(a).

The AMR now has a different environment layout. Therefore, it is necessary to update the path to reach the target position. After a while, when the AMR has traveled 2.617 meters, it reaches the position (4.477, 3.385). The AMR senses the second new dynamic obstacle that is located at (5.100, 3.300) at that moment. The AMR calculates the obstacle position to update the environment layout, as shown in Fig. 5(b). Then, when the AMR has traveled 2.383 meters, it reaches the position (3.684, 3.110). The AMR senses a third new dynamic obstacle, located at (6.206, 3.278) at that moment. The AMR calculates the obstacle position to update the environment layout, as shown in Fig. 5(c). Finally, the AMR follows the new path to reach the target position; the complete path is shown in Fig. 5(d). The total path length from the original start point to the target position is $1.471 + 2.617 + 2.383 + 2.240 = 8.711$ m.

D. COMPUTATIONAL PERFORMANCE RESULTS

A summary of the results in the computational performance evaluation is presented in Fig. 6 for the proposed MemPBPF algorithm with $m = 2$, and Fig. 7 shows the results for the MemPBPF algorithm with $m = 4$. Fig. 6 and 7 show the average execution time for the sequential implementation of the MemPBPF algorithm in Matlab, for the sequential implementation in C/C++, and for the parallel implementation in C/C++ using the application-programming interface Open Multi-Processing (OpenMP). To compare the different implementations, we carried out thirty independent run tests on each environment for each MemPBPF algorithm implementation with two and four elementary membranes; the average execution time was recorded.

Fig. 6 and 7 clearly show the advantages of the parallel implementation in execution time. The best example in the results of Fig. 6 shows that the average execution time for path planning on the test environment Env4 takes 159.352 seconds with a standard deviation of 30.361 for the sequential implementation in Matlab of the proposed MemPBPF algorithm with two elementary membranes. It takes 41.863 seconds with a standard deviation of 4.888 for the sequential implementation in C/C++, and 11.242 seconds with a standard deviation of 1.710 for the parallel implementation. Giving a 14.175 times faster execution on

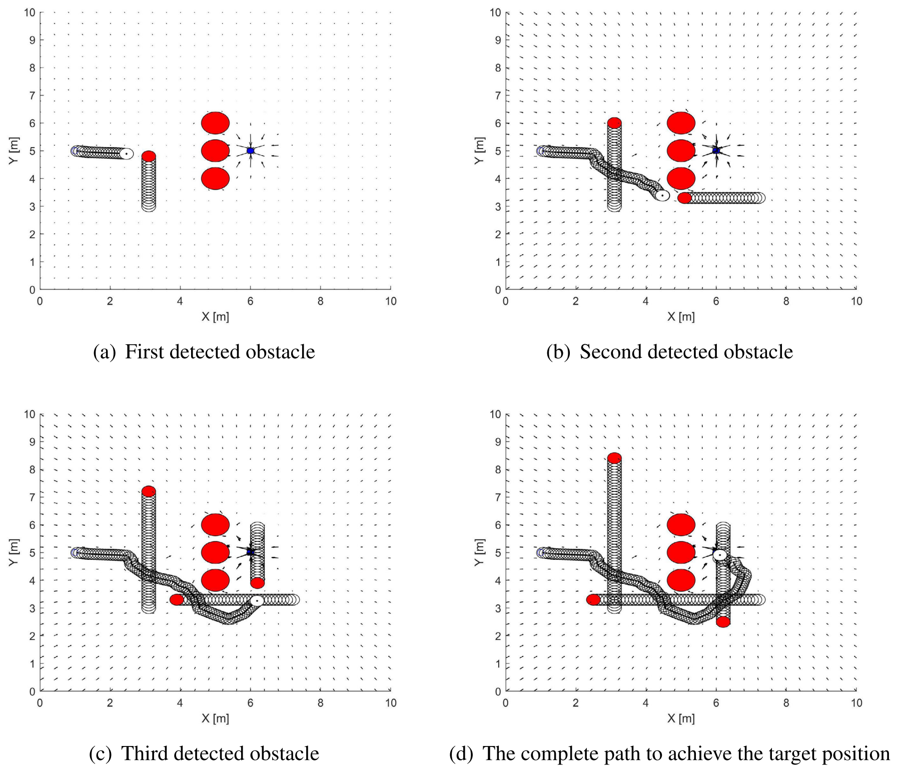


FIGURE 5. Path planning results for the test environment Env4 with unknown dynamic obstacles, the shortest path followed by the AMR was generated with the optimal set of gain parameters computed by the MemPBPF algorithm with $m = 4$.

