Applied Soft Computing 116 (2022) 108108

Contents lists available at ScienceDirect

Applied Soft Computing

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

Trajectory planning of autonomous mobile robots applying a particle swarm optimization algorithm with peaks of diversity

P.B. Fernandes [a,*], R.C.L. Oliveira [b], J.V. Fonseca Neto [c]

[a] Federal University of South and Southwest of Pará, Marabá, Pará, Brazil
[b] Federal University of Pará, Belém, Pará, Brazil
[c] Federal University of Maranhão, São Luis, Maranhão, Brazil

ARTICLE INFO

Article history:
Received 12 April 2021
Received in revised form 30 September 2021
Accepted 23 October 2021
Available online 16 December 2021

Keywords:
Particle swarm optimization
Trajectory planning
Autonomous vehicles
Optimization
Static and dynamic environments

ABSTRACT

This paper presents a new quantum-behaved particle swarm optimization (QPSO) algorithm for the trajectory planning task of mobile robotic vehicles in static and dynamic environments—it is called enhanced diversity particle swarm optimization (EDPSO). The main characteristic of this algorithm is that it has peaks of diversity in its population, making it possible to escape from local minima effectively, avoiding stagnation. Through the proposed PSO, it is possible to obtain safe and efficient routes, avoiding energy waste and maintaining system integrity in several possible applications. The parameters of the proposed algorithm were tuned using the benchmarking functions. The same functions were used to compare the algorithm with those already established in the literature. Once the proposed algorithm showed promising results, it was simulated in four environments, each with different complexities, presenting dangerous regions and terrains unsuitable for robot navigation, and a large number of obstacles or even moving objects. Further, the algorithms used for comparison were also simulated and the EDPSO presented satisfactory results. Through simulations it was possible to notice that the proposed approach resulted in collision-free and planned routes, and the algorithm presented increased exploration features owing to the diversity peaks that occur during the optimization.

1. Introduction

Autonomous mobile robots have a great variety of applications in several areas of study; such as cinematography, monitoring, research, sports, and agriculture. Technological advances have enabled the acquisition of faster and cheaper microcontrollers, allowing the development of more complex algorithms for the control of robotic vehicle movements, in addition to allowing these vehicles to become autonomous [1].

The first autonomous vehicles were completely reactive; that is, their behaviors depended entirely on the sensor response to achieve their objectives and avoid obstacles. However, for today's desired complex applications with autonomous decision-making, a fully reactive architecture is inadequate. The microprocessor that controls the vehicle must be able to "look ahead" and manage to plan a trajectory even before the robotic vehicle starts moving. This planned trajectory should consider the current state of the vehicle and the navigation environment, leaving the sensors responsible for detecting any change that might modify the initial planning. Thus, besides having a reactive component, the robot must have a deliberative component to carry out prior trajectory planning, in addition to making decisions about when the initial plans should be changed or not, resulting in a hybrid architecture [2].

The navigation environment can be classified into two groups: static and dynamic. Static environments are those where no change in the state of the environment is expected; that is, obstacles, and the final position of a displacement remain immobile during the journey. On the other hand, dynamic environments are those in which there may be movement of obstacles or even the arrival point of the path. Several studies have addressed both types of environments for the development of fast and efficient planners [3,4].

Before going any further, it is better to highlight the definition of a better trajectory, which can be mistaken as the shortest path between two points; however, a slightly deeper analysis reveals some other parameters that should be taken into consideration. In addition to minimizing the distance traveled, the terrain to be used in the trajectory should be analyzed, avoiding unnecessary wear and tear of the vehicle as well as minimizing energy consumption. In some applications, an altitude restriction can be made in the case of unmanned aerial vehicles (UAVs) or

* Corresponding author.
E-mail addresses: pedrobfer@unifesspa.edu.br (P.B. Fernandes), limao@ufpa.br (R.C.L. Oliveira), joao.fonseca@ufma.br (J.V. Fonseca Neto).

https://doi.org/10.1016/j.asoc.2021.108108
1568-4946/© 2021 Elsevier B.V. All rights reserved.

depth restrictions for autonomous underwater vehicles. In some cases, certain areas are considered very dangerous for the vehicle and should be avoided as much as possible. Thus, depending on the case, the definition of the best trajectory should be adapted according to convenience [5].

It is worth noting that there is a differentiation in the literature between the terms trajectory planning and movement planning. As observed by LaValle [6], movement planning focuses on the translation and rotation movements necessary to follow a path. Trajectory planning, in turn, determines a path to be navigated by a vehicle, respecting its movement restrictions. This study fits LaValle's definition for trajectory planning.

Some considerations must be made to simplify the task of the planner to be developed. The vehicle in which this planner is applied is a UAV for filmography purposes. First, sudden changes in direction will be avoided, as applications for large spaces are sought and, in this way, faster navigation with better fluidity in the movement of the vehicle is possible—avoiding unnecessary energy consumption with brakes and accelerations [7]. The mobile robot to be worked on is considered slow, as it will not be necessary for a high speed of translation between two points. In addition, it is considered that the vehicle should maintain a constant altitude, so the planning of trajectories can be performed in two dimensions, resulting in a planner whose application can be extended to autonomous vehicles on land and underwater.

According to Richter et al. [8], in order to result in better trajectories, each route must be planned in two stages: the first one is the planning of a viable path between two points, while the second part consists of the refinement of the result obtained in the first stage—in which the trajectory can be shortened and smoothed for a better performance of the mobile robot during navigation.

To address this challenge, computational intelligence techniques have been studied with promising results in different environments [9]. These algorithms present valuable features, such as high performance, universality, and simplicity. In addition, computational intelligence implementation for path planning has been shown to promote the decentralization of activities involving autonomous mobile robots [10]. Artificial intelligence techniques have some drawbacks for over-simplifying the environments; for example, obstacles and robotic vehicles are usually treated as geometric shapes, and constraints are normally incorporated into the problem by penalization into a single objective function, which might be harmful for the optimization because this approach does not guarantee a feasible path [11]. Despite these disadvantages, these techniques have a large number of applications in the path planning field, and their results continue to improve [12].

The results presented in this paper are a step forward in the research by Fernandes et al. [13], which addresses a deliberative trajectory planner based on artificial intelligence for complex static and dynamic open environments. The algorithm presented is based on particle swarm optimization (PSO), which consists of a population-based evolutionary intelligence technique.

The entire planning process can be summarized in three stages: the treatment of the navigation environment, in which the map of the robot's movement location is modeled, the planning of a valid trajectory between the starting and finishing points of the desired movement, and the post-processing of the determined path, making the vehicle navigate through a smooth, safe, and as short as possible route—respecting the restrictions inherent to the mobile robot. This last stage will not be addressed in this paper.

Here are the main advantages of the proposed algorithm:

- Simple and understandable definition of the navigation environment.
- Adaptability to slight changes in environmental characteristics.
- Optimal solution guarantee.
- Possibility of application for land, air and water vehicles.
- Safe trajectory planning, free from collision.

This paper is organized as follows: Section 2 presents an overview of the subject, surveying recent research in the area of trajectory planning; Section 3 addresses the theory behind the PSO algorithms; Section 4 presents some basic concepts about trajectory optimization starting from the treatment of the map of the navigation environment, to the handling of the constraints for the task; in Section 5, the algorithm proposed in this paper is presented; in Section 6, the results of simulations using the proposed algorithm in various types of environments are presented and compared to the performance of other algorithms already well established in the path planning optimization task; finally, in Section 7, the results are discussed and indications of possible future research are presented.

2. Literature review

Trajectory planning for robotic vehicles or handlers is a very wide field of research, allowing several approaches to address various aspects of this problem. The first techniques to present good results in various applications are sample-based algorithms. There are several algorithms, such as artificial potential fields [14] and visibility graphs [15]. The two mentioned algorithms must present a good representation of the space, usually done through occupation grids [6], which consists of dividing the space into small squares, called cells, allowing the analysis of the passage of the vehicle from one cell to another adjacent until the desired path is created. There are several studies on the treatment of these cells and the cost contained in the passage from one to another [16].

Within sample-based algorithms, there is a large group of probabilistic algorithms, developed mainly during the 1980s, which have the advantage of lower computational cost compared to deterministic algorithms such as those previously mentioned [17]. The central idea of these algorithms is to create several random routes, checking the final cost of choosing the most feasible one. One disadvantage of this approach is that normally the route created is larger than necessary and may present several loops in the same area, and a post-processing stage is usually necessary to improve the resulting trajectory [18]. Among these, probabilistic roadmaps (PRMs) [19] and the rapidly exploring random trees (RRT) [7] are still widely applied.

With the development of more complex programming languages, as well as the acquisition of increasingly powerful microcontrollers, computational intelligence algorithms have begun to gain space in the field of trajectory planning. These algorithms are largely implemented for optimization tasks in many areas such as scheduling problems. Many studies approach algorithms for workflow applications in cloud environments, such as the gray wolf optimizer [20], cuckoo search (CS) [21], particle swarm optimization based algorithms [22], among others, have achieved promising results. There are also applications of meta-heuristic algorithms in power flow for smart grid configurations [23], energy generation [24], and structure analysis of building [25]. To solve path planning problems, numerous studies have applied genetic algorithms (GA), fuzzy logic, ant colony optimization (ACO), and many other nature-inspired algorithms.

Santana et al. [26] developed a trajectory planner for a multi-robotic system based on GAs through an approach that combines solutions for team orienteering and multiple backpack problems. Some studies reveal that the GA presents some limitations in its application to path planning—such as a lack of guarantee to obtain optimal solutions, difficulty in tuning parameters such as mutation rate and population size, and low speed of convergence [1]. To mitigate these disadvantages, Li et al. [27] presented a strategy to speed up the convergence of a GA without reducing the optimization quality. Another modified GA was presented by Jianjun et al. [28], in which the chromosomes size is adaptable, resulting in a lower processing cost. Patle et al. [29] proposed a GA in which the chromosomes are represented by binary matrices for movement with a dynamic point of arrival. Shivgan and Dong [30] modeled the path planning task as a traveling salesmen problem to apply a GA for energy consumption purposes, achieving up to 80% of economy.

Trajectory planning strategies with the application of fuzzy logic have also been successful, similar to the navigation of humanoid robots [31] and underwater vehicles [32]. Neural networks (NNs) also find room for application in this field of study for planning in several environments [33]; but in most applications, NNs are applied jointly with another algorithm for better efficiency [34].

There are many other examples of computational intelligence techniques used for trajectory planning, such as ant colony optimization (ACO) [35], artificial bee colony (ABC) [36], cuckoo search (CS) [37], and the firefly algorithm (FA) [38]. In [4], an ACO algorithm based on the age of the ants was presented to solve the path planning in static and dynamic environments. An ABC global path planner was used in [39] to provide offline information about the points of interest in a dynamic environment. These points will then help another path planner based on Dijkstra's algorithm, which will act online by choosing the best way. Sharma et al. [40] proposed a modified CS algorithm using a tournament for path planning, replacing the concept of random selection. To validate the achieved routes, the length of the path and the estimated time were used as tuning parameters. Li et al. [41] presented an FA with an adjustable population size proportional to the number of obstacles in an environment. They proposed two nonlinear functions to determine the number of fireflies in each iteration. These functions indicate how many fireflies must be added or deleted in order to obtain a better evolution of swarm fitness.

In a recent paper, Fan and Akhter [42] presented a PSO for UAV trajectory planning using a time-varying inertia weight approach, obtaining good results. Song et al. [43] developed a new strategy to plan a smooth path for mobile robots through an improved PSO algorithm in combination with a continuous high-degree Bezier curve. There are also studies applying PSO algorithms along with other artificial intelligence approaches, as in the work by Liu et al. [44], who designed a particle swarm optimization-trained fuzzy neural network algorithm to solve the path planning problem.

The algorithm proposed in this paper addresses static and dynamic scenarios by considering the environment in two dimensions. The QPSO algorithm was chosen for trajectory planning because many studies have shown that it is possible to overcome its convergence disadvantages by making additions and adaptations, such as changing the evaluation function [45] or updating particles by means of classification [46]. All of the aforementioned studies are specific for path planning. The proposed algorithm changes the update process of QPSO by using alternative equations, and will have its results side by side with those of other consolidated algorithms based on artificial intelligence to verify its efficiency.

3. Particle swarm optimization algorithms

3.1. Particles Swarm Optimization (PSO)

The particle swarm optimization (PSO) algorithm is a meta-heuristic based on the social behavior of birds within a flock. This algorithm was proposed in 1995 by Kennedy and Eberhart [47] and quickly became a powerful tool in several optimization tasks.

PSO is based on the creation of an initial random swarm of particles, each containing a solution for the desired optimization task. Each solution contains a fitness value—that is, a value added to the efficiency of the solution—and the algorithm causes the particles with the worst fitness values to be attracted to those with the best ones. In order to avoid premature convergence, all particles contain memory of the best fitness value they have ever achieved, and this information is also used to update the position of particles in the search for a better overall solution.

For the trajectory planning task, each particle has a position vector $x = (x_1 \;\; x_2 \;\; \cdots \;\; x_n)$ in the configuration space, where x_i represents points in the navigation space to be interpolated to form a valid route. For each dimension used in the configuration space, a new position vector must be contained in the particle. In short, there is a position matrix $P = [x \;\; y \;\; z]$ contained in each particle. In addition, the particles have a velocity matrix $V = \left[v_x \;\; v_y \;\; v_z\right]$, which indicates the changing rate of the particle's position in each iteration. When two-dimensional configuration spaces are used then $z = v_z = 0$.

