Received: May 28, 2022. Revised: July 17, 2022.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Robot Path Planning in Unknown Environments with Multi-Objectives Using an Improved COOT Optimization Algorithm

Jaafar Ahmed Abdulsaheb[1,2*] **Dheyaa Jasim Kadhim[1]**

[1] *Department of Electrical Engineering, College of Engineering, University of Baghdad, Baghdad, Iraq*
[2] *Department of Electronics and Communication, College of Engineering, Uruk University, Baghdad, Iraq*
* Corresponding author's Email: jaafer@uruk.edu.iq

Abstract: This paper discusses the best path planning algorithm for an autonomous mobile robot in an unknown environment with irregular static and dynamic obstacles and a static and dynamic target based on the improved COOT (ICOOT) optimization algorithm. The ICOOT overcomes the drawbacks of unstable searches in the conventional COOT. The path planning problem is solved by finding the collision-free path between multi-objective shortest path and smoothness. The ICOOT tries to imitate the real world by adding the mobile robot's actual size and the kinematic model with specifications for mobile robots. In order to evaluate the proposed algorithm, thirteen benchmark test functions are used to make a comparison with 30, 100, and 500 dimensions. To test the efficiency of the proposed technique, results are compared with five swarm optimization algorithms. The standard deviation results show that the proposed algorithm gets the best results in 84% of the thirteen test functions for 30 dimensions and 92% for 100 and 500 dimensions. Also, in four complex maps (10×10) m, the mean results show that this method is very useful for robot paths from the start to the target, and the mean distance for ten runs is 13.3797 m for map 1, 13.5164 m for map 2, and 11.9312 m for map 3, and 16.3937 m for map 4. It showed how well it could move quickly and easily around both fixed and moving obstacles.

Keywords: Robot path planning, Multi-objectives, COOT optimization algorithm, Shortest distance, Path smoothness, Obstacle detection, Obstacle avoidance.

1. Introduction

Autonomous mobile robot (AMR) navigation is used in an enormous range of applications, including mining, search and rescue, military, agricultural, healthcare, and entertainment [1]. Three key concerns must be addressed when dealing with robot navigation (RN) issues: safety, accuracy, and efficiency. Finding a collision-free path and following the precise addressed path are the safety and accuracy concerns. Efficiency means that the algorithm doesn't let robots stop and turn over and over again. This wastes time and energy. RN problems can be classified into localization, path planning, cognitive mapping, and motion control. Among these problems, it can be said that path planning is the most important point in the RN. The goal of path planning is to find the best and most direct path that is free of collisions from a start position to a target in a given environment. Generally, there are several ways for a robot to reach a target, but the best path is chosen based on a set of criteria [2].

In the 1960s, the field of robot path planning (RPP) was started, and numerous methodologies such as cell decomposition [3], roadmap approaches [4], and potential fields [5] have been proposed. The main disadvantages of the above algorithms are their inefficiency (the cost of computation is considerable) and inaccuracy (a significant risk of becoming trapped in relative minima). Different heuristic methods, such as the use of neural networks, genetics, and nature-inspired algorithms, can be used to overcome the limitations of these algorithms [6].

Depending on the environment where the robot is situated, RPP may be classified into two categories: static (environment with fixed obstacles) and dynamic (the environment has moving obstacles). Each of these two categories could be divided further into subgroups, global path planning (GPP), where the entire information of fixed and moving obstacles can be known ahead of time; Thus, the GPP can be prepared before the robot begins to move (offline), and there is a local path planning (LPP). Here, it is not possible to get information about the environment in advance. So, there are sensors (online) that collect information about what is around the mobile robot as it moves through the world [7].