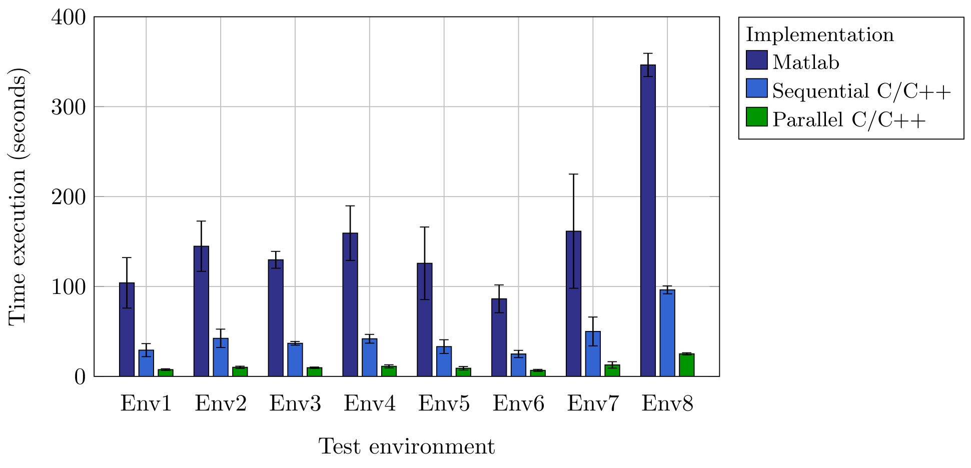


FIGURE 6. The average execution time of the proposed MemPBPF algorithm with $m = 2$.

the parallel implementation over the sequential implementation in Matlab, and a 3.724 times faster than the sequential implementation in C/C++.

For the results in Fig. 7, the best example is found in the test environment Env8, where the average execution time takes 695.480 seconds with a standard deviation of 21.295 for the sequential implementation in Matlab of the MemPBPF algorithm with four elementary membranes. It takes 190.791 seconds with a standard deviation of 4.788 for the sequential implementation in C/C++ of the MemPBPF algorithm, and 50.389 seconds with a standard deviation of 1.739 for the parallel implementation. Giving a 13.802 times faster execution on the parallel implementation over the sequential implementation in Matlab, and 3.786 times faster than the sequential implementation in C/C++.

Fig. 8 shows a summary of the speedup achieved for the computational performance evaluation. The speedup establishes how much a parallel implementation is faster than its equivalent in the sequential form. The speedup is defined by the quotient of T_1 divided by T_p, where T_1 is the execution time in a processor, and T_p is the run time on N_p number of processors. With the aim of a fair comparison in this

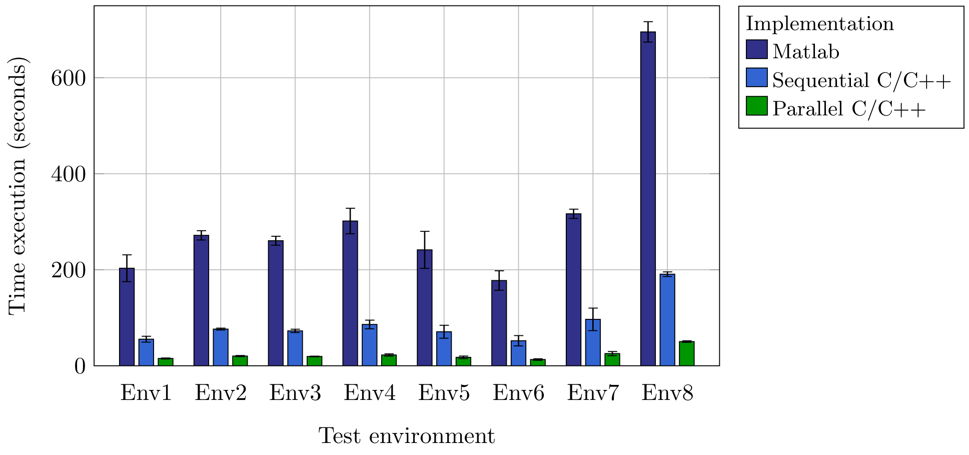


FIGURE 7. **The average execution time of the proposed MemPBPF algorithm with $m = 4$.**