Consider a swarm of N particles with D points to be interpolated in the configuration space; then, the updating of the positions and speeds contained in the particles follows

$$x_{i,d}(k+1) = x_{i,d}(k) + v_{i,d}(k) \tag{1}$$

and

$$v_{i,d}(k+1) = \omega v_{i,d}(k) + c_1 r_1 (pbest_i - x_{i,d}(k)) + c_2 r_2 (gbest - x_{i,d}(k)) \tag{2}$$

where i $(1 < i < N)$ and d $(1 < d < D)$ denote the particle and the point updated during iteration k, respectively. The best solution of a certain particle i is represented by $pbest_i$, and the best solution found among all the particles is called $gbest$. The parameter ω is called the inertia weight, c_1 and c_2 are positive constants, and r_1 and r_2 are random numbers with a uniform distribution between 0 and 1.

3.2. Quantum-behaved Particle Swarm Optimization (QPSO)

The PSO ruled by Eqs. (1) and (2), despite showing good results, often has problems of stagnation at points that may not even be locally optimal [48]. Several studies have addressed techniques in an attempt to improve this problem. According to Clerc and Kennedy [49], convergence in local optima can be guaranteed if all particles are attracted to point p given by

$$p_{i,d}(k) = \varphi pbest_i + (1 - \varphi) gbest \tag{3}$$

where $p_{i,d} = \left(p_{i,1} \;\; p_{i,2} \;\; \cdots \;\; p_{i,D}\right)$ is known as the local attractor of particle i and the constant φ is calculated by

$$\varphi = \frac{c_1 r_1}{c_1 r_1 + c_2 r_2} \tag{4}$$

From Eqs. (3) and (4), Sun et al. [50] developed an algorithm named quantum-behaved particle swarm optimization (QPSO). This algorithm modifies the position update equation of the particle, which is

$$x_{i,d}(k+1) = p_{i,d}(k) \pm L \times \ln\left(\frac{1}{u_{i,d}}\right) \tag{5}$$

where L is a control parameter for the maximum movement a particle can make in an iteration, which must decrease in each iteration—therefore the QPSO has a good performance. To obtain this characteristic, L is given by:

$$L = \alpha \left|x(k) - mbest(k)\right| \tag{6}$$

where α is

$$\alpha = \alpha_1 + \frac{(T-k)\times(\alpha_0-\alpha_1)}{T} \tag{7}$$

The parameter α should decrease from an initial value α_0 to a final value α_1 over the total number of iterations T of the algorithm. In turn, *mbest* is the average of all the best N particle fitness values of the swarm and it is calculated by

$$mbest(k) = \frac{1}{N}\sum_{i=1}^{N} pbest_i \tag{8}$$

As can be observed from Eq. (5), there is no speed update step in the QPSO algorithm because this parameter is controlled by L. Thus, in addition to ensuring convergence, this algorithm also has lower computing costs than the standard PSO, which justifies its popularity in several applications [51]. The QPSO pseudocode is shown in Algorithm 1, and the flowchart is presented in Fig. 1.

The QPSO computational complexity can be given by the time needed to create or update the population (T_p) to evaluate the fitness of each particle (T_f) and the number of iterations (N_{it}). Thus, the computational cost of the QPSO algorithm is given by:

$$C_c = T_p \times T_f \times N_{it}. \tag{9}$$

Algorithm 1 QPSO pseudocode.

```
1:  Procedure QPSO
2:  Initialization: $n \leftarrow$ swarm population ; $t \leftarrow$ number of iterations
3:  Initialize the swarm with random solutions
4:  for $i = 1$ to $t$ do
5:      for $j = 1$ to $n$ do
6:          Calculate $p_{j,d}$ by Equation (3)
7:          Calculate $pbest_j$
8:          if $pbest_j < gbest$ then
9:              $gbest = pbest_i$
10:     Calculate $mbest$ by Equation (8)
11:     Update $x_{i,d}$ by Equation (5)
12: end Procedure
```

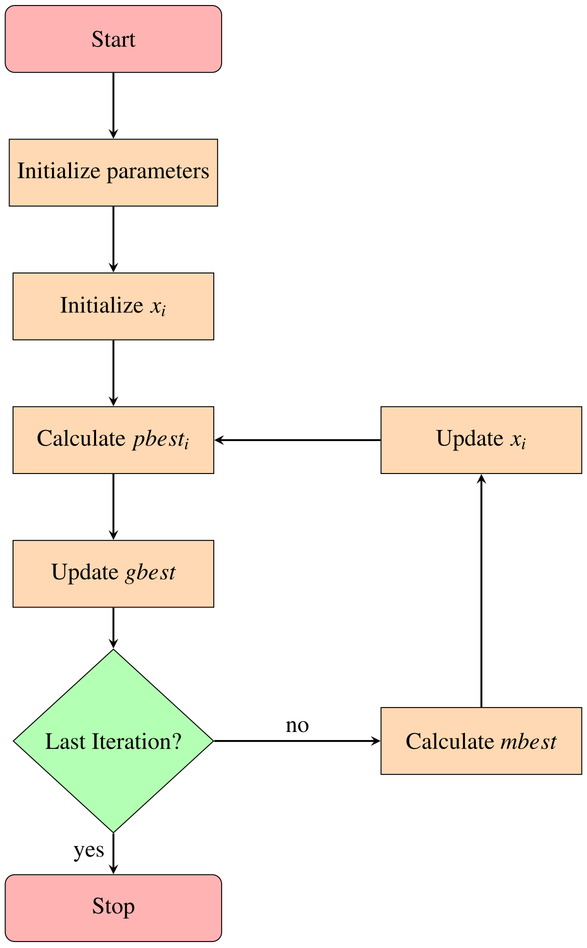

Fig. 1. Flowchart of QPSO algorithm.

4. Optimization approach for trajectory planning

4.1. Configuration space

The configuration space is a representation of the environment to be navigated, the mobile vehicle, and the obstacles in an n-dimensional space, with the objective of allowing the creation of a formulation that is convenient and appropriate for the current problem. In general, for mobile vehicles, the proper manipulation of the configuration space allows the transformation of the mobile robot movement problem into a trajectory planning one for a point that represents the vehicle [52].

In this study, to reduce the complexity of the problem, the vehicle will be treated as a disk-shaped object, so that the configuration space can be a bi-dimensional version of the map of the environment to be navigated, and the vehicle is only one point on this map—following the algorithms described by Latombe [52] and LaValle [6].

Because a metaheuristic is applied, it is necessary to determine the number of points that will be interpolated between the beginning and the end of the movement to generate the path. It is common for this number to be determined empirically, or through the creation of equations [53]. However, in practical applications, a straight line can be the best path in many situations; thus, there is no need for points to be interpolated as it would make the optimization itself unnecessary [13].

To obtain a satisfactory definition of this number of points, Fernandes et al. [13] presented a simple and efficient algorithm in which was stated that the number of points needed for a good optimization is equal to the number of obstacles crossed by a straight line between the starting point and the arrival point of the movement. All of this except when there are fewer than three obstacles crossed by the straight line—in which case n is equal to the number of obstacles plus one. If no obstacles cross the straight line, then it itself is the best path—and no optimization should be performed.

4.2. Cost function

As mentioned in Section 1 of this paper, trajectory optimization is not necessarily a task of looking for the shortest path between two points, although the extent of the route is always taken into consideration. It is also important to analyze other parameters depending on the application of the task. The energy consumption to cross an area, the terrain conditions in a certain region, or the safety and integrity of the vehicle can enter the analysis during the planning of a route.

In this work, three parameters are used for the cost function: the length of the route created, the terrain crossed, and the avoidance of obstacles, according to the following equation:

$$C = \kappa C_l + \pi C_t + \eta C_c; \tag{10}$$

where C_l is the cost of the trajectory length, C_t is the cost for the crossed terrain, and C_c is the cost of collisions with obstacles in the environment. The terms κ, π and η are parameters chosen by the programmer for the costs related to trajectory extent, terrain, and collision, respectively.

In the case of the parameter κ, its goal is simply to provide weight to the cost of trajectory extent in relation to the other parameters, so that the distance of the trajectory may have less or more importance on certain occasions. In this study, the value of κ is one. In turn, π aims to evaluate and penalize a certain terrain; therefore, for a totally flat and intact terrain, $\pi = 1$; for a terrain considered unsuitable for the traffic of the vehicle, π should have a relatively high value. The constant η can also be considered a weight that should be high as avoiding collisions is one of the main objectives for trajectory planning. However, it should be observed that if this parameter has an extremely high value, the optimization task can be impaired because the algorithm would learn very little from invalid solutions (with collisions) [54]. For this research, we used $\eta = 20$, a value found by trials and errors that is high enough to avoid collisions without preventing the algorithm from finding invalid solutions during some iterations.

It is worth noting that the term C_l is simply the extent of the trajectory, while C_t and C_c correspond to the extent of trajectory that passes through a certain terrain and through obstacles, respectively. As the environment has various types of possible terrains, C_t is the result of the sum of the trajectory extent through each terrain multiplied by its penalty factor.

4.3. Constraint handling

In the trajectory optimization task, constraints are represented by collisions because a route that passes through an obstacle is considered invalid. A simple way to treat restrictions would be to use the technique called Death Penalty [54], which basically consists of assigning an extremely high value to η, such as one million. In this way, the algorithm avoids any collisions. However, as already mentioned, such a strategy prevents the algorithm from exploring some spaces in the search scope of the task, avoiding the exploration of regions that might have minimum solutions.

In this study, the technique chosen for use was stochastic ranking [55]. Using this technique, there will be a probability Pr on each iteration that, rather than C being used as a cost function, only the value of C_c is considered. The value of Pr should decrease with each iteration t, because the more advanced the algorithm is, the less relevance is presented in the information of invalid cases. The updating of the value of Pr is given by

$$Pr = 1 - \frac{t}{T} \tag{11}$$

where T is the total number of iterations.

5. The proposed PSO algorithm

In the algorithm proposed in this paper, the position update of a particle is performed through Equation (5), in a similar way to QPSO. The approach, however, proposes the use of the concept of a particle neighborhood for the calculation of the average for the best individual fitness (*mbest*). In Eq. (8), it is verified that *mbest* is updated at each iteration; however, the value is the same for all particles.

To obtain a swarm with greater diversity, enhancing its ability to explore, and even exploit the search region, it is proposed that each particle i has a different *mbest*—defined as the average of the best fitness value among the particle i and its n neighbors. From this point on, this parameter is called $nbest_i$ to avoid any misunderstanding. The formula for $nbest_i$ in use is

$$nbest_i(k) = \frac{1}{n+1}\left(pbest_i + \sum_{c=1}^{n} pbest_c\right) \tag{12}$$

where c is the index of the neighboring particle relative to particle i. As stated in Fernandes et al. [13], such a change will not affect the QPSO convergence guarantee if an adequate value of α is used. Several papers have approached the idea of a neighborhood and presented several configurations for its application [56]. In this study, a cyclic configuration was used.

Another characteristic of the proposed algorithm is that Eq. (5) is used at each iteration in which there is an improvement in the optimization. At other times, another PSO algorithm configuration will run and update the position of the particles using

$$x_{i,d}(k+1) = x_{i,d}(k) - pbest_i + gbest + \rho_i(k) r_j \tag{13}$$

where r_j is defined as a uniform random function ranging from -1 to 1 in every iteration and $\rho_i(k)$ is a scaling factor defined as

$$\rho_i(0) = 1$$
$$\rho_i(k+1) = \begin{cases} \phi, & if \;\; f > f_t \\ \xi, & otherwise \end{cases} \tag{14}$$

Eq. (14) creates a mechanism for adjusting the search space according to the performance of each particle. The term f is the number of consecutive times in which there is no improvement in the fitness value of a particle. If the value of f is greater than the threshold f_t, then the parameter ρ has its value reduced to restrict the exploration scope of the particle—hence, $\phi < 1$. On the other hand, when this threshold is not reached, it is desirable that the search space be expanded—that is, $\xi > 1$. This part of the algorithm is based on the guaranteed convergence particle swarm optimization (GCPSO) [57], which tends to converge rapidly to the local optimum. The proposal of merging this algorithm with the modified QPSO algorithm aims to enhance the advantages of GCPSO in the final stages of optimization, in which its high power of convergence is harnessed at a local level. The modified QPSO algorithm is activated more frequently during the first iterations, where the diversity of the proposed algorithm aims to increase the exploration of the search scope.

Finally, the algorithm causes a renewal of the swarm whenever the optimization stagnates a number of iterations in a row. This renewal consists of inserting random solutions in all the particles, but without changing the values of *pbest* of each particle—as well as the *gbest* of the swarm. This type of tool in the algorithm causes what is known as a swarm explosion; which is responsible for increasing the diversity among the particles when activated. For nomenclature purposes, in this study, the proposed algorithm is called enhanced diversity particle swarm optimization (EDPSO). Algorithm 2 shows the EDPSO pseudocode, and Fig. 2 presents the flowchart for better comprehension.

As for the computational complexity of the EDPSO algorithm; it is given by the time needed to create or update the population (T_p), to evaluate the fitness of each particle (T_f), the number of iterations (N_{it}), the time to update the parameter ρ (T_u), and the time to analyze stagnation and reinitialize the swarm (T_s). This way the computational cost of the EDPSO algorithm is given by

$$C_c = T_p \times T_f \times T_u \times T_s \times N_{it}. \tag{15}$$

Comparing Equations (15) to (9), it is clear that EDPSO has a computational cost larger than that of QPSO. The time T_s varies depending on the need to reinitialize the swarm.

In terms of trajectory planning, this algorithm will result in routes that can be improved through post-processing. This last stage is necessary to reduce the final trajectory slightly, when possible, by increasing the speed of movement and energy savings. This stage will not be detailed in this paper, but it consists of verifying whether the path between each interpolated point can be reduced without causing instability in the movement or sharp curves.