Some of the related works are described below. In [8] it was proposed to improve the traditional ACO to avoid being trapped in local minimums or the slower convergence during the process of path planning. In [9], a new strategy based on adaptive particle swarm optimization (APSO) is described for solving the problem of mobile RPP. The APSO algorithm is more intelligent than the traditional PSO algorithm and is commonly utilized to solve real-time problems. It is hoped that a new strategy will help the robot avoid obstacles and get to its goal faster. In the work [10], a group of mobile robots presents a unique odor source localization approach based on the cuckoo search algorithm. It employs a robot that identifies the highest gas concentration among all robots in order to send other robots to look for odor sources upwind. The robots can use this method to get away from both local high concentration and eddy locations. The research [11] develops a new algorithm based on the bacterial foraging optimization (BFO) technique. It uses particles that are spread out in a circle around the robot to figure out a way to get to the target and avoid the obstacles in its way. The goal of the paper [12] is to plan a path for mobile robots that avoids obstacles based on the artificial immune algorithm (AIA) based on the principle of immunity. Simulated results show that the mobile robot can avoid obstacles, escape traps, and reach its goal by using AIA. The goal of this study is to use the artificial immune algorithm (AIA) created from the immune principle to plan an obstacle-avoiding path for mobile robots. Simulated results show that the mobile robot can avoid obstacles, escape traps, and reach its goal by using AIA. A novel multi-objective method for optimal mobile robot path planning based on the Whale optimization algorithm (WOA) has been proposed [13]. In WOA, the distance and smoothness of the robot's path planning issues are transformed into minimization ones. In each iteration, the robot chooses the best whale and moves toward it in order. GA has been widely applied to path optimization problems [14]. The new proposed crossover operator avoids premature convergence and makes possible paths that have better fitness values than their parents. A fuzzy logic controller [15] has been implemented in I robot create (a mobile robot) by interfacing with the arduino uno. When the robot moves, fuzzy rules are used to control how fast the robot moves on its left and right wheels. The hybrid multi-objective bare-bones particle swarm optimization with differential evolution [16] is used to plan better routes for mobile robots. A new Pareto domination with collision constraints has been developed to select the personal best position of a particle. Simulation results confirm the effectiveness of this algorithm. An improved chicken swarm optimization algorithm (ICSO) [17] is proposed and applied in RPP. The numerical results show that the ICSO For unconstrained optimization, the algorithm has better accuracy and stability and has a stronger search capability in RPP. An optimal path planning algorithm based on the firefly algorithm with self-adaptive population size is proposed in [18]. The feasible solution and the infeasible solution are distinguished by population size. In terms of stability, convergence speed, and running time, the proposed algorithm is better than the fixed population size firefly algorithm. The Morphin algorithm [19] was introduced in order to avoid moving obstacles in real time. Simulation results indicate that the proposed method performs well in planning an initial static optimal path. [20] The authors offer a dynamic window technique based on the improved ant colony (IACO-DWA) method for designing an adaptive distance induction factor and combining the maximum and minimum ant systems. Simulated results show that the strategy improves the global path optimization performance while also avoiding local dynamic obstacles, which is what it does. [21] This paper presents an improvement to the actual output trajectory tracking performance of a mobile robot based on a convolutional neural network controller with off-line and on-line tuning back-propagation algorithms. The task of the proposed feedback (CNNTT) controller is to obtain precisely and quickly the robust left and right wheel velocity, which are used to control the position and orientation of the mobile robot system. These algorithms are simulated by MATLAB in a fixed obstacle environment to show the effectiveness of the hybrid swarm optimization algorithm. The work [22] aims at developing a path planning method that provides the shortest path with collision avoidance. The generated hybrid method is called the quarter orbits particle swarm optimization (QOPSO) algorithm. It combines two algorithms to get a more efficient, smoother, and shorter path for the mobile robot. [23] The collision avoidance method is based on velocity

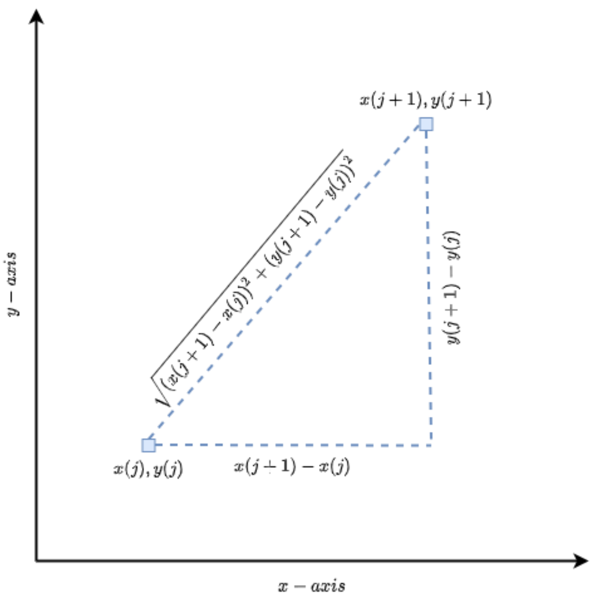


Figure. 1 The Euclidean distance between two points

control with respect to several moving objects in the vicinity of a wheeled mobile robot. Collision avoidance is achieved by considering static robots, other mobile robots, and moving objects with nonlinear trajectories.