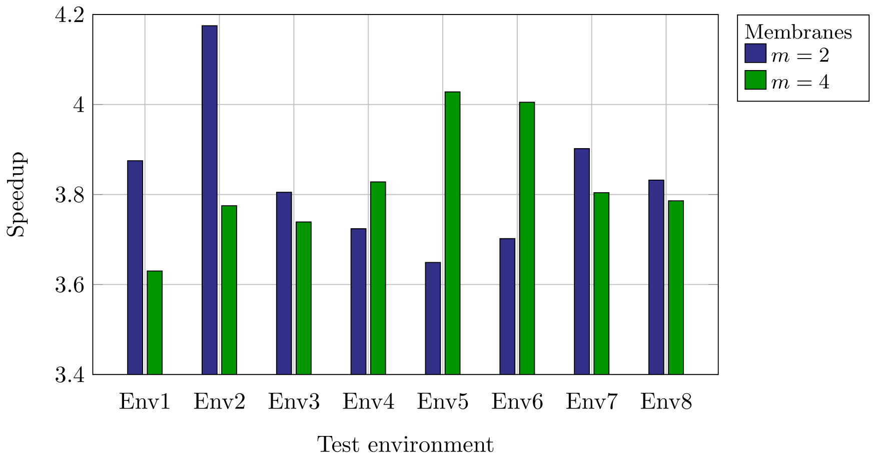


FIGURE 8. **Speedup results for the computational implementation of the proposed MemPBPF algorithm with two and four elementary membranes.**

study, we compared the sequential and parallel implementations in C/C++ of the MemPBPF algorithm with two and four elementary membranes. We have considered T_1 as the execution time taken by the MemPBPF algorithm using the C/C++ sequential implementation and T_p as the execution time taken by the MemPBPF algorithm in C/C++ parallel implementation with $N_p = 4$. The results show that the best speedup was 4.175 for the MemPBPF algorithm with $m = 2$ on the test environment Env2, and 4.028 with $m = 4$ on the test environment Env5. The average speedup for the MemPBPF algorithm with $m = 2$ was 3.833 and with $m = 4$ was 3.825 times faster.

The results are very consistent. It clearly can be seen in Fig. 6 and 7 that the average execution time is almost augmented twice. The main reason is that the number of elementary membranes was augmented in two with the aim to obtain better results in terms of path length (shorter paths). Through the parallel implementation results, we can observe that the MemPBPF algorithm can be implemented on GPU. In GPU computing, the parallelism arises from each thread

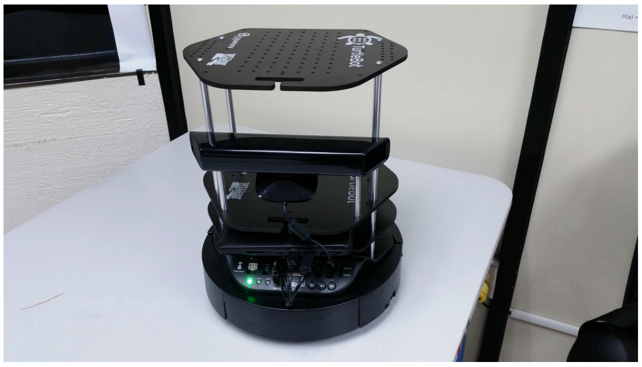


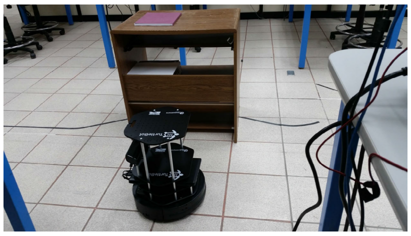
FIGURE 9. **Differential wheeled robot TurtleBot2 employed to validate the MemPBPF algorithm for path planning.**

independently running the same program on different data. The MemPBPF algorithm for path planning can meet the criteria of computationally intensive and massively parallel. Therefore, a GPU implementation of the MemPBPF algorithm can be potent to accelerate the evaluation of