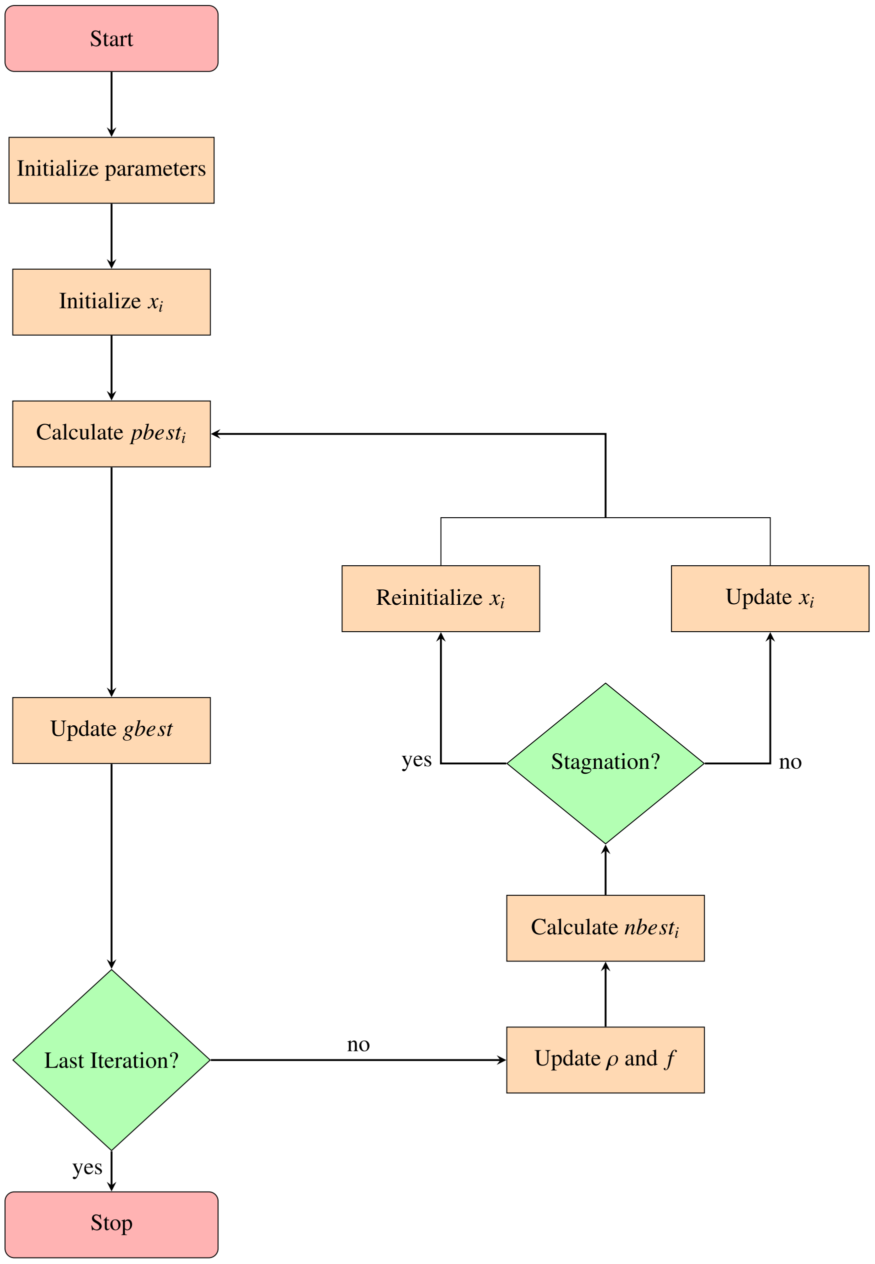

Fig. 2. Flowchart of EDPSO algorithm.

5.1. Tuning EDPSO parameters

To tune all EDPSO parameters, 10 complex multimodel functions were used from the CEC 2019 benchmarking suite [58]. The benchmarking suite can verify the performance of an algorithm systematically using performance indicators. In particular, the indicator of the reciprocal Pareto sets proximity (rPSP) [59] will be used as it provides information about the overlapping rates between the obtained PSs, and the true PSs. It also evaluates the diversity and convergence of a given algorithm.

The parameters were set initially to $\phi = -2$, $\xi = 1$, $f_t = 2$ and $n_s = 6$. Then these values were varied to obtain the best value for each. First the values of ϕ was changed, then, when the best performance was determined, ξ had its value modifies and so on.

Tables 1–4 show the mean of rPSP obtained after 21 runs. In Table 1 it is possible to verify that tuning $\phi = -0.5$ achieves the best results in 9 of the cases, being MMF7 the only exception for which $\phi = -0.2$ was better. The parameter ξ obtained the best rPSP values when it was equal to 2 in eight of the cases. For the threshold f_t, when it is tuned to $f_t = 6$ it gets the better results in six cases against four cases when its value is set to 8. Finally, the better number of iterations without improvement before performing the renewal of the swarm is $n_s = 10$ in eight of the analyzed functions. This way, Tables 1–4 indicate that the better values for the EDPSO algorithm are: $\phi = -0.5$, $\xi = 2$, $f_t = 6$ and $n_s = 10$. The complete CEC 2019 benchmarking suite presents 22 functions and must be tested with 3 other indicators besides the rPSP. We present only the rPSP for 10 functions in this section for organization reasons and because all other tests indicate the same conclusions.

6. Results

To test the algorithm proposed in Section 5, its performance is compared with the AG, CS [37] PSO, QPSO, GCPSO, hybrid

Table 1
Mean rPSP values varying ϕ.

	$\phi = -0.8$	$\phi = -0.5$	$\phi = -0.2$	$\phi = 0.2$	$\phi = 0.5$	$\phi = 0.8$
MMF1	0.1966	**0.1557**	0.1832	0.1779	0.1571	0,2330
MMF2	0.1239	**0.1069**	0.1299	0.1332	0.1567	0.1160
MMF3	0.1936	**0.1421**	0.1907	0.1667	0.1453	0.2184
MMF4	0.2933	**0.2161**	0.2595	0.2705	0.2351	0.2747
MMF5	0.1321	**0.0703**	0.0951	0.0738	0.1156	0.1137
MMF6	0.0568	**0.0472**	0.0667	0.0645	0.0774	0.0793
MMF7	0.0784	0.0529	**0.0459**	0.0627	0.0733	0.0591
MMF8	0.1576	**0.1193**	0.1414	0.1764	0.1576	0.1766
MMF9	0.0512	**0.0371**	0.0415	0.0331	0.0484	0.0431
MMF10	0.0144	**0.0098**	0.0131	0.0104	0.0138	0.0142

Table 2
Mean rPSP values varying ξ.

	$\xi = 1$	$\xi = 2$	$\xi = 3$	$\xi = 4$	$\xi = 5$	$\xi = 6$
MMF1	**0.1434**	0.1489	0.1779	0.1873	0.2065	0.2259
MMF2	0.1155	**0.0862**	0.1396	0.1449	0.1428	0.1943
MMF3	0.1343	**0.1296**	0.1579	0.1708	0.1943	0.2802
MMF4	0.2305	**0.1982**	0.2589	0.2286	0.3367	0.4127
MMF5	0.0948	**0.0682**	0.1060	0.0953	0.1135	0.1349
MMF6	0.0544	**0.0382**	0.0498	0.0764	0.0661	0.0892
MMF7	0.0577	**0.0502**	0.0625	0.0570	0.0823	0.0985
MMF8	0.1739	**0.1320**	0.1502	0.1742	0.2387	0.2565
MMF9	0.0344	0.0341	**0.0333**	0.0433	0.0529	0.0654
MMF10	0.0135	**0.0102**	0.0133	0.0130	0.0141	0.0194

AG-PSO [5],GWO-PSO [60] CMOPSO [61], and EEPSO [13] algorithms. All the cited meta-heuristics were compared using 10 complex multimodel functions from the CEC 2019 benchmarking suite [58].

All the mentioned algorithms had an initial population of 100 particles and there were 100 generations for all simulations. Every parameter is consistent with the original literature, and the algorithm's fitness values are evaluated for each generation every time a particle is updated—that is, 10,000 times per run. All algorithms were run 21 times.

Algorithm 2 EDPSO pseudocode.

```
1: Procedure EDPSO
2: Initialization: $n \leftarrow$ swarm population ; $t \leftarrow$ number of iterations
3: Initialize $f = 0$, $\rho_i = 0$, $f_t$ and $n_s$
4: Initialize the swarm with random solutions
5: for $i = 1$ to $t$ do
6:   for $j = 1$ to $n$ do
7:     Calculate $p_{j,d}$ by Equation (3)
8:     Calculate $pbest_j$
9:     if $pbest_j < gbest$ then
10:      $gbest = pbest_j$
11:      $f = 0$
12:    else
13:      $f = f + 1$
14:    Calculate $nbest_i$ by Equation (12)
15:    if $f > f_t$ then
16:      $\rho_j = -0.5$
17:    else
18:      $\rho_j = 2$
19:    Update $x_{j,d}$ by Equation (13)
20:  if $gbest$ is the same after $n_s$ iterations then
21:    Reinitialize swarm with random solutions
22: end Procedure
```

Table 3
Mean rPSP values varying f_t.

	$f_t = 2$	$f_t = 4$	$f_t = 6$	$f_t = 8$	$f_t = 10$	$f_t = 12$
MMF1	0.2302	0.1459	0.1236	**0.1134**	0.1248	0.2014
MMF2	0.1605	0.1142	0.0847	**0.0833**	0.0886	0.1404
MMF3	0.2359	0.1537	**0.1087**	0.1190	0.1393	0.1851
MMF4	0.3029	0.2270	**0.1544**	0.1827	0.1898	0.2463
MMF5	0.1162	0.0910	**0.0675**	0.0747	0.0829	0.0990
MMF6	0.0796	0.0572	0.0410	**0.0395**	0.0415	0.0659
MMF7	0.0833	0.0532	**0.0355**	0.0401	0.0490	0.0649
MMF8	0.1919	0.1705	**0.0992**	0.1200	0.1290	0.1928
MMF9	0.0477	0.0360	0.0268	**0.0255**	0.0343	0.0373
MMF10	0.0161	0.0118	**0.0082**	0.0095	0.0096	0.0149

Table 4
Mean rPSP values varying n_s.

	$n_s = 6$	$n_s = 8$	$n_s = 10$	$n_s = 12$	$n_s = 14$	$n_s = 16$
MMF1	0.1212	**0.0943**	0.0922	0.1006	0.1355	0.1424
MMF2	0.0737	0.0759	**0.0597**	0.0709	0.0887	0.1001
MMF3	0.1179	0.0927	**0.0852**	0.0962	0.1290	0.1335
MMF4	0.1734	0.1458	**0.1303**	0.1365	0.1807	0.2094
MMF5	0.0689	0.0594	**0.0522**	0.0572	0.0748	0.0818
MMF6	0.0388	0.0318	**0.0317**	0.0346	0.0419	0.0529
MMF7	0.0393	0.0323	**0.0301**	0.0304	0.0449	0.0517
MMF8	0.1124	0.0976	0.0856	**0.0835**	0.1272	0.1457
MMF9	0.0235	0.0231	**0.0218**	0.0222	0.0318	0.0324
MMF10	0.0079	0.0071	**0.0070**	0.0071	0.0100	0.0106

The performance of the analyzed algorithms are verified on the CEC benchmarking suite [58]. The suite contains 22 test functions with different characteristics and is an important tool to verify the performance of the proposed algorithm. In order to use the suite correctly, four distinct indicators must be applied: the reciprocal of Pareto Sets Proximity (rPSP) [59], the reciprocal of Hypervolume (rHV) [58], Inverted Generational Distance (IGD [62]) in decision space (IGDx) and in objective space (IGDf). The indicators rPSP and IGDx are used to evaluate the distribution of the population in the decision space while rHV and IGDf are used to evaluate the distribution of the population in the objective space.

Tables 5–8 record statistical comparison of each algorithm for the four indicators regarding mean value and standard deviation. Better results are shown in bold.

From Table 5, it can be noticed that EDPSO gets the smallest mean rPSP value on 12 functions and is the second best on other five. Table 6 shows that the EDPSO algorithm achieved the smallest rHV mean value on 14 functions and obtains the second best on other 3 functions. Table 7 presents the statistical results the IGDx indicator and again the EDPSO is the algorithm that presents the smallest mean values more often, being the best on 10 functions and the second best on 6. Finally, from Table 8, it can be obtained that the EDPSO presents the best IGDf mean values on 15 functions and is the second best on 4. These results show the effectiveness and superiority of the EDPSO algorithm over the other compared in this study.

As the results in Tables 4–8 show the efficiency of EDPSO in the optimization of benchmarking functions, it is time to verify its performance for trajectory planning tasks alongside the same algorithms compared before. In addition, two sample-based algorithms were simulated for comparison purposes. The chosen algorithms were PRM and RRT.

The PRM algorithm used in this study had 150 configurations on the map, and each was linked to its five nearest neighbors. The RRT, on its turn, performed 1000 iterations with bidirectional search, using a maximum step size of five percent of the perimeter of the navigation space [5].

The genetic algorithms used for the test had an initial population of 150 individuals, with 150 generations for each simulation.

Table 5
Mean and standard deviations of rPSP values obtained by all algorithms.