The mobile robot was treated as a basic particle in the experiments stated above, which is one of their flaws. While some of these algorithms focused on minimizing the length of the path while avoiding static obstacles, other studies concentrated on minimizing the length of the path while avoiding dynamic obstacles without taking the smoothness of the path into account. The grid-based methods utilized in some of the aforementioned research are also simple to implement, but they come with a number of drawbacks, such as an imperfect representation of the obstacle that reserves the entire cell even if the obstacle only fills a small portion of the cell. Space is wasted as a result, and surroundings that are dynamic have less flexibility. The following is a list of the major contributions made by this research project:

(1) The main drawback in the conventional COOT algorithm is the absence of the parameter of transfer from exploitation and exploration. This results in unstable searches (stagnation in the local optimum) and the squandering of more time. So, the ICOOT algorithm was proposed to overcome this drawback.
(2) Combinations of multi-objective proposed in this paper (shortest path and smoothness), are generated and selected using this algorithm. Also, thirteen benchmark test functions were used to compare the new algorithm to the old one.
(3) The proposed ICOOT algorithm is integrated with a local search strategy that transforms infeasible solutions into viable ones in an unknown environment with irregular static obstacles and a static and dynamic target.
(4) 80 sensors to detect obstacles. The sensors are positioned in all the mobile robot's surroundings, and when they are found, an algorithm is activated that increases or decreases the speed of the robot to avoid them.
(5) Also taken into consideration are the mobile robot's actual size and the kinematic model with specifications for robots (taking turtlebot3 burger as an assumption).

This paper is organized as follows: section 2 represents the problem formulation. Section 3 discusses the COOT optimization technique, and section 4 describes our proposed improved COOT algorithm, while section 5 shows the simulation result and discussion. Finally, the conclusion of this paper is represented in section 6.

2. Problem formulation

Assume a mobile robot is moving from a starting position to a goal in an environment that has both dynamic and fixed obstacles. RPP can be thought of as an optimization problem in which the goal is to find the best path according to some objectives. These objectives are the safest, shortest, and smoothest.

2.1 Shortest distance

The first objective is to figure out the shortest distance between the robot's starting point and its goal. The Euclidian distance f_1 (x,y) is the objective function for minimizing the distance between the present position of the coot and the goal point, formulated as [24]:

$$f_1(x,y)=\sum_{j=1}^{n-1}\sqrt{(\Delta x)^2+(\Delta y)^2} \qquad (1)$$

Where n represents the number of via points (trajectory change occurred) which represent a project parameter of this case and:

$$\Delta x = x(j+1)-x(j) \qquad (2)$$

$$\Delta y = y(j+1)-y(j) \qquad (3)$$

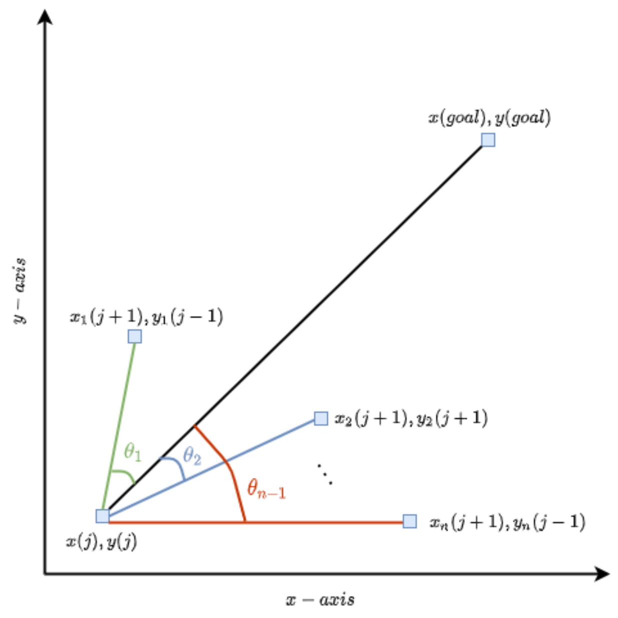


Figure. 2 Path smoothness

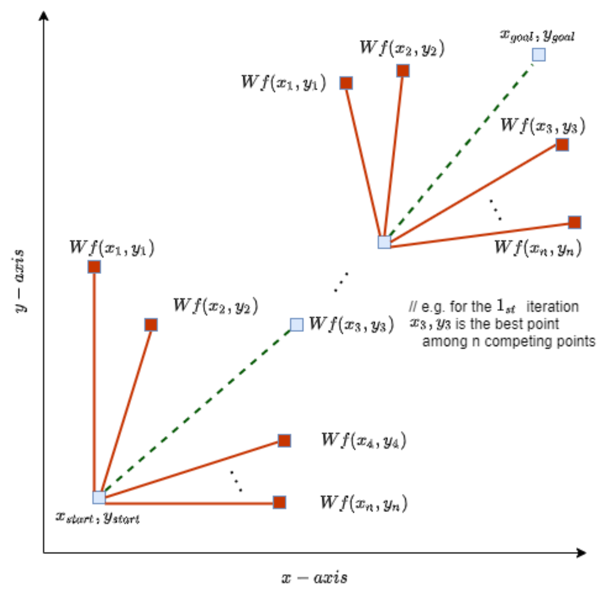


Figure. 3 Weighted sum selections for MOO

2.2 Path smoothness

To enhance the smoothness of the robot's path, minimizing the angles between the straight lines connecting the goal in relation to the current position and the next suggested position in relation to the current position, as shown in Fig. 2. would be a second objective function (Eqs. (4) and (5)) that the algorithm should satisfy in addition to the first objective, which is minimizing the distance in Eq. (1).

$$f_2(x, y) = \sum_{j=1}^{n} \Delta\theta_j \tag{4}$$

$$\Delta\theta_j = tan^{-1}\left(\frac{\Delta y}{\Delta x}\right) - tan^{-1}\left(\frac{y(goal) - y(j)}{x(goal) - x(j)}\right) \tag{5}$$

Where, $j = 1,2, \dots n-1$

2.3 Multi-objective approach

Multiple objective optimization (MOO) is used when an optimization issue has more than one objective function and the goal is to identify one or more optimal solutions. The weighted sum method is a common way to deal with multiple objective optimization. It uses the weighted sum to combine all of the many objective functions into a single scalar or composite objective function, which is easier to understand [25].

$$f(x, y) = \sum_{m=1}^{M} W_m\, f_m\,(x, y)$$
$$= W_1\, f_1\,(x, y) + W_2\, f_2(x, y) \tag{6}$$

The importance of defining the weighting coefficient, W = (W1, W2, ..., Wm), is that the strong solution is determined by W. These weights have obviously been beneficial and satisfying [26]. $\sum_{m=1}^{M} W_m = 1$, $W_m \in [0,1]$.

2.4 Obstacles movement

In this paper, in the case of the dynamic obstacle that moves from one location to another at each time step, the velocity is (v_{obs}) and direction (θ_{obs}) of the dynamic obstacles are assumed to be random according to the following equations:

$$\chi_{obs} = \chi_{obs} + v_{obs} \times cos\,\theta_{obs} \tag{7}$$

$$y_{obs} = y_{obs} + v_{obs} \times sin\,\theta_{obs} \tag{8}$$

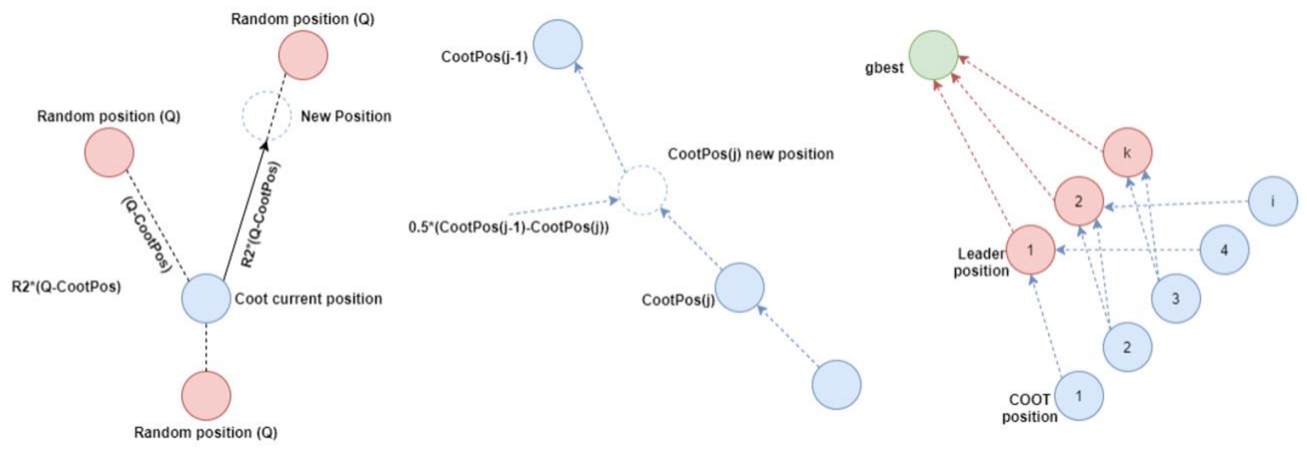
Where,

$$\theta_{obs} = 360 \times rand(0,1) \tag{9}$$

$$v_{obs} = rand(0,1) \tag{10}$$

3. COOT optimization technique

New swarm-based algorithm, COOT [27] , is discussed in this paper. The COOT algorithm was invented by Naruei and Keynia in 2021 as a swarm-based algorithm inspired by collective movements (irregular and regular movements on the water's surface). A few coots in front of the group, which is regarded as a group leader, steer the entire group



(a) (b) (c)

Figure. 4: (a) Random movement, (b) Chain movement, and (c) Choose a leader by coot

toward the destination (food). Coots make four distinct movements on the water's surface, which are as follows [27]:

1. Random movement.
2. Chain movement.
3. Adjusting the position based on the leaders of the group.
4. Led by their leaders, they lead the group towards the optimal location.

The population is calculated randomly using Eq. (11).

$$CP(j) = rand(1, d) \times (u - l) + l \qquad (11)$$

Where $CP(j)$ denotes the COOT position at instant j, d denotes the search space dimension (number of optimization variables), and l and u denote the lower and upper search space limits. Then the four movement phases are defined as:

(1) Phase of random movement:

In this phase, the random position (Q) is determined using Eq. (12), to look through the many regions of the search space

$$Q = rand(1, d) \times (u - l) + l \qquad (12)$$

The COOT's new positions must escape local optimal solutions. Thus, Eq. (13) can calculate the new COOT positions as follows:

$$CP(j) = CP(j) + A \times R2 \times (Q - CP(j)) \qquad (13)$$

Where R2 is a random variable in ∈ [0, 1], A is defined by Eq. (14) as follows:

$$A = 1 - L \times \left(\frac{1}{Iter}\right) \qquad (14)$$

L shows the current iteration, and the maximum number of iterations is shown in $Iter$.

(2) Chain movement phase:

This phase is represented mathematically by using Eq. (15).

$$CP(j) = 0.5 \times (CP\,(j-1) + CP(j) \qquad (15)$$

(3) Phase of Adjusting the position based on the group leader's:

In this movement, Eq. (16) is used to adjust the position.

$$K = 1 + (j\ MOD\ NL) \qquad (16)$$

Where j is a number that is represented by an index of all the coots, NL shows the leader's number, and K is a number that indicates the number of the leader. The position of the next coot is calculated based on the leader number k as follows:

$$CP(j) = LP(K) + 2\ \times R1\ \times cos\ (2R\pi) \times \big(LP(K) - CP(j)\big) \qquad (17)$$

Where LP represents the leader position.

(4) phase of leading the group towards the optimal area: Eq. (18) is used to change the leader's position.

$$LP(j) = \begin{cases} B \times R3 \times cos(2R\pi) \times (gBest - LP\,(j)) + gBest & R4 \prec 0.5 \\ B \times R3 \times cos(2R\pi) \times (gBest - LP\,(j)) - gBest & R4 \succ 0.5 \end{cases} \qquad (18)$$

Where $gBest$ represent, the best place is found, R3, R4, and R are all random numbers. R3 and R4 ∈[0, 1], R ∈ [- 1, 1], and B is defined in Eq. (19):

$$B = 2 - L \times \left(\frac{1}{Iter}\right) \quad (19)$$

Pseudocode that shows how the COOT algorithm works is shown in [27].