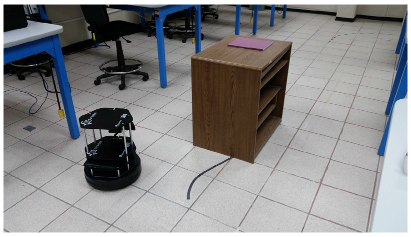
(a) Start position q_0

(b) Execution of the path

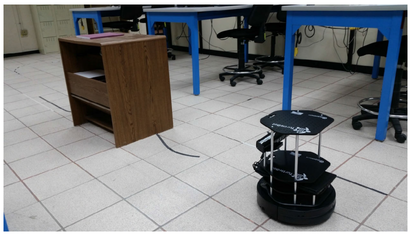
(c) Execution of the path



(d) Execution of the path



(e) Execution of the path



(f) Target position q_f

FIGURE 10. Implementation of the proposed MemPBPF algorithm on the differential wheeled robot TurtleBot2.

the solutions, because the problem is stated as a data-parallel problem.

E. ROBOTIC PLATFORM IMPLEMENTATION

To test the proposed MemPBPF algorithm for AMR path planning, we have implemented it on an open robotic platform: the differential wheeled robot TurtleBot2, see Fig. 9. The main functional specifications of the TurtleBot2 are the maximum translational velocity: 0.7 m/s, and the maximum rotational velocity: 180 deg/s.

The experiments were realized using a test environment like the Env1 over a real laboratory environment considering just one static obstacle (brown drawer cabinet). The start position was fixed at $q_0 = (2.8, 5.0)$, the target position at $q_f = (6.5, 5.0)$, and the obstacle information was $O_j = (5.0, 5.0, 0.3)$. For testing purposes, the linear velocity of the differential wheeled robot was fixed to 0.1 m/s and the angular velocity was fixed to 36 deg/s. Fig. 10(a)-(f) show a sequence of images with the position of the AMR in a given time. The proposed MemPBPF (Algorithm 1) generates the path Q_G to provide a set of motion commands from q_0 to q_f formed by the primitives of rotate and advance, i.e., from point to point in the resultant path, the primitives are calculated and given in degrees for the rotations and in meters for the advance commands. The relation between the environment simulation and the real environment is 1:1, therefore, one meter in the simulation is equivalent to one meter in the real environment. By employing these motion primitives, the AMR executes the commands to reach the target position q_f. As can be seen in Fig. 10 the differential wheeled robot executes its path satisfactorily in a real laboratory environment.

V. CONCLUSION

A hybrid algorithm based on membrane pseudo-bacterial potential field (MemPBPF) was proposed to solve path planning problems efficiently for AMR navigation. The proposed MemPBPF algorithm combines membrane computing, pseudo-bacterial genetic algorithm, and the artificial potential field method to generate a feasible, safe and smooth path for AMR navigation. Dynamic membranes that include a pseudo-bacterial potential field in which the evolution rules such as membrane division and merge are the main features of the proposed MemPBPF algorithm. Therefore, the MemPBPF algorithm reaches an evolutionary behavior for finding an efficient path planning in terms of path length and time execution under a parallel computing implementation.

We employed different test scenarios to evaluate the MemPBPF algorithm. The results obtained in the different static and dynamic scenarios show that the proposed MemPBPF algorithm achieves the three requirements efficiently to solve the path planning problem: safety, length, and smoothness, which makes the MemPBPF algorithm suitable to find competitive results for AMR navigation in complex and real scenarios.

The path planning results demonstrate that the MemPBPF algorithm yields better solutions in all the test environments. Also, the path planning results are improved when the MemPBPF algorithm is established with a double number of elementary membranes. Another advantage of the proposed MemPBPF algorithm is low variability, making it highly reliable for AMR path planning in real-world scenarios. The experimental results show the benefits of the proposed MemPBPF algorithm for path planning in AMR navigation in terms of path length.