		EDPSO	EEPSO	CMOPSO	GWO-PSO	AG-PSO	GCPSO	QPSO	PSO	AG	CS
MMF1	mean	**0.0806**	0.1021	0.1836	0.0947	0.2141	0.2238	0.3447	0.2584	0.2779	0.1348
	std. dev.	**0.0121**	0.0128	0.0218	0.0138	0.0271	0.0301	0.046	0.0309	0.0319	0.0184
MMF2	mean	**0.0564**	0.1106	0.1283	0.0975	0.3245	0.1509	0.4293	0.153	0.2382	0.1274
	std. dev.	**0.025**	0.0696	0.0696	0.0645	0.201	0.0916	0.2643	0.0959	0.1488	0.0797
MMF3	mean	0.0759	0.1348	0.1033	**0.0629**	0.1596	0.2629	0.1626	0.0996	0.3331	0.0886
	std. dev.	0.0538	0.0912	0.0601	**0.0431**	0.1048	0.1459	0.1068	0.0606	0.2269	0.0553
MMF4	mean	0.1131	**0.098**	0.131	0.144	0.1985	0.3271	0.5278	0.3997	0.2381	0.151
	std. dev.	0.0345	**0.0276**	0.0372	0.041	0.0559	0.0854	0.1545	0.1266	0.0696	0.0441
MMF5	mean	**0.0456**	0.1155	0.233	0.1635	0.1812	0.2505	0.2825	0.2115	0.2607	0.2575
	std. dev.	**0.0066**	0.0194	0.0336	0.0163	0.0161	0.0225	0.0321	0.0251	0.0529	0.0247
MMF6	mean	**0.0273**	0.0933	0.0312	0.1654	0.1735	0.0289	0.2208	0.2832	0.2007	0.0322
	std. dev.	**0.0045**	0.0277	0.0063	0.0306	0.0313	0.0266	0.0479	0.0545	0.0398	0.0147
MMF7	mean	0.0262	0.0852	0.1418	**0.0034**	0.0044	0.0929	0.1503	0.1462	0.1395	0.0263
	std. dev.	0.0214	0.0791	0.1117	**0.003**	0.0046	0.0845	0.0813	0.1278	0.1294	0.1242
MMF8	mean	0.0742	**0.0424**	0.1078	0.4716	0.4394	0.574	0.5327	0.5241	0.0702	0.5932
	std. dev.	0.0525	**0.0096**	0.0257	0.1542	0.1655	0.1502	0.1201	0.1245	0.0401	0.1689
MMF9	mean	0.0179	0.0602	0.0559	0.0482	0.0722	0.0262	0.0547	0.0555	0.025	**0.0019**
	std. dev.	0.0101	0.0304	0.0295	0.0285	0.0403	0.025	0.0302	0.0337	0.0211	**0.0012**
MMF10	mean	**0.0059**	0.042	0.1114	0.3176	0.1083	0.1057	0.4904	0.5237	0.0864	0.2039
	std. dev.	**0.0062**	0.0317	0.1067	0.3429	0.1115	0.1108	0.2141	0.2505	0.0907	0.2118
MMF11	mean	0.0055	0.0060	0.0070	0.0066	**0.0053**	0.0087	0.0063	0.0102	0.0062	0.0086
	std. dev.	0.0004	0.0004	0.0005	0.0005	**0.0004**	0.0005	0.0007	0.0007	0.0006	0.0004
MMF12	mean	**0.0020**	0.0027	0.0023	0.0026	0.0030	0.0025	0.0044	0.0033	0.0026	0.0027
	std. dev.	**0.0001**	0.0001	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
MMF13	mean	**0.0324**	0.0381	0.0538	0.0511	0.0508	0.0366	0.0495	0.0553	0.0436	0.0559
	std. dev.	**0.0009**	0.0011	0.0014	0.0011	0.0018	0.0017	0.0023	0.0015	0.0015	0.0015
MMF14	mean	0.0528	**0.0467**	0.0589	0.0551	0.0717	0.0856	0.1101	0.0578	0.0680	0.0738
	std. dev.	0.0009	**0.0008**	0.0009	0.0010	0.0012	0.0009	0.0015	0.0012	0.0014	0.0009
MMF15	mean	0.0576	0.0623	**0.0564**	0.0582	0.0765	0.0716	0.0848	0.1031	0.0705	0.0572
	std. dev.	0.0007	0.0009	**0.0011**	0.0010	0.0011	0.0009	0.0010	0.0016	0.0011	0.0010
MMF1_z	mean	0.0409	0.0476	0.0447	**0.0388**	0.0412	0.0564	0.0516	0.0465	0.0605	0.0568
	std. dev.	0.0010	0.0015	0.0014	**0.0013**	0.0019	0.0014	0.0025	0.0015	0.0017	0.0016
MMF1_e	mean	**0.4467**	0.6001	0.4710	0.6608	0.5152	0.7305	0.5451	0.5439	0.8201	0.5055
	std. dev.	**0.1963**	0.2225	0.3317	0.3445	0.2716	0.2339	0.5230	0.3433	0.2702	0.3504
MMF14_a	mean	0.1052	0.1123	0.1343	**0.0962**	0.1036	0.1536	0.1525	0.1812	0.1368	0.1501
	std. dev.	0.0021	0.0027	0.0028	**0.0028**	0.0027	0.0022	0.0033	0.0032	0.0038	0.0021
MMF15_a	mean	**0.0588**	0.0659	0.0926	0.0677	0.0690	0.0831	0.1211	0.1091	0.1080	0.0705
	std. dev.	**0.0015**	0.0019	0.0025	0.0021	0.0023	0.0017	0.0036	0.0026	0.0031	0.0018
SYM-PART simple	mean	**0.0351**	0.0414	0.0506	0.0502	0.0601	0.0603	0.0492	0.0498	0.0622	0.0409
	std. dev.	**0.0236**	0.0231	0.0323	0.0279	0.0248	0.0345	0.0387	0.0432	0.0239	0.0324
SYM-PART rotated	mean	**0.1345**	0.1484	0.1586	0.1805	0.1735	0.1763	0.3033	0.2970	0.1660	0.2197
	std. dev.	**0.3605**	0.3131	0.4458	0.4613	0.3353	0.3277	0.3935	0.4798	0.4719	0.5002
Omni-test	mean	**0.1702**	0.1887	0.2809	0.2472	0.2025	0.2775	0.4262	0.3105	0.2131	0.2384
	std. dev.	**0.0699**	0.0758	0.1040	0.1030	0.1197	0.0851	0.1145	0.1122	0.0910	0.0885

The parent selection operator used was the proportionate selection, and a single-point crossover was performed with chromosomes containing real numbers. The mutation operator used was random and non-uniform, with a 5% probability. Finally, it was decided to use elitism, in which the top 5% (rounded down when necessary) of the best individuals from a population are copied and passed on to the next generation.

The cuckoo search algorithm used 150 nests with one egg each. There was a probability of 25% of the eggs being disposed and the nest being renewed.

In the case of the PSO-based algorithms, all of them contained an initial population of 150 particles and 150 iterations were performed during each optimization. All algorithms were run 100 times for each environment. The GWO-PSO used 75 particles for exploitation, and 75 search agents to explore the search space. All of the parameters for all of the algorithms are summarized in Table 9.

To evaluate the proposed algorithm, four navigation environments were created, as shown in Fig. 3, with varying difficulties. Environments 1 to 3 were static. The blue areas denote static obstacles, while the orange ones represent non-ideal terrains—through which it is possible to travel, but with greater difficulty.

The first environment, shown in Fig. 3(a), presents four static obstacles and two non-ideal areas. For both non-ideal terrains, the penalty parameter was $\pi = 1.5$, which can be interpreted as a mild slope. There is an L-shaped obstacle between the starting point and the target, which will be seen as two different obstacles by the algorithm; for this reason, there are three points to be optimized for by the planner. An example of an efficient trajectory is presented in Fig. 4(a), where it is shown that although there is a clear path between the two non-ideal areas, the algorithm interpreted that it is better to travel through the non-ideal terrain than go around it, since it is going to lengthen the travel distance, resulting in unnecessary energy consumption.

Table 6
Mean and standard deviations of rHV values obtained by all algorithms.

		EDPSO	EEPSO	CMOPSO	GWO-PSO	AG-PSO	GCPSO	QPSO	PSO	AG	CS
MMF1	mean	1.4822	1.4470	**1.4101**	1.4527	1.6357	1.5914	1.9185	1.7867	1.6830	1.5362
	std. dev.	0.0003	0.0004	**0.0014**	0.0016	0.0035	0.0015	0.1132	0.0075	0.0021	0.0032
MMF2	mean	**1.4282**	1.4732	1.4643	1.4370	1.5040	1.4375	1.6426	1.5808	1.4324	1.4836
	std. dev.	**0.0013**	0.0013	0.0014	0.0017	0.0020	0.0018	0.0263	0.0231	0.0014	0.0016
MMF3	mean	**1.2698**	1.3653	1.3055	1.3409	1.3643	1.3204	1.3365	1.3995	1.3791	1.3730
	std. dev.	**0.0009**	0.0009	0.0011	0.0008	0.0008	0.0010	0.0323	0.0015	0.0011	0.0012
MMF4	mean	2.0189	2.0063	2.0071	2.0409	2.0970	2.0209	2.1578	2.0245	**2.0057**	2.0210
	std. dev.	0.0017	0.0015	0.0022	0.0019	0.0035	0.0011	0.0135	0.0029	**0.0026**	0.0017
MMF5	mean	1.2014	1.2093	1.2291	1.2592	1.2117	1.2940	**0.7819**	1.2327	1.2020	1.2809
	std. dev.	0.0017	0.0017	0.0025	0.0015	0.0020	0.0029	**0.2549**	0.0056	0.0025	0.0020
MMF6	mean	**1.2772**	1.5053	1.4575	1.8608	1.6826	1.8619	1.5617	1.7564	1.9435	1.4962
	std. dev.	**0.0012**	0.0011	0.0015	0.0029	0.0025	0.0021	0.0329	0.0031	0.0029	0.0015
MMF7	mean	**1.6959**	1.7384	1.9432	1.8348	1.8173	1.8009	1.7993	1.7903	1.8846	1.7221
	std. dev.	**0.0003**	0.0003	0.0004	0.0003	0.0003	0.0003	0.0016	0.0004	0.0003	0.0004
MMF8	mean	3.0911	3.0135	3.0446	3.1784	3.0847	3.2817	3.1499	3.2063	3.1845	**3.0005**
	std. dev.	0.0025	0.0031	0.0025	0.0031	0.0023	0.0036	1.3909	0.0029	0.0026	**0.0029**
MMF9	mean	**0.1353**	0.1721	0.1659	0.1952	0.1884	0.1635	0.1550	0.1749	0.2139	0.1378
	std. dev.	**0.0001**	0.0001	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
MMF10	mean	**0.0952**	0.1310	0.1245	0.1349	0.1248	0.1266	0.1222	0.2100	0.1145	0.1409
	std. dev.	**0.0001**	0.0001	0.0001	0.0002	0.0001	0.0002	0.0003	0.0002	0.0002	0.0002
MMF11	mean	**0.0961**	0.0989	0.1098	0.1059	0.1483	0.1491	0.1083	0.1529	0.1108	0.0949
	std. dev.	**0.0001**	0.0001	0.0002	0.0001	0.0001	0.0002	0.0003	0.0002	0.0002	0.0002
MMF12	mean	0.9316	0.9770	1.0455	0.9315	1.2206	1.2931	1.2668	1.0870	**0.9070**	1.2049
	std. dev.	0.0001	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0003	**0.0001**	0.0002
MMF13	mean	**0.0617**	0.0913	0.1018	0.0848	0.1065	0.0826	0.1050	0.1002	0.0913	0.0705
	std. dev.	**0.0001**	0.0001	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
MMF14	mean	0.4306	**0.4057**	0.4851	0.4395	0.5077	0.4488	0.5419	0.4726	0.4668	0.4598
	std. dev.	0.0200	**0.0223**	0.0231	0.0276	0.0248	0.0281	0.0458	0.0432	0.0370	0.0321
MMF15	mean	**0.2826**	0.3423	0.2910	0.3762	0.3913	0.3791	0.3283	0.3945	0.3602	0.2988
	std. dev.	**0.0096**	0.0132	0.0125	0.0112	0.0119	0.0167	0.0169	0.0207	0.0138	0.0149
MMF1_z	mean	**1.4848**	1.6136	1.7675	**1.5598**	1.4897	1.5278	1.6609	1.6932	1.4917	1.5369
	std. dev.	**0.0003**	0.0003	0.0003	**0.0003**	0.0004	0.0003	0.0003	0.0005	0.0004	0.0004
MMF1_e	mean	1.4784	1.4970	1.5528	1.5475	1.5110	1.6109	**1.4361**	1.5240	1.5421	1.4827
	std. dev.	0.0125	0.0167	0.0210	0.0186	0.0168	0.0186	**0.3212**	0.0280	0.0191	0.0163
MMF14_a	mean	**0.3748**	0.4299	0.4372	0.4153	0.4423	0.4673	0.5037	0.4458	0.4980	0.4832
	std. dev.	**0.0183**	0.0178	0.0205	0.0201	0.0268	0.0194	0.0347	0.0226	0.0207	0.0247
MMF15_a	mean	**0.2494**	0.2758	0.2908	0.3015	0.3021	0.3045	0.3563	0.3601	0.3279	0.3107
	std. dev.	**0.0085**	0.0102	0.0109	0.0093	0.0135	0.0147	0.0123	0.0200	0.0114	0.0098
SYM-PART simple	mean	0.7841	0.7643	0.7838	0.8504	0.8322	0.8059	0.9508	**0.7674**	0.8272	0.8551
	std. dev.	0.0001	0.0001	0.0002	0.0002	0.0002	0.0002	0.0003	**0.0010**	0.0002	0.0001
SYM-PART rotated	mean	**0.8499**	0.8826	0.9025	0.9530	0.9011	0.8549	0.9320	0.9692	0.9094	0.8507
	std. dev.	**0.0001**	0.0001	0.0001	0.0001	0.0002	0.0001	0.0002	0.0002	0.0002	0.0002
Omni-test	mean	**0.0248**	0.0293	0.0269	0.0254	0.0270	0.0287	0.0249	0.0250	0.0256	0.0286
	std. dev.	**0.0001**	0.0001	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002

In the second simulated environment (Fig. 3(b)), there are six static obstacles and only one bad terrain with $\pi = 2.5$ (bumpy terrain) in an area that could be used to obtain a shorter path. Three static obstacles are positioned between the starting point and the target; thus, it is necessary to use three points for trajectory optimization. As a bad area could threaten the vehicle's integrity, the algorithm avoids choosing a longer path that does not cross this terrain, as shown in Fig. 4(b).