4. Proposed improved COOT algorithm

A The details of the improved COOT (ICOOT) algorithm are presented in the following:

1. To keep the local optimum from becoming stagnant and further wasting time, individuals in the swarm must be led by the best in the entire swarm. It will be used to make the random movement equation in the proposed ICOOT:

$$CP(j) = CP(j) + A \times R2 \times (Q + gBest - CP(j)) \quad (20)$$

2. The main drawback in the conventional COOT algorithm is the absence of the parameter of transfer from exploitation and exploration. This ends in unstable searches (stagnation in the local optimum) and the squandering of more time. To overcome this drawback, a new parameter called "acceleration" (acc) is utilized. According to Hugh Trenchard [28], the transition from disordered to synchronized states appears to be induced by two main factors: the first one is acceleration between individual coots within the flock, which brings accelerating coots closer to slower moving coots, causing slower moving coots to adjust their orientation to align with accelerating coots. The second thing that happens is that coots in leadership positions or on the periphery of the group speed up. This causes a widening gap between coots, which forces others to change their orientations and speeds to follow. In this paper, the acceleration depends on density, meaning that as a low-density group they can swim faster by alternating leading positions than when swimming in higher densities. According to Fig. 5, density can be defined as

$$D(j) = \frac{N}{\pi r^2} \quad (21)$$

Where N is the number of coots and πr^2 the total area occupied by all birds.

To find the radius (r), first find the center coot position (CCP) of the swarm:

$$CCP(x,y) = \frac{FCP(x,y)}{LCP\ (x,y)} = \left(\frac{|max(x)|-|min(x)|}{2}, \frac{|max(y)|-|min(y)|}{2}\right) \quad (22)$$

Where FCP is the first coot in the swarm and LCP is the last coot in the swarm. So, the radius (r) can be found as:

$$r = max\ \left(CCP(x), CCP(y)\right) + 1 \quad (23)$$

Now, "acceleration" (acc) can be found according to the previous rule:

$$acc(j) = \left|(1-D) + \left(1 - \frac{1}{|a|}\right) - rand\ (0,2)\right| \quad (24)$$

Where α is the number of free points with integer coordinates that lie inside the circle.

$$\alpha = \beta 1 * \beta 2 - \eta \quad (25)$$

Where β1 reprsent number of integer coordinates inside circle, it can be found by using the following equation:

$$\beta 1 = \sqrt{\left|x_p - x_c\right|^2 + \left|y_p - y_c\right|^2} < r \quad (26)$$

Where, (x_c, y_c) represents the center of the circle and $\left(x_p, y_p\right)$ whether the point is inside the circle or not, $x_p = \min(x){:}\,1{:}\max(x)$ and $y_p = \min(y){:}\,1{:}\max(y)$

and β2 is the minimum difference between two coots in the swarm,

$$\beta 3 = Sort(COOT_{POS(:,1)})) \quad (27)$$

$$\beta 4 = Sort\left(COOT_{POS(:,2)}\right) \quad (28)$$

$$\beta 2 = \sqrt{\left(\beta 3(2) - \beta 3(1)\right)^2 + \left(\beta 4(2) - \beta 4(1)\right)^2} \quad (29)$$

and η = number of coordinates accoupied by the coot

So, according to the above, the following are new movement phases:

(1) Phase of random movement:

$$\mathrm{CP}(j) = \mathrm{CP}(j) + A \times acc(j) \times R2 \times (gbest + Q - \mathrm{CP}(j)) \quad (30)$$

Figure. 5 14 coots, density = $\frac{14}{\pi r^2}$

(2) Phase of adjusting the position based on the group leader's:

$$CP(j) = LP(k) + 2 \times acc(j) \times R1 \times cos\ (2R\pi) \times \left(LP(k) - CP(j)\right) \quad (31)$$

(3) Phase of leading the group towards the optimal area:

$$LP(j) = \left\{ \begin{matrix} 2\ \times acc(j)\ \times B \times R3 \times cos(2R\pi) \\ \times (gBest - LP\ (j)) + gBest \quad R4 < 0.5 \\ 2\ \times acc(i)\ \times B \times R3 \times cos(2R\pi) \\ \times (gBest - LP\ (j)) - gBest \quad R4 > 0.5 \end{matrix} \right\} \quad (32)$$

4.1 Proposed a local search strategy

The proposed local search (LS) is a local search strategy that transforms infeasible solutions into viable ones. The solution is considered infeasible in two cases: first, if the next selected point by the ICOOT algorithm is within the obstacle region (Fig. 6, case (1)). The second, if it is outside the obstacle but a line connects this point with the previous point, or the consequent point is passing through the obstacle (Fig. 6 case (2)). According to the following suggested criteria, these two cases are solved by trying to evict the possible solutions from the obstacle's occupied area.