The performance of the MemPBPF algorithm has been assessed using the test environments composed of different configurations and several obstacles. Where the parallel implementation has shown the high-performance scalability of the proposed MemPBPF algorithm and the capability to perform the path planning for AMR navigation, hence, sequential implementation of the proposed MemPBPF algorithm can be employed for low-cost onboard computers. Besides, for an accelerated execution, parallel implementation can be performed in a multicore CPU for taking advantage of novel computing architectures, and for future implementations, a GPU can be employed to speed up the path computation. In this regard, the MemPBPF algorithm could be useful to many applications in AMR for global and local path planning, including domestic and industrial AMRs, exploration vehicles, autonomous underwater vehicles, unmanned aerial vehicles, and self-driving cars.

REFERENCES

[1] A. Hidalgo-Paniagua, M. A. Vega-Rodríguez, and J. Ferruz, "Applying the MOVNS (multi-objective variable neighborhood search) algorithm to solve the path planning problem in mobile robotics," *Expert Syst. Appl.*, vol. 58, pp. 20–35, Oct. 2016.

[2] M. A. Contreras-Cruz, V. Ayala-Ramirez, and U. H. Hernandez-Belmonte, "Mobile robot path planning using artificial bee colony and evolutionary programming," *Appl. Soft Comput.*, vol. 30, pp. 319–328, May 2015.

[3] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge Univ. Press, 2006.

[4] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, "Path planning," in *Wheeled Mobile Robotics*, G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, Eds. London, U.K.: Butterworth, 2017, ch. 4, pp. 161–206.

[5] G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, "Autonomous guided vehicles," in *Wheeled Mobile Robotics*, G. Klančar, A. Zdešar, S. Blažič, and I. Škrjanc, Eds. London, U.K.: Butterworth, 2017, ch. 7, pp. 161–206.

[6] G. Paŭn, "Computing with membranes," *J. Comput. Syst. Sci.*, vol. 61, no. 1, pp. 108–143, Aug. 2000.

[7] N. E. Nawa, T. Furuhashi, T. Hashiyama, and Y. Uchikawa, "A study on the discovery of relevant fuzzy rules using pseudobacterial genetic algorithm," *IEEE Trans. Ind. Electron.*, vol. 46, no. 6, pp. 1080–1089, Dec. 1999.

[8] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robot. Res.*, vol. 5, no. 1, pp. 90–98, Mar. 1986.

[9] O. Montiel, U. Orozco-Rosas, and R. Sepúlveda, "Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles," *Expert Syst. Appl.*, vol. 42, no. 12, pp. 5177–5191, 2015.

[10] K. J. Waldron and J. Schmiedeler, "Kinematics," in *Springer Handbook Robotics*, B. Siciliano and O. Khatib, Eds. Cham, Switzerland: Springer, 2016, ch. 2, pp. 11–36.

[11] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA, USA: MIT Press, 2011.

[12] T. T. Mac, C. Copot, D. T. Tran, and R. De Keyser, "Heuristic approaches in robot path planning: A survey," *Robot. Auto. Syst.*, vol. 86, pp. 13–28, Dec. 2016.

[13] C. Cai and S. Ferrari, "Information-driven sensor path planning by approximate cell decomposition," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 39, no. 3, pp. 672–689, Jun. 2009.

[14] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning—Using the voronoi diagram for a clearance-based shortest path," *IEEE Robot. Autom. Mag.*, vol. 15, no. 2, pp. 58–66, Jun. 2008.

[15] E. Masehian and D. Sedighizadeh, "Classic and heuristic approaches in robot motion planning-a chronological review," *Int. J. Mech., Aerosp., Ind., Mech. Manuf. Eng.*, vol. 23, no. 5, pp. 101–106, 2007.

[16] J.-H. Kim, J.-H. Han, Y.-H. Kim, S.-H. Choi, and E.-S. Kim, "Preference-based solution selection algorithm for evolutionary multiobjective optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 1, pp. 20–34, Feb. 2012.

[17] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *Int. J. Robot. Res.*, vol. 25, no. 7, pp. 627–643, 2006.

[18] W. Khaksar, T. S. Hong, K. S. M. Sahari, M. Khaksar, and J. Torresen, "Sampling-based online motion planning for mobile robots: Utilization of tabu search and adaptive neuro-fuzzy inference system," *Neural Comput. Appl.*, vol. 31, no. 2, pp. 1275–1289, Feb. 2019.