The third scenario is the most difficult among the static environments. There are 19 obstacles and four non-ideal terrains located in areas that can be used to achieve short trajectories. All of these bad areas have $\pi = 5$, which means that these terrains could easily destroy the vehicle, such as a sandy area. There are five points to be interpolated in this case because there are five obstacles between the beginning and the end of the movement. There are a great number of possibilities for designing a path between the starting point and the target, and the one that showed better performance, as seen in Fig. 4(c), crosses the middle of the configuration space, passing near the smaller, non-ideal area.

Environment 4 has two static obstacles and one dynamic obstacle with its initial position shown in red—and its probable future positions are presented in pink with dashed borders. This type of scenario presents a challenge for computational processing, because the vehicle's sensor must detect the direction and velocity of the dynamic obstacle and estimate its future positions—which might have to be recalculated if there is a great modification in the direction of movement of the obstacle. For this simulation, the obstacle is moving downward with half the velocity of the vehicle. The best trajectory for this case is shown in Fig. 4(d). There are three points interpolated because there are two static obstacles between the starting point and the target at the beginning of the movement.

Fig. 4 presents the value of the cost function below each configuration space. To be considered an efficient trajectory, the cost functions for environments 1, 2, 3, and 4 must be less than or equal to 25, 35, 30, and 18, respectively. Fig. 5(a) presents a

Table 7
Mean and standard deviations of IGDx values obtained by all algorithms.

		EDPSO	EEPSO	CMOPSO	GWO-PSO	AG-PSO	GCPSO	QPSO	PSO	AG	CS
MMF1	mean	**0.0541**	0.0547	0.0662	0.0583	0.0554	0.0651	0.0624	0.0902	0.0825	0.0669
	std. dev.	**0.0149**	0.0142	0.0145	0.0148	0.0196	0.0192	0.0177	0.0164	0.0154	0.0169
MMF2	mean	0.0153	**0.0132**	0.0146	0.0159	0.0195	0.0230	0.0266	0.0183	0.0171	0.0197
	std. dev.	0.0020	**0.0029**	0.0025	0.0036	0.0034	0.0024	0.0046	0.0042	0.0040	0.0032
MMF3	mean	0.0116	**0.0113**	0.0123	0.0134	0.0128	0.0178	0.0141	0.0248	0.0141	0.0157
	std. dev.	0.0021	**0.0023**	0.0031	0.0032	0.0030	0.0039	0.0039	0.0027	0.0033	0.0027
MMF4	mean	0.0310	0.0365	**0.0308**	0.0365	0.0473	0.0364	0.0458	0.0660	0.0506	0.0427
	std. dev.	0.0012	0.0015	**0.0016**	0.0017	0.0021	0.0024	0.0029	0.0028	0.0021	0.0017
MMF5	mean	**0.1039**	0.1180	0.1118	0.1424	0.1415	0.1486	0.2079	0.1999	0.1050	0.1261
	std. dev.	**0.0038**	0.0041	0.0049	0.0056	0.0051	0.0072	0.0095	0.0079	0.0054	0.0054
MMF6	mean	0.0859	0.0901	**0.0847**	0.1029	0.1349	0.1263	0.1371	0.1544	0.1287	0.1200
	std. dev.	0.0025	0.0034	**0.0030**	0.0030	0.0043	0.0029	0.0055	0.0044	0.0033	0.0031
MMF7	mean	**0.0281**	0.0373	0.0404	0.0342	0.0288	0.0406	0.0524	0.0428	0.0435	0.0388
	std. dev.	**0.0019**	0.0022	0.0023	0.0020	0.0024	0.0018	0.0031	0.0034	0.0030	0.0025
MMF8	mean	**0.0540**	0.0726	0.0653	0.0743	0.0986	0.0794	0.0741	0.1003	0.1049	0.0664
	std. dev.	**0.0049**	0.0044	0.0044	0.0050	0.0050	0.0058	0.0063	0.0085	0.0073	0.0067
MMF9	mean	0.0048	**0.0043**	0.0046	0.0045	0.0047	0.0061	0.0070	0.0077	0.0054	0.0062
	std. dev.	0.0001	**0.0001**	0.0002	0.0002	0.0002	0.0002	0.0002	0.0003	0.0002	0.0002
MMF10	mean	**0.0024**	0.0026	0.0025	0.0025	0.0040	0.0041	0.0047	0.0052	0.0041	0.0037
	std. dev.	**0.0003**	0.0005	0.0006	0.0005	0.0006	0.0007	0.0006	0.0009	0.0006	0.0004
MMF11	mean	0.0052	**0.0047**	0.0063	0.0057	0.0077	0.0084	0.0102	0.0082	0.0074	0.0076
	std. dev.	0.0001	**0.0001**	0.0001	0.0001	0.0002	0.0002	0.0002	0.0003	0.0002	0.0002
MMF12	mean	**0.0022**	0.0026	0.0032	0.0025	0.0032	0.0024	0.0041	0.0033	0.0030	0.0026
	std. dev.	**0.0001**	0.0001	0.0002	0.0002	0.0001	0.0001	0.0003	0.0002	0.0002	0.0002
MMF13	mean	0.0397	**0.0391**	0.0467	0.0462	0.0552	0.0483	0.0608	0.0598	0.0649	0.0540
	std. dev.	0.0010	**0.0012**	0.0013	0.0012	0.0013	0.0015	0.0021	0.0022	0.0016	0.0015
MMF14	mean	**0.0456**	0.0487	0.0553	0.0702	0.0500	0.0775	0.0762	0.0695	0.0852	0.0567
	std. dev.	**0.0007**	0.0008	0.0009	0.0012	0.0013	0.0009	0.0010	0.0010	0.0009	0.0011
MMF15	mean	0.0557	0.0597	0.0740	**0.0504**	0.0537	0.0770	0.0813	0.0713	0.0721	0.0727
	std. dev.	0.0008	0.0010	0.0012	**0.0011**	0.0013	0.0014	0.0017	0.0015	0.0014	0.0009
MMF1_z	mean	**0.0328**	0.0368	0.0420	**0.0553**	0.0410	0.0642	0.0607	0.0649	0.0516	0.0479
	std. dev.	**0.0013**	0.0011	0.0018	**0.0015**	0.0015	0.0020	0.0021	0.0017	0.0020	0.0015
MMF1_e	mean	**0.3657**	0.3903	0.4121	0.4099	0.5547	0.4828	0.6529	0.6885	0.6093	0.5004
	std. dev.	**0.0875**	0.0836	0.1271	0.0902	0.1456	0.1565	0.2038	0.1927	0.1104	0.0911
MMF14_a	mean	0.1067	0.0989	0.1326	0.1072	0.1218	0.1292	0.1328	0.2008	0.1545	**0.1064**
	std. dev.	0.0020	0.0022	0.0031	0.0031	0.0022	0.0029	0.0030	0.0044	0.0030	**0.0027**
MMF15_a	mean	0.0690	0.0693	0.0797	0.0853	**0.0626**	0.0896	0.0795	0.0818	0.0811	0.0851
	std. dev.	0.0017	0.0022	0.0019	0.0019	**0.0019**	0.0022	0.0036	0.0022	0.0023	0.0019
SYM-PART simple	mean	0.0419	**0.0403**	0.0414	0.0576	0.0497	0.0494	0.0732	0.0854	0.0516	0.0450
	std. dev.	0.0233	**0.0268**	0.0326	0.0241	0.0342	0.0218	0.0285	0.0328	0.0307	0.0263
SYM-PART rotated	mean	0.1449	**0.1367**	0.1390	0.1491	0.2084	0.2039	0.2063	0.1520	0.1833	0.1502
	std. dev.	0.3015	**0.2627**	0.2966	0.2895	0.2829	0.3450	0.4514	0.5588	0.4822	0.3199
Omni-test	mean	**0.2004**	0.2459	0.2615	0.2868	0.2805	0.3108	0.2793	0.2338	0.3105	0.2394
	std. dev.	**0.0730**	0.0829	0.1030	0.1014	0.0997	0.0982	0.1031	0.1605	0.0892	0.1093

satisfactory result for environment 2, and Fig. 5(b) shows a result that, despite being a valid route, is not considered efficient. It can be seen that, while the first trajectory manages to be as direct as possible, the second trajectory contains two unnecessarily long curves, besides passing through an non-ideal terrain.

Table 10 shows the percentage of times that satisfactory and invalid routes were obtained for each environment with static obstacles for comparison. All algorithms were able to perform relatively well in the first environment, which had a lower complexity. In the second scenario, the PRM, AG, CS, PSO, and QPSO algorithms presented great difficulty; while in the third scenario, only the EEPSO and EDPSO achieved a higher rate than 70% to find a trajectory that was considered optimal. The PSO-based algorithms GCPSO, AG-PSO, GWO-PSO, and CMOPSO achieved good optimization in the first and fourth scenarios, but when there were many obstacles, their performance decreased. It is worth mentioning that the EDPSO obtained the best results, as expected, because it is an evolution of the first algorithm.

Table 11 shows the overall performance of all analyzed algorithms, presenting the mean value, standard deviation, and best and worst optimizations. All algorithms were run 100 times to obtain the values. It can be seen that the EDPSO has the best mean values for Scenarios 1 and 3, and is the second best in Scenarios 2 and 4, after GWO-PSO and EEPSO, respectively. In addition, EDPSO has a smaller standard deviation for all scenarios, and its best result is the overall best in Scenario 3, and the second best for all other cases.

Fig. 6 shows a comparison between the convergence of AG, PSO, QPSO, and EDPSO using the second scenario. It should be noted that in the first iteration, EDPSO presents a better result. This is because the other algorithms create the first population randomly in a single batch. In EDPSO, the particles are created in series and use the information provided by the particles that came before. In addition, we can see that EDPSO shows a slow convergence requiring a larger number of iterations than AG and PSO to achieve the final results. The advantage of this characteristic is that the algorithm hardly stagnates, while the AG and

Table 8
Mean and standard deviations of IGDf values obtained by all algorithms.

		EDPSO	EEPSO	CMOPSO	GWO-PSO	AG-PSO	GCPSO	QPSO	PSO	AG	CS
MMF1	mean	**0.0028**	0.0029	0.0030	0.0038	0.0037	0.0051	0.0058	0.0050	0.0050	0.0045
	std. dev.	**0.0001**	0.0002	0.0001	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
MMF2	mean	**0.0051**	0.0067	0.0079	0.0077	0.0086	0.0072	0.0093	0.0121	0.0080	0.0069
	std. dev.	**0.0005**	0.0006	0.0006	0.0007	0.0007	0.0009	0.0009	0.0010	0.0007	0.0007
MMF3	mean	**0.0042**	0.0048	0.0051	0.0066	0.0056	0.0057	0.0089	0.0056	0.0060	0.0068
	std. dev.	**0.0012**	0.0013	0.0013	0.0016	0.0014	0.0016	0.0018	0.0018	0.0020	0.0016
MMF4	mean	0.0035	**0.0030**	0.0036	0.0039	0.0048	0.0032	0.0067	0.0035	0.0044	0.0047
	std. dev.	0.0002	**0.0001**	0.0001	0.0001	0.0002	0.0002	0.0002	0.0002	0.0002	0.0001
MMF5	mean	**0.0034**	0.0037	0.0042	0.0045	0.0035	0.0044	0.0045	0.0065	0.0054	0.0035
	std. dev.	**0.0001**	0.0001	0.0002	0.0001	0.0002	0.0002	0.0002	0.0003	0.0002	0.0002
MMF6	mean	**0.0028**	0.0032	0.0043	0.0043	0.0045	0.0045	0.0052	0.0047	0.0050	0.0033
	std. dev.	**0.0001**	0.0002	0.0002	0.0002	0.0001	0.0002	0.0002	0.0003	0.0002	0.0002
MMF7	mean	**0.0034**	0.0039	0.0035	0.0046	0.0047	0.0037	0.0049	0.0061	0.0042	0.0038
	std. dev.	**0.0001**	0.0001	0.0002	0.0002	0.0001	0.0002	0.0003	0.0003	0.0002	0.0002
MMF8	mean	0.0036	**0.0034**	0.0046	0.0041	0.0054	0.0047	0.0045	0.0070	0.0049	0.0045
	std. dev.	0.0001	**0.0002**	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
MMF9	mean	**0.0139**	0.0154	0.0201	0.0167	0.0197	0.0213	0.0161	0.0279	0.0182	0.0169
	std. dev.	**0.0007**	0.0008	0.0009	0.0008	0.0013	0.0009	0.0011	0.0016	0.0011	0.0009
MMF10	mean	**0.0153**	0.0165	0.0216	0.0227	0.0262	0.0175	0.0320	0.0334	0.0234	0.0214
	std. dev.	**0.0045**	0.0040	0.0047	0.0056	0.0048	0.0045	0.0091	0.0056	0.0071	0.0066
MMF11	mean	0.0170	**0.0150**	0.0163	0.0196	0.0239	0.0203	0.0220	0.0280	0.0259	0.0177
	std. dev.	0.0011	**0.0015**	0.0017	0.0016	0.0019	0.0019	0.0020	0.0030	0.0019	0.0014
MMF12	mean	0.0030	**0.0028**	0.0029	0.0032	0.0041	0.0029	0.0037	0.0041	0.0041	0.0030
	std. dev.	0.0001	**0.0001**	0.0002	0.0001	0.0001	0.0002	0.0003	0.0003	0.0002	0.0001
MMF13	mean	**0.0187**	0.0208	0.0239	0.0283	0.0292	0.0251	0.0226	0.0250	0.0210	0.0200
	std. dev.	**0.0043**	0.0052	0.0061	0.0059	0.0062	0.0074	0.0102	0.0093	0.0075	0.0046
MMF14	mean	**0.1281**	0.1461	0.1353	0.1674	0.1303	0.1368	0.2557	0.1543	0.1671	0.1884
	std. dev.	**0.0015**	0.0018	0.0020	0.0019	0.0015	0.0023	0.0017	0.0033	0.0020	0.0022
MMF15	mean	0.1247	0.1367	0.1287	0.1564	0.1966	0.1659	0.1968	0.2415	0.1986	**0.1209**
	std. dev.	0.0017	0.0020	0.0017	0.0023	0.0022	0.0027	0.0031	0.0032	0.0030	**0.0025**
MMF1_z	mean	**0.0027**	0.0031	0.0046	0.0038	0.0045	0.0040	0.0047	0.0040	0.0036	0.0039
	std. dev.	**0.0001**	0.0001	0.0001	0.0001	0.0002	0.0002	0.0003	0.0002	0.0002	0.0002
MMF1_e	mean	**0.0039**	0.0048	0.0067	0.0055	0.0058	0.0057	0.0090	0.0070	0.0053	0.0055
	std. dev.	**0.0004**	0.0005	0.0005	0.0005	0.0005	0.0006	0.0005	0.0009	0.0006	0.0005
MMF14_a	mean	**0.1231**	0.1243	0.1508	0.1626	0.1379	0.1299	0.1906	0.2634	0.1327	0.1587
	std. dev.	**0.0031**	0.0026	0.0038	0.0041	0.0045	0.0045	0.0055	0.0040	0.0050	0.0034
MMF15_a	mean	0.1311	**0.1236**	0.1612	0.1711	0.1491	0.1557	0.2334	0.1879	0.2033	0.1640
	std. dev.	0.0029	**0.0039**	0.0042	0.0041	0.0053	0.0040	0.0051	0.0053	0.0056	0.0046
SYM-PART simple	mean	**0.0124**	0.0156	0.0151	0.0196	0.0153	0.0237	0.0312	0.0285	0.0215	0.0201
	std. dev.	**0.0016**	0.0020	0.0023	0.0019	0.0022	0.0026	0.0019	0.0027	0.0025	0.0024
SYM-PART rotated	mean	**0.0126**	0.0154	0.0174	0.0179	0.0157	0.0224	0.0191	0.0231	0.0147	0.0140
	std. dev.	**0.0013**	0.0016	0.0020	0.0020	0.0023	0.0018	0.0024	0.0028	0.0019	0.0021
Omni-test	mean	0.0134	**0.0129**	0.0135	0.0193	0.0187	0.0188	0.0266	0.0239	0.0153	0.0136
	std. dev.	0.0007	**0.0006**	0.0010	0.0008	0.0009	0.0010	0.0013	0.0015	0.0009	0.0009