In case 1, the suggested next position lies inside the obstacle: There is a way to get out of this situation by checking if the next suggested position in the path is occupied or not. If a place is already occupied, the algorithm looks for the next best unoccupied position.

Case 2, the line between two points passes through the obstacle: This issue can be fixed by finding the equation of a straight n-point line between the current point and the next suggested point. Then, if any n-points are occupied and the line is vertical (difference between y axis n-points > difference x n-point), the algorithm goes horizontally left and right, then finds the nearest unoccupied point (x (i+1), y (i+1)) to (x (i), y (i)) and vice versa, if the line passes through the obstacle is horizontal.

4.2 Obstacle detection and avoidance (ODA)

A description of the sensor is needed since the robot is working in an unknown environment. Obstacle detection sensing (ODS) is the name of a proposed method for finding obstacles and avoiding them, and it's what we're going to talk about now.

(1) The procedure for detecting obstacles

The robot map is imported into the MTALAB workspace and a binary map of occupancy is constructed. The occupancy map is simply a 2D matrix. Every pixel on the map is labeled with either binary 0 (non-occupied) or binary 1 (occupied by static and/or dynamic obstacles). ODS is performed by encircling the mobile robot with eighty virtual sensors (VS). The VS is placed in four layers. The first layer has eight VS, and each sensor has a specific angle range of 30. The second one is sixteen VS, and each sensor has a specific angle range of 22.5◦. The third layer, including twenty-four VS, and each sensor, covers an angle range of 14.4. The last layer, thirty-two VS, and each sensor, covers an angle range of 11.25◦. These VS positions are pointed at each iteration according to the robot's current position (RCP), as shown in Fig. 7.

(2) Obstacle avoidance algorithm

At each iteration, the occupancy map provides information about the existence of obstacles (occupied (binary 1) or unoccupied (binary 0)). The algorithm finds the best unoccupied position according to objectives among eighty possible positions that are randomly arranged around RCP (gBest). Each of these eighty positions with RCP is updated at each iteration by the ICOOT algorithm.

So, the robot has more options to increase speed or slow down to avoid obstacles. So, the proposed robot path planning algorithm is illustrated in this simple pseudocode.

4.3 Kinematic model

Kinematic modeling of the robot is needed to figure out where the robot will go next and how to get around obstacles. The robot is depicted as a horizontally traveling, rigid body on wheels. The robot chassis moves in a 2D plane defined by the coordinates [x, y]. To make the model easier to understand, things like wheel axels and steering

Received: May 28, 2022. Revised: July 17, 2022.