[19] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda, "Mobile robot path planning using membrane evolutionary artificial potential field," *Appl. Soft Comput.*, vol. 77, pp. 236–251, Apr. 2019.

[20] O. Montiel, R. Sepúlveda, and U. Orozco-Rosas, "Optimal path planning generation for mobile robots using parallel evolutionary artificial potential field," *J. Intell. Robotic Syst.*, vol. 79, no. 2, pp. 237–257, Aug. 2015.

[21] F. Bayat, S. Najafinia, and M. Aliyari, "Mobile robots path planning: Electrostatic potential field approach," *Expert Syst. Appl.*, vol. 100, pp. 68–78, Jun. 2018.

[22] B. Kovács, G. Szayer, F. Tajti, M. Burdelis, and P. Korondi, "A novel potential field method for path planning of mobile robots by adapting animal motion attributes," *Robot. Auton. Syst.*, vol. 82, pp. 24–34, Aug. 2016.

[23] S. M. H. Rostami, A. K. Sangaiah, J. Wang, and X. Liu, "Obstacle avoidance of mobile robots using modified artificial potential field algorithm," *EURASIP J. Wireless Commun. Netw.*, vol. 2019, no. 1, p. 70, Mar. 2019.

[24] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda, "Pseudo-bacterial potential field based path planner for autonomous mobile robot navigation," *Int. J. Adv. Robotic Syst.*, vol. 12, no. 7, p. 81, Jul. 2015.

[25] S. S. Ge and Y. J. Cui, "New potential functions for mobile robot path planning," *IEEE Trans. Robot. Autom.*, vol. 16, no. 5, pp. 615–620, Oct. 2000.

[26] T. Furuhashi, Y. Miyata, K. Nakaoka, and Y. Uchikawa, "A new approach to genetic based machine learning and an efficient finding of fuzzy rules," in *Advances in Fuzzy Logic, Neural Networks and Genetic Algorithms*, T. Furuhashi, Ed. Berlin, Germany: Springer, 1995, pp. 173–189.

[27] N. E. Nawa, T. Hashiyama, T. Furuhashi, and Y. Uchikawa, "A study on fuzzy rules discovery using pseudo-bacterial genetic algorithm with adaptive operator," in *Proc. IEEE Int. Conf. Evol. Comput.*, Apr. 1997, pp. 589–593.

[28] O. Kramer, *Genetic Algorithms Essentials*. Cham, Switzerland: Springer, 2017, pp. 11–19.

[29] J. Botzheim, Y. Toda, and N. Kubota, "Bacterial memetic algorithm for offline path planning of mobile robots," *Memetic Comput.*, vol. 4, no. 1, pp. 73–86, 2012.

[30] J. Botzheim, L. Gál, and L. T. Kóczy, "Fuzzy rule base model identification by bacterial memetic algorithms," in *Recent Advances in Decision Making*, E. Rakus-Andersson, R. R. Yager, N. Ichalkaranje, and L. C. Jain, Eds. Berlin, Germany: Springer, 2009, pp. 21–43.

[31] D. B. Fogel, "An introduction to evolutionary computation," in *Evolutionary Computation: The Fossil Record*. Hoboken, NJ, USA: Wiley, 1998, p. 656.

[32] X. Liang, L. Li, J. Wu, and H. Chen, "Mobile robot path planning based on adaptive bacterial foraging algorithm," *J. Central South Univ.*, vol. 20, no. 12, pp. 3391–3400, Dec. 2013.

[33] U. Orozco-Rosas, O. Montiel, and R. Sepúlveda, "An optimized GPU implementation for a path planning algorithm based on parallel pseudo-bacterial potential field," in *Nature-Inspired Design of Hybrid Intelligent Systems* (Studies in Computational Intelligence), vol. 667, P. Melin, O. Castillo, and J. Kacprzyk, Eds. Cham, Switzerland: Springer, 2017, ch. 31, pp. 477–492.