Table 9
Parameters of all algorithms.

	Number of particles (solutions)	generations	other parameters
PRM	-		150 configurations; k=5
RRT	-	1000 iterations	step size = 0.05
CS	150 nests	150	pa = 0.25
AG	150	150	m = 0.05; e = 0.05
AG-PSO	AG = 150; PSO = 50	150	m = 0.05; e = 0.05; c_1=0.4; $c_2 = 0.4$; $\omega = 1$
PSO	150	150	$c_1 = 0.4$; $c_2 = 0.4$; $\omega = 1$
QPSO	150	150	$c_1 = 0.4$; $c_2 = 0.4$; $\alpha_1 = 0.4$; $\alpha_0 = 0.7$
GCPSO	150	150	$c_1 = 0.4$; $c_2 = 0.4$; $f_t = 10$; $s_t = 10$
CMOPSO	150	150	$c_1 = 0.4$; $c_2 = 0.4$; $\omega = 1$
GWO-PSO	GWO = 75; PSO = 75	150	$c_1 = 0.4$; $c_2 = 0.4$
EEPSO	150	150	$c_1 = 0.4$; $c_2 = 0.4$; $f_t = 5$
EDPSO	150	150	$c_1 = 0.4$; $c_2 = 0.4$; $f_t = 2$; $n_s = 6$

PSO present difficulty in improving results after iterations 70 and 20, respectively. As seen in the graphics, this slow convergence comes from the part of the algorithm based on the QPSO, which shows the same characteristic. However, other parameters of the EDPSO, which will be discussed later, avoid the intermittent periods of stagnation observed in QPSO.

Fig. 7 shows the average evolution of the optimization using EDPSO for the other three scenarios. It is observed that, even

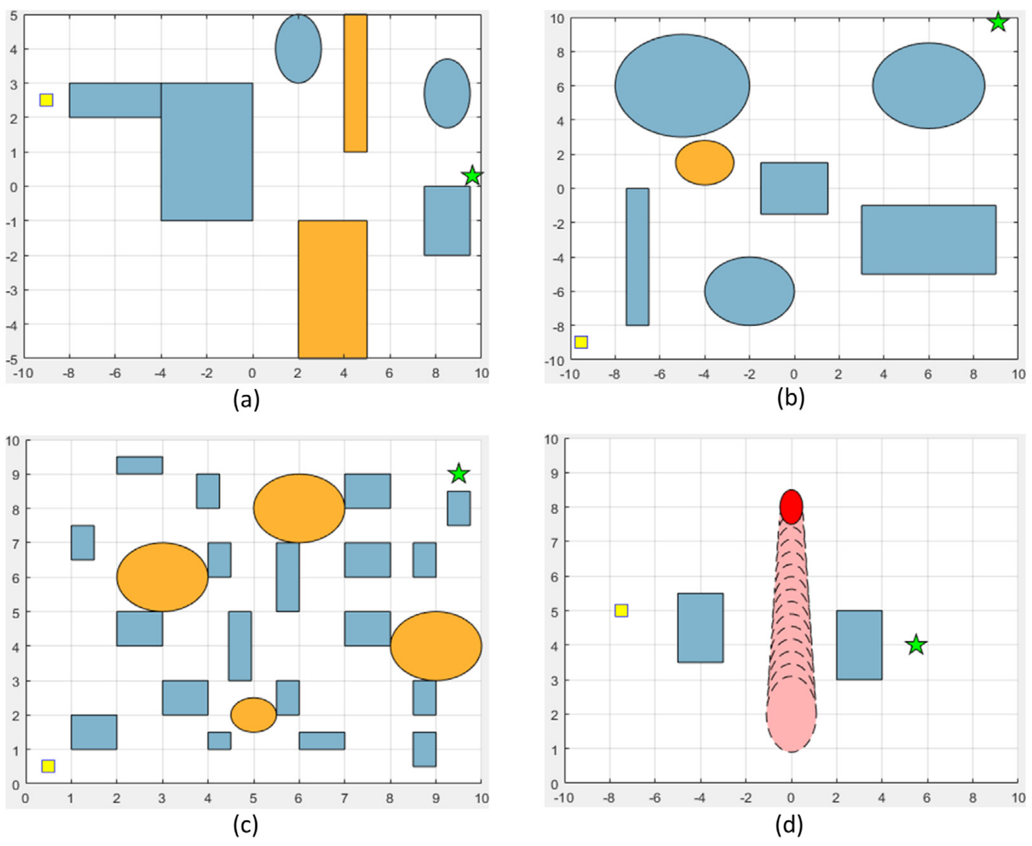

Fig. 3. Configuration space of scenarios used in simulations for trajectory planning: (a) four static obstacles and two areas not ideal for traveling ($\pi = 2$); (b) six static obstacles and one terrain with $\pi = 2$; (c) nineteen static obstacles and four areas with $\pi = 2.5$, and; (d) two static and one dynamic obstacles, the last one moving downwards through the configuration space.

Table 10
Performance of the algorithms tested for all environments analyzed.

	Scenario 1		Scenario 2		Scenario 3		Scenario 4	
	Satisfactory	Invalid	Satisfactory	Invalid	Satisfactory	Invalid	Satisfactory	Invalid
PRM	65%	21%	2%	89%	0%	99%	60%	19%
RRT	71%	2%	62%	9%	5%	91%	65%	9%
AG	73%	19%	23%	51%	18%	49%	67%	7%
CS	80%	17%	51%	20%	42%	36%	75%	5%
PSO	77%	9%	14%	45%	1%	82%	71%	5%
QPSO	89%	3%	8%	58%	1%	74%	73%	4%
GCPSO	89%	5%	58%	24%	6%	76%	75%	5%
AG-PSO	91%	1%	61%	15%	55%	20%	85%	2%
GWO-PSO	85%	7%	65%	9%	62%	32%	87%	1%
CMOPSO	74%	12%	54%	24%	1%	81%	71%	13%
EEPSO	99%	0%	82%	2%	71%	5%	95%	1%
EDPSO	100%	0%	87%	0%	76%	0%	100%	0%

by the definition of this algorithm, the evolution is slow—which necessitates an adequate number of iterations. As it is intended to embed the algorithm in a mobile vehicle, the number of iterations used is considered efficient because there is a dedicated circuit for the planner. Moreover, the performance presented in Tables 10 and 11 justifies this undertaking.

For the fourth scenario, with dynamic obstacles, the various algorithms performed well; however, only EDPSO achieved 100% efficiency in finding a safe trajectory. Fig. 4(d) shows that the algorithm estimates the initial velocity of displacement of the obstacle and traces a route that avoids any chance of collision. In Fig. 6(c), it is observed again that, on average, the algorithm takes time to converge because of its own formulation; however, after an adequate number of iterations, convergence occurs. In this case, it can be seen that 100 iterations would be sufficient for a good performance.

The mean computational times for all algorithms are listed in Table 12. It is shown that the proposed algorithm has a considerable drawback in this parameter when compared to all the other algorithms. However, when the algorithm is embedded into a dedicated circuit, the processing time is approximately ten times faster—which makes the implementation of the EDPSO algorithm possible despite this disadvantage.

To prove the high level of diversity that the proposed algorithm gives to the swarm, a test was performed in which the positions of each particle were mapped and the Euclidean distances between all of them were added up, according to

$$D = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\sqrt{\left(x_i - x^j\right)^2 + \left(y_1 - y_j\right)^2}\right)}{2} \tag{16}$$

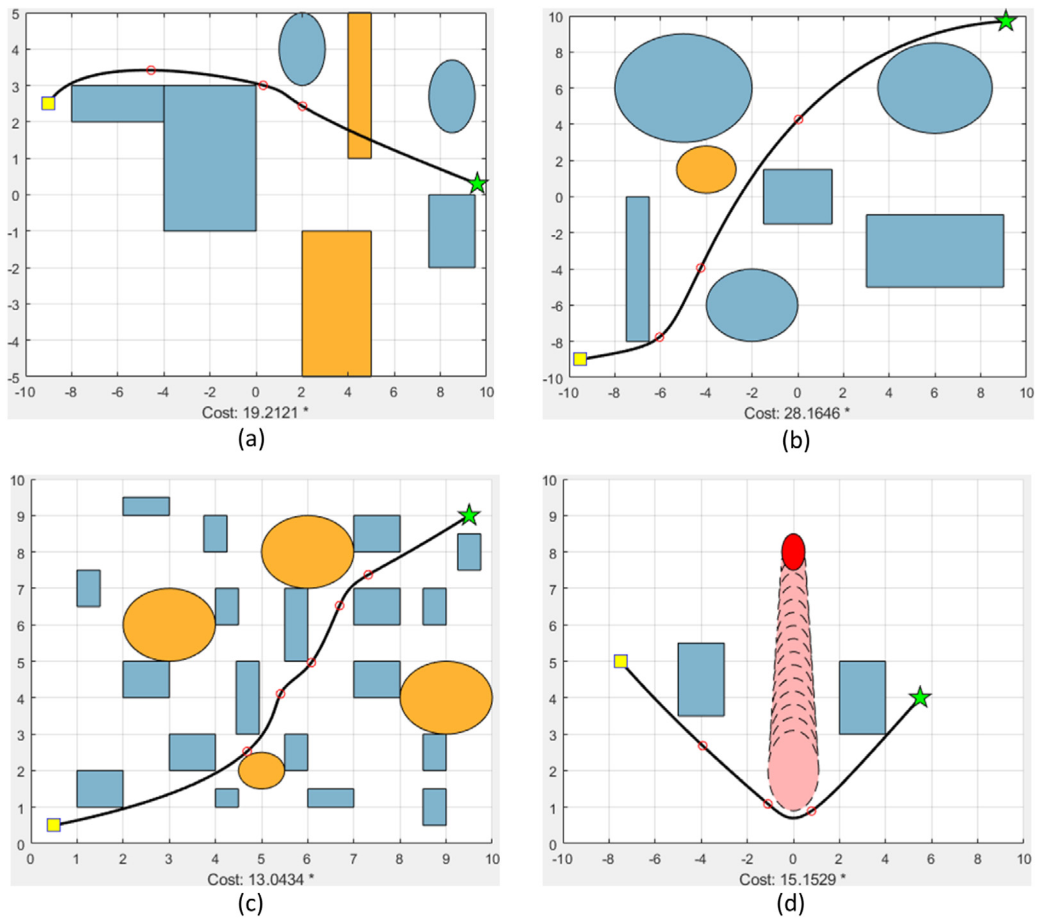

Fig. 4. Optimal trajectories for the four simulated environments.

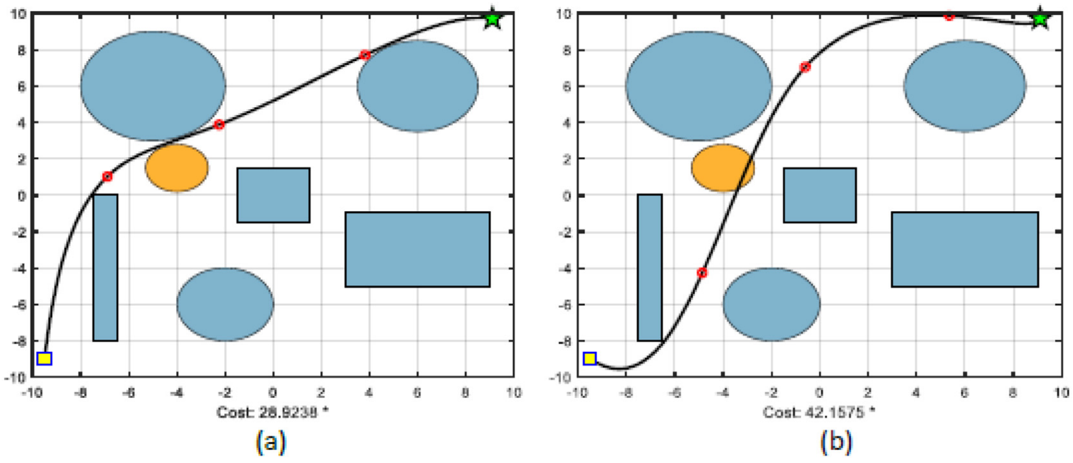

Fig. 5. (a) Trajectory considered as optimal for environment 2, with cost below 35; (b) Trajectory with no collisions, however it is considered inefficient.

The calculation described in Eq. (16) was performed in each iteration so that the graph presented in Fig. 8 could be assembled. It can be observed that the algorithm causes peaks of diversity that occur at each ”restart” of the swarm. It is noticeable that this action of reorganizing the particles does not occur in an exaggerated manner—because as is shown in the case, it occurs three times. Another important characteristic of the diversity of this algorithm is that each peak is followed by a minimum value after a few iterations. Thus, it is observed that this characteristic of the algorithm is quite useful for the adequate convergence of the EDPSO, although it seems contradictory, because it avoids premature stagnation in minimal solutions. For comparison purposes, Fig. 9 shows the evolution of diversity for the PSO and QPSO algorithms. It is noticeable that, although there are diversity oscillations, there is not a great variation as seen in EDPSO.

Table 11
Overall performance of all algorithms for the trajectory planning task for each scenario.

		Scenario 1	Scenario 2	Scenario 3	Scenario 4
PRM	mean	27.577	37.2868	62.2631	16.38
	std. dev.	5.632	8.3003	29.6208	5.3199
	best	20.9671	29.3258	21.005	15.9612
	worst	39.0355	51.5053	100.6467	30.1718
RRT	mean	26.9621	33.7092	56.9528	17.7767
	std. dev.	5.5748	6.5894	35.0449	5.878
	best	19.7004	28.8082	20.4671	16.8938
	worst	48.5481	54.9312	105.7979	56.9671
AG	mean	23.4006	32.7639	45.7252	16.5707
	std. dev.	9.7197	5.426	20.3344	2.9318
	best	19.8041	28.3995	19.4072	15.1595
	worst	80.8935	49.5914	110.7218	34.9994
CS	mean	32.41	40.6353	50.4228	17.9448
	std. dev.	17.1499	5.7241	27.4513	3.1173
	best	18.9682	28.5729	19.0361	15.2437
	worst	84.0398	62.2318	98.3439	35.7724
PSO	mean	31.6061	45.931	56.8944	18.28
	std. dev.	20.5533	7.2559	34.3003	16.5147
	best	19.7475	28.8046	20.9703	15.8419
	worst	74.4503	61.4426	97.8207	38.1878
QPSO	mean	31.139	46.2706	60.1643	18.7012
	std. dev.	44.2385	20.9993	39.1003	21.179
	best	19.0731	29.755	21.2697	15.0898
	worst	92.644	72.3556	110.4398	45.5586
GCPSO	mean	25.335	31.3888	41.7445	16.0048
	std. dev.	8.0368	5.126	16.4275	2.1535
	best	19.2454	28.2115	19.5387	15.2684
	worst	65.3919	47.5905	86.6004	34.0398
AG-PSO	mean	25.4629	31.7553	40.0581	16.258
	std. dev.	7.4956	6.7396	15.0313	2.891
	best	19.8281	28.383	19.5631	15.82
	worst	61.456	48.1553	88.3463	35.265
GWO-PSO	mean	24.1117	**30.2377**	36.9838	15.8708
	std. dev.	5.9712	**3.0706**	11.7207	2.2495
	best	19.5399	**28.1876**	13.5637	16.8405
	worst	49.4423	**40.9689**	60.1428	20.5354
CMOPSO	mean	24.7755	31.7707	39.303	16.7229
	std. dev.	5.6828	5.9866	15.5022	3.7084
	best	19.0869	28.2256	19.5725	16.0294
	worst	50.6628	46.344	65.4915	23.8868
EEPSO	mean	22.1305	30.5773	31.7395	**15.5487**
	std. dev.	3.5376	2.096	10.8629	**1.8549**
	best	19.5938	28.861	15.8629	**15.1995**
	worst	27.7132	35.7642	57.537	**20.8799**
EDPSO	mean	**19.7092**	30.5489	**29.0063**	15.6598
	std. dev.	**0.3957**	1.2554	**6.9354**	0.6667
	best	**19.1967**	28.1646	**13.0434**	15.1529
	worst	**20.1178**	33.1631	**51.8848**	18.1822

Table 12
Mean of computational times for all scenarios.

	Computational Time (seconds)
EDPSO	34.470
GCPSO	30.854
EEPSO	29.926
AG-PSO	29.652
PSO	28.247
GWO-PSO	28.118
CMOPSO	27.583
CS	27.531
AG	27.464
QPSO	27.432
RRT	16.253
PRM	16.231

7. Conclusion

This paper presented a PSO algorithm with diversity peaks to ensure convergence and avoid stagnation of the optimization process, making it easier to find a satisfactory result for the trajectory optimization task.

This algorithm was developed for application in trajectory planning for autonomous vehicles with relatively slow movements; it is suitable for mapping, monitoring, or cinematography in a certain location in addition to several other applications. The EDPSO requires a considerable number of iterations to be performed, since its convergence takes time. Therefore, it should be applied in cases that do not require urgent and rapid travel. For applications that require quick answers, such as rescue missions or navigation through risky locations, faster algorithms are required.

Compared to some established algorithms in trajectory planning, the EDPSO proved to be reliable and efficient, resulting in optimal routes that guarantee safety and energy savings for the

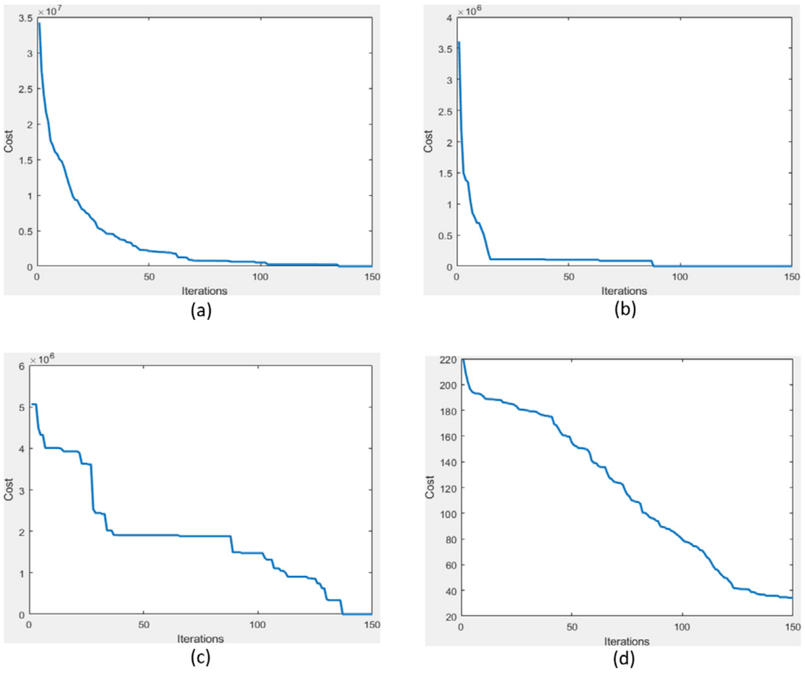

Fig. 6. Convergence average of (a) AG, (b) PSO, (c) QPSO and (d) EDPSO for the second scenario.

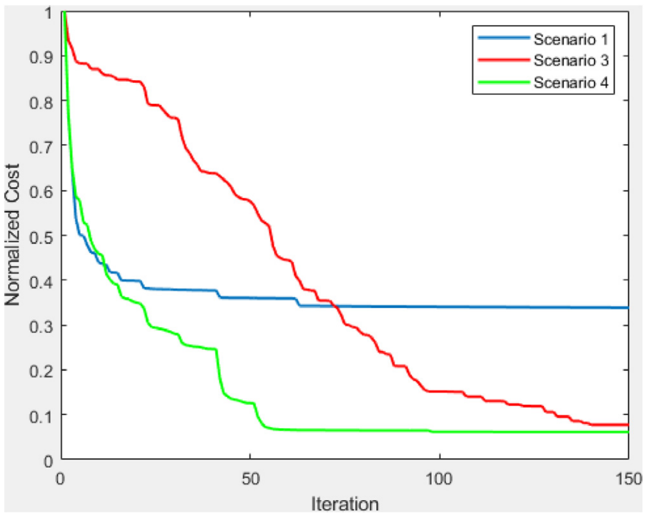

Fig. 7. Convergence average of the EDPSO algorithm for the scenarios 1, 3 and 4.

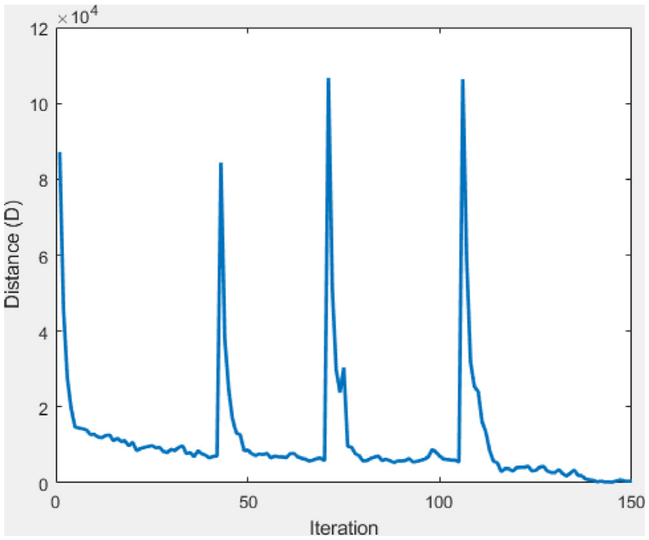

Fig. 8. Evolution of the swarm diversity in the EDPSO algorithm.

autonomous robotic vehicles in which it will be applied. Thus, EDPSO has emerged as a good alternative for the development of mobile robotic vehicles.

The computational cost of EDPSO must be further improved. Table 12 shows that the proposed algorithm takes 25% more time to process than the other analyzed algorithms on average. In future work, we propose that the trajectory planner can be further developed by adding mutation operators. The algorithm must also be tested for multiple mobile vehicles, and with moving target points.

CRediT authorship contribution statement

P.B. Fernandes: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing – original draft, Visualization. **R.C.L. Oliveira:** Data curation, Writing – review & editing, Supervision, Project administration. **J.V. Fonseca Neto:** Resources, Data curation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

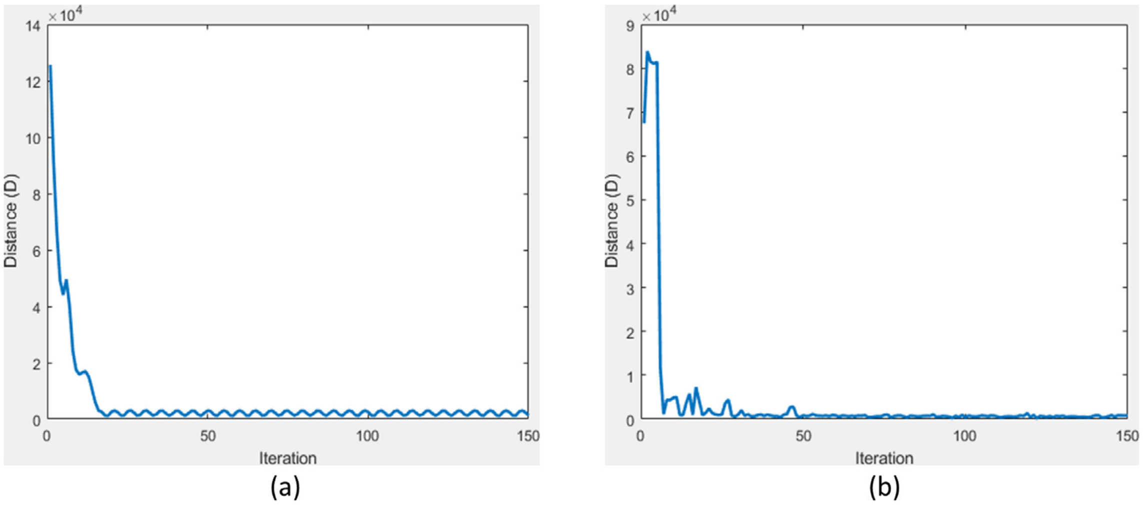

Fig. 9. Evolution of the swarm diversity in the PSO and QPSO algorithms.