[34] X. Wang, G. Zhang, J. Zhao, H. Rong, F. Ipate, and R. Lefticaru, "A modified membrane-inspired algorithm based on particle swarm optimization for mobile robot path planning," *Int. J. Comput. Commun. Control*, vol. 10, no. 5, pp. 732–745, Oct. 2015.
[35] G. Păun and G. Rozenberg, "A guide to membrane computing," *Theor. Comput. Sci.*, vol. 287, no. 1, pp. 73–100, 2002.
[36] G. Zhang, J. Cheng, M. Gheorghe, and Q. Meng, "A hybrid approach based on differential evolution and tissue membrane systems for solving constrained manufacturing parameter optimization problems," *Appl. Soft Comput.*, vol. 13, no. 3, pp. 1528–1542, Mar. 2013.
[37] G. Zhang, J. Cheng, and M. Gheorghe, "Dynamic behavior analysis of membrane-inspired evolutionary algorithms," *Int. J. Comput. Commun. Control*, vol. 9, no. 2, pp. 227–242, Apr. 2014.
[38] G. Zhang, M. Gheorghe, and C. Wu, "A quantum-inspired evolutionary algorithm based on P systems for knapsack problem," *Fundam. Informat.*, vol. 87, no. 1, pp. 93–116, Nov. 2008.
[39] G. Zhang, M. Gheorghe, L. Pan, and M. J. Pérez-Jiménez, "Evolutionary membrane computing: A comprehensive survey and new results," *Inf. Sci.*, vol. 279, pp. 528–551, Sep. 2014.
[40] C. Liu, G. Zhang, H. Liu, M. Gheorghe, and F. Ipate, "An improved membrane algorithm for solving time-frequency atom decomposition," in *Membrane Computing*, G. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, Eds. Berlin, Germany: Springer, 2010, pp. 371–384.
[41] G. Zhang, J. Cheng, M. Gheorghe, F. Ipate, and X. Wang, "QEAM: An approximate algorithm using p systems with active membranes," *Int. J. Comput. Commun. Control*, vol. 10, no. 2, pp. 263–279, Apr. 2015.
[42] G. Păun, "A quick introduction to membrane computing," *J. Logic Algebr. Program.*, vol. 79, no. 6, pp. 291–294, Aug. 2010.

ULISES OROZCO-ROSAS received the B.Eng. degree in electronics engineering from the Universidad Autónoma de Baja California, Mexico, in 2004, and the M.Sc. and Ph.D. degrees in digital systems from the Instituto Politécnico Nacional, Mexico, in 2014 and 2017, respectively. He held a postdoctoral position; from 2017 to 2018, he was appointed at ETSII, Department of Computer Science, Universidad Rey Juan Carlos, Spain. He is currently an Associate Professor with the School of Engineering, CETYS Universidad. His research activities include the design of algorithms and software development for path planning. His research interests include machine learning, computational and artificial intelligence, parallel and heterogeneous computing, autonomous vehicles, and mobile robots. He is a member of the Research National System from CONACYT.

KENIA PICOS received the Ph.D. degree from the Instituto Politécnico Nacional, Centro de Investigación y Desarrollo de Tecnología Digital. She is currently a full-time Professor with the School of Engineering, CETYS Universidad Campus Tijuana, Mexico. Her current research interests include computer vision, object recognition, three-dimensional object tracking, pose estimation, and parallel computing with graphics processing units. She is a member of the National System of Researchers (Sistema Nacional de Investigadores) from CONACYT.

OSCAR MONTIEL received the M.Sc. degree in digital systems from the Instituto Politécnico Nacional (IPN), in 1999, the M.Sc. degree from the Tijuana Institute of Technology, Tijuana, Mexico, and the Ph.D. degree from the Universidad Autónoma of Baja California, Tijuana, in 2000 and 2006, respectively, both in computer science. He works as a Researcher with the Centro de Investigación y Desarrollo de Tecnología Digital (CITEDI) from the IPN. He has published papers about mobile robotics, evolutionary computation, mediative fuzzy logic, ant colonies, type-2 fuzzy systems, and embedded systems. He received the "Research Award 2016" from IPN. He is a Co-Founder and an Active Member of the Hispanic American Fuzzy Systems Association (HAFSA), and the Mexican Chapter of the Computational Intelligence Society (IEEE). He is a member of the International Association of Engineers (IANG), and also a member of the Mexican Science Foundation CONACYT (Consejo Nacional de Ciencia y Tecnología).