References

[1] B.K. Patle, G. Babu, A. Pandei, D.R.K. Parhi, A. Jagadeesh, A review: On path planning strategies for navigation of mobile robot, Déf. Technol. 15 (2019) 582–606.

[2] J.S. Sanchez-Lopez, M. Wang, M.A. Olivares-Mendez, H. Voos, A real-time 3d path planning solution for collision-free navigation of multirotor aerial robots in dynamic environments, J. Intell. Robot. Syst. 93 (2018) 33–53.

[3] L. Zhang, Y. Zhang, Y. Li, Path planning for indoor mobile robot based on deep learning, Optik 219 (2020).

[4] F.H. Ajeil, I.K. Ibraheem, A.T. Azar, A.J. Humaidi, Grid-based mobile robot path planning using aging-based ant colony optimization algorithm in static and dynamic environments, Sensors 20 (7) (2020).

[5] V. Roberge, M. Tarbouchi, G. Labonté, Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning, IEEE Trans. Ind. Inf. 9 (1) (2013) 132–141.

[6] S.M. LaValle, Planning Algorithms, 1st ed., Cambridge University Press, 2006.

[7] B. Song, Z. Wang, L. Zou, L. Xu, F.E. Alsaadi, A new approach to smooth global path planning of mobile robots with kinematic constraints, Int. J. Mach. Learn. Cybern. 10 (2019) 107–119.

[8] C. Richter, A. Bry, N. Roy, Robotics Research, 114, Chapter Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments, Springer International Publishing, 2016, pp. 649–666.

[9] A.T. Salawudeen, P.J. Nyabvo, A.S. Nuhu, E.K. Akut, K.Z. Cinfwat, I.S. Momoh, M.L. Imam, Recent metaheuristics analysis in path planning optimization problems, in: Int. Conf. in Mathematics, Computer Engineering and Computer Science, ICMCECS, Ayobo, Nigeria, 2020, pp. 1–7.

[10] G. Fragapane, R. Koster, F. Sgarbossa, J.O. Strandhagen, Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda, European J. Oper. Res. 294 (2021) 405–426.

[11] X. Zhang, H. Duan, An improved constrained differential evolution algorithm for unmanned aerial vehicle global route planning, Appl. Soft Comput. 26 (2015) 270–284.

[12] G. Flores-Caballero, A. Rodríguez-Molina, M. Aldape-Pérez, M.G. Villarreal-Cervantes, Optimized path planning in continuous spaces for unmanned aerial vehicles using meta-heuristics, IEEE Access 8 (2020) 176774–176788.

[13] P.B. Fernandes, R.C.L. Oliveira, J.V. Fonseca Neto, A modified qpso for robotic vehicle path planning, in: Proc. of the IEEE World Conference on Computational Intelligence, Rio de Janeiro, Brazil, 2018.

[14] Y.K. Hwang, N. Ahuja, A potential field approach to path planning, IEEE Trans. Robot. Autom. 1 (9) (1992).

[15] L. Lacasa, B. Luque, F. Ballesteros, J. Luque, J.C. Nuno, From time series to complex networks: the visibility graph, Proc. Natl. Acad. Sci. 13 (105) (2008) 4972–4975.

[16] S. Thrun, A. Bücken, Integrating grid-based and topological maps for mobile robot navigation, in: Proc. of the 13th National Conference on Artificial Intelligence, Portland, OR, USA, 1996, pp. 944–950.

[17] M.S. Branicky, S.M. LaValle, K. Olson, L. Yang, Quasi-randomized path planning, in: Proc. of the 2001 IEEE International Conference on Robotics and Automation, Vol. 2, Seoul, South Korea, 2001, pp. 1481–1487.

[18] A. Gasparetto, P. Boscariol, A. Lanzutti, R. Vidoni, Trajectory planning in robotics, Math. Comput. Sci. 6 (2012) 269–279.

[19] L.E. Kavraki, P. Svetska, J.C. Latombe, M.H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. 4 (2) (1996) 566–580.

[20] B.H. Abed-Alguni, N.A. Alawad, Distributed grey wolf optimizer for scheduling of workflow applications in cloud environments, Appl. Soft Comput. 102 (2021).

[21] N.A. Alawad, B.H. Abed-Alguni, Discrete island-based cuckoo search with highly disruptive polynomial mutation and opposition-based learning strategy for scheduling of workflow applications in cloud environments, Arab. J. Sci. Eng. 46 (2021) 3213–3233.

[22] A. Pradhan, S.K. Bisoy, A. Das, A survey on pso based meta-heuristic scheduling mechanism in cloud computing environment, J. King Saud Univ. - Comput. Inform. Sci. (2021).

[23] M. Papadimitrakis, M. Giamarelos, M. Stogiannos, E.N. Zois, N.A.-I. Livanos, A. Alexandridis, Metaheuristic search in smart grid: A review with emphasis on planning, scheduling and power flow optimization applications, Renew. Sustain. Energy Rev. 145 (2021).

[24] S.-R. Massan, A.I. Wagan, M.M. Shaikh, A new metaheuristic optimization algorithm inspired by human dynasties with an application to the wind turbine micrositing problem, Appl. Soft Comput. 90 (2020).

[25] G.F. Gomes, F.A. Almeida, Tuning metaheuristic algorithms using mixture design: Application of sunflower optimization for structural damage identification, Adv. Eng. Softw. 149 (2020).

[26] K.A. Santana, V.P. Pinto, D.A. Souza, Information technology and systems, 1137, chapter multi-robots trajectory planning using a novel GA, in: ICITS 2020: Information Technology and Systems, in: Advances in Intelligent Systems and Computing, 2020, pp. 353–363.

[27] Y. Li, Z. Huang, Y. Xie, Path planning of mobile robot based on improved genetic algorithm, in: Proc. of the 3rd International Conference on Electronic Device and Mechanical Engineering, ICEDME, Suzhou, China, 2020, pp. 691–695.

[28] N. Jianjun, W. Kang, H. Haohao, W. Liuying, L. Chengming, Robot path planning based on na improved genetic algorithm with variable length cromossome, in: Proc. of the 12th Int. Conf. on Natural Computation, Fuzzy Systems and Knowledge Discovery, Changsha, China, 2016, pp. 145–149.

[29] B.K. Patle, D.R.K. Parhi, A. Jagadeesh, S.K. Kashyap, Matrix-binary codes based genetic algorithm for path planning of mobile robot, Comput. Electr. Eng. 67 (2018) 708–728.

[30] R. Shivgan, Z. Dong, Energy-efficient drone coverage path planning using genetic algorithm, in: IEEE 21st International Conference on High Performance Switching and Routing, HPSR, Newark, NJ, USA, 2020, pp. 1–6.

[31] A.K. Rath, D.R.K. Parhi, H.C. Das, M.K. Muni, P.B. Kumar, Analysis and use of fuzzy intelligent technique for navigation of humanoid robot in obstacle prone zone, Déf. Technol. 14 (6) (2018) 677–682.

[32] X. Li, W. Wang, J. Song, D. Liu, Path planning for autonomous underwater vehicle in presence of moving obstacle based on three inputs fuzzy logic, in: Proc. of the 4th Asia-Pacific Conf. on Intelligent Robot Systems, Nagoya, Japan, 2019, pp. 265–268.

[33] M. Pflueger, A. Agha, G.S. Sukhatme, Rover-irl: Inverse reinforcement learning with soft value iteration networks for planetary rover path planning, IEEE Robot. Autom. Lett. 4 (2) (2019) 1387–1394.

[34] W. Khaksar, T.S. Hong, K.S.M. Sahari, M. Khaksar, J. Torresen, Sampling-based online motion planning for mobile robots: utilization of tabu search and adaptive neuro-fuzzy inference system, Neural Comput. Appl. 31 (2019) 1275–1289.

[35] S. Kumar, K.K. Pandey, M.K. Muni, D.R. Parhi, Innovative product design and intelligent manufactured systems, in: Chapter Path Planning of the Mobile Robot using Fuzzified Advanced Ant Colony Optimization, 2020, pp. 1043–1052.
[36] A. Nayyar, N.G. Nguyen, R. Kumari, S. Kumar, Frontiers in intelligent computing: Theory and applications, in: Advances in Intelligent Systems and Computing, 1014, Chapter Robot Path Planning using Modified Artificial Bee Colony Algorithm, 2020, pp. 25–36.
[37] P. Song, J. Pan, S. Chu, A parallel compact cuckoo search algorithm for three-dimensional path planning, Appl. Soft Comput. 94 (2020) http://dx.doi.org/10.1016/j.asoc.2020.106443.
[38] L. Tighzert, C. Fonlupt, B. Mendil, A set of new compact firefly algorithms, Swarm Evol. Comput. Base Data (2018) http://dx.doi.org/10.1016/j.swevo.2017.12.006.
[39] R. Szczepanski, T. Tarczewski, Global path planning for mobile robot based on artificial bee colony and dijkstra's algorithm, in: 2021 IEEE 19th International Power Electronics and Motion Control Conference, PEMC, Gliwice, Poland, 2021, pp. 724–730.
[40] K. Sharma, S. Singh, R. Doriya, Optimized cuckoo search algorithm using tournament selection function for robot path planning, Int. J. Adv. Robot. Syst. 18 (3) (2021) 1–11.
[41] F. Li, X. Fan, Z. Hou, A firefly algorithm with self-adaptive population size for global path planning of mobile robot, IEEE Access 8 (2020).
[42] M. Fan, Y. Akhter, A time-varying adaptive inertia weight based modified pso algorithm for uav path planning, in: 2021 2nd Int. Conf. on Robotics, Electrical and Signal Processing Techniques, ICREST, Dhaka, Bangladesh, 2021, pp. 573–576.
[43] B. Song, Z. Wang, L. Zou, An improved pso algorithm for smooth path planning of mobile robots using continuous high-degree bezier curve, Appl. Soft Comput. 100 (2021).
[44] X. Liu, D. Zhang, J. Zhang, T. Zhang, H. Zhu, A path planning method based on the particle swarm optimization trained fuzzy neural network algorithm, Cluster Comput. (2021).
[45] C. Huang, J. Fei, W. Deng, A novel route planning method of fixed-wing unmanned aerial vehicle based on improved qpso, IEEE Access 8 (2020) 65071–65084.
[46] C. Liu, Y. Wang, Y. Gu, J. He, H. Tong, H. Wang, Uuv path planning method based on qpso. Global Oceans 2020: Singapore - U.S. Gulf Coast, 2020, pp. 1–5.
[47] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proc. of the IEEE Int. Conf. on Neural Networks, Vol. 4, Perth, Australia, 1995, pp. 1942–1948.
[48] F. van den Bergh, An Analysis of Particle Swarm Optimizers, (Ph.D. thesis), University of Pretoria, Pretoria, South Africa, 2002.
[49] M. Clerc, J. Kennedy, Paththe particle swarm – explosion, stability and convergence in a multidimensional complex space, IEEE Trans. Evol. Comput. 5 (1) (2002) 325–331.
[50] J. Sun, B. Feng, W. Xu, Particle swarm optimization with particles having quantum behavior, in: Proc. of IEEE Congress on Evolutionary Computing, Portland, OR, USA, 2004, pp. 325–331.
[51] A. Nabaei, M. Hamian, M.R. Parsaei, R. Safdari, H.T. Samad-Soltani, Zarrabi, A. Ghassemi, Topologies and performance of intelligent algorithms: a comprehensive review, Artif. Intell. Rev. 49 (2018) 79–103.
[52] J.C. Latombe, Robot Motion Planning, Kluwer Press, 1991.
[53] S. Hosseininejad, C. Dadkhah, Mobile path planning in dynamic environment based on cuckoo optimization algorithm, Int. J. Adv. Robot. Syst. (2019) 1–13.
[54] E. Mezura-Montes, C.A.C. Coello, Constraint-handling in nature-inspired numerical optimization: Past, presente and future, Swarm Evol. Comput. 1 (4) (2011) 173–194.
[55] T.P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary optimization, IEEE Trans. Evol. Comput. 4 (3) (2000) 284–294.
[56] D. Freitas, L.G. Lopes, F. Morgado-Dias, Particle swarm optimisation: A historical review up to the current developments entropy 2020, Intell. Tools Appl. Eng. Math. 22 (3) (2020) 362–397.
[57] F. van den Bergh, A. Engelbrecht, A new locally convergent particle swarm optimizer, in: Proc. of IEEE Conf. on Systems, Man and Cybernetics, Vol. 3, Hammamet, Tunisia, 2002, pp. 94–99.
[58] J.J. Liang, B.Y. Qu, D.W. Gong, C.T. Yue, Problem definitions and evaluation criteria for the cec 2019, in: Proceedings of the Computational Intelligence Laboratory, Zhengzhou University, China, 2019.
[59] C. Yue, B. Qu, K. Yu, J. Liang, X. Li, A novel scalable test problem suite for multimodal multiobjective optimization, Swarm Evol. Comput. 48 (2019) 62–71.
[60] F. Gul, W. Rahiman, S.S.N. Alhady, A. Ali, I. Mir, A. Jalil, Meta-heuristic approach for solving multi-objective path planning for autonomous guided robot using pso-gwo optimization algorithm with evolutionary programming, J. Ambient Intell. Humaniz. Comput. (2020).
[61] T.T. Mac, C. Copot, D.T. Tran, R. Keyser, A hierarchical global path planning approach for mobile robots based on multi-objective particle swarm optimization, Appl. Soft Comput. 59 (2017) 68–76.
[62] A. Zhou, Q. Zhang, Y. Jin, Approximating the set of pareto-optimal solutions in both the decision and objective spaces by an estimation of distribution algorithm, IEEE Trans. Evol. Comput. 13 (5) (2009) 1167–1189.