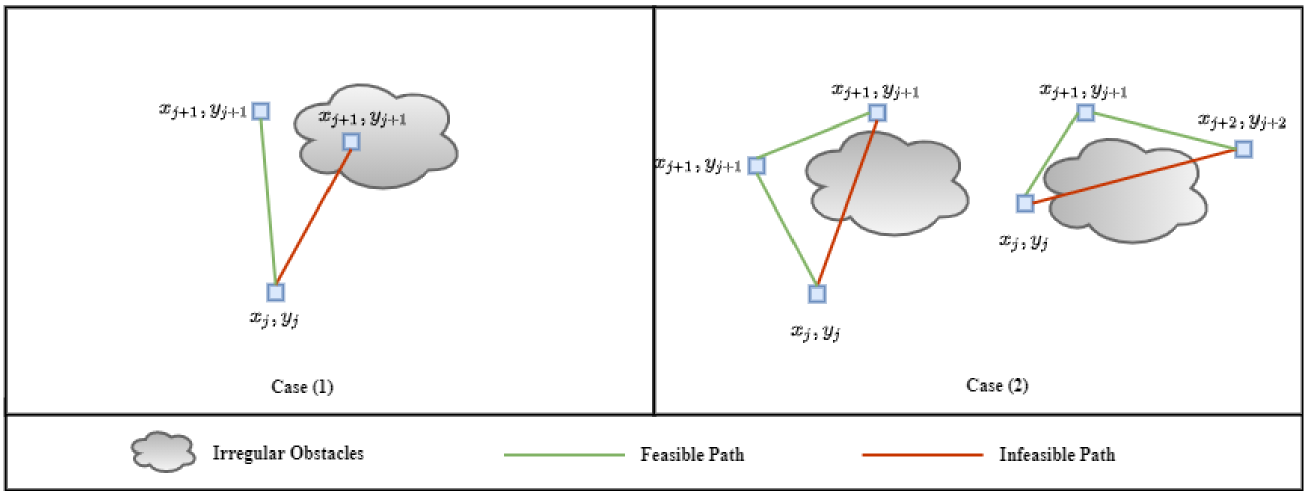


Figure. 6 Infeasible path type

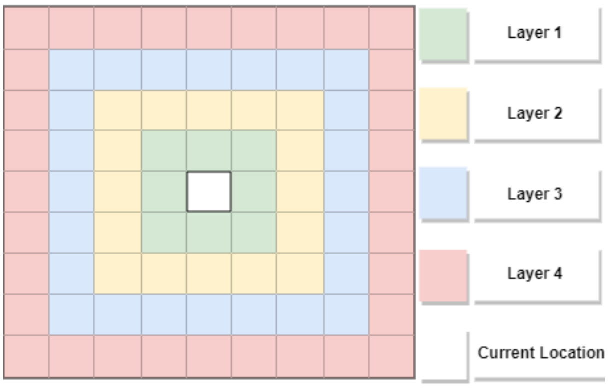


Figure. 7 Obstacle detection

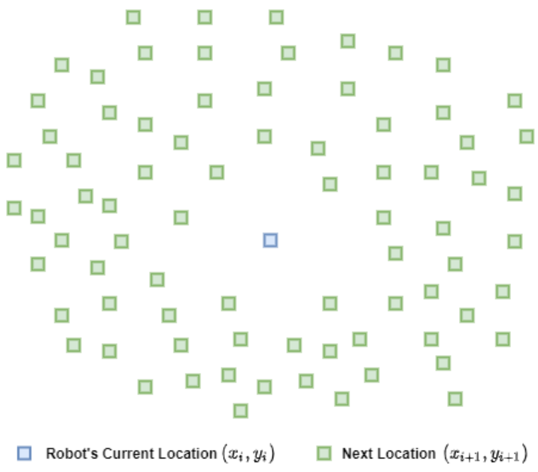

Robot's Current Location (x_i, y_i)
Next Location (x_{i+1}, y_{i+1})

Figure. 8 Robot possible positions

```
Initiliaze: first population of coots
Initiliaze: start position, target position, error Threshold, number of leaders
Set: number of coots and random selection of leaders
Calculate: the fitness of coots and leaders
find the best coot or leaders as the global optimum (gbest)
While gbest = target postion || error>error Threshold
Start: Obstacle Detection and Avoidance (ODA) algorithm
Update the best postion (gbest)
Start: a local search strategy
Mobile robot moves to new position (gbest)
iter=iter+1
end while
```

wheel joints are left out [29].

The relationship between the plane global reference frame GRF and the robot local refence frame LFR is shown in Fig. 9. The axes Xl-Yl define the GRF at the origin O: {Xl,Yl The robot's reference point is chosen as point P on the robot chassis. x, y, and the angle between the GFR, the LFR and θ, establish the position of reference point P in the global frame. The position of the robot can be described as a vector:

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{33}$$

The attitude associated with the GFR is denoted by the subscript "I." The mapping is determined by the robot's present position, and the orthogonal rotation matrix is used to explain it:

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{34}$$

The motion of the robot is mapped from the reference frame [XI, YI] to a motion term in the LRF [XR, YI] using the matrix of Equation [4]. The velocity vector $\dot{\xi}_I$ is used to express this mapping

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \tag{35}$$

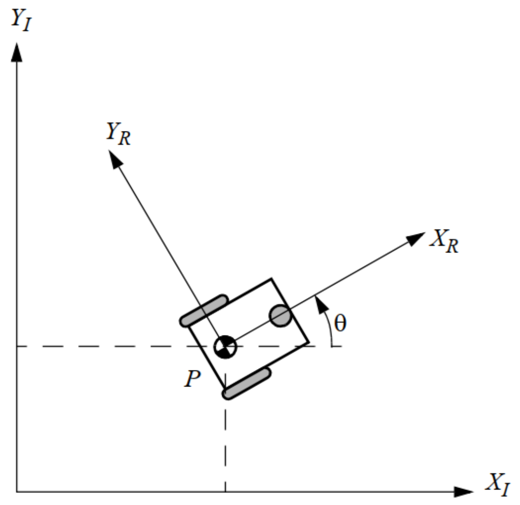


Figure. 9 GRF [XI, YI] and the LRF [XR, YR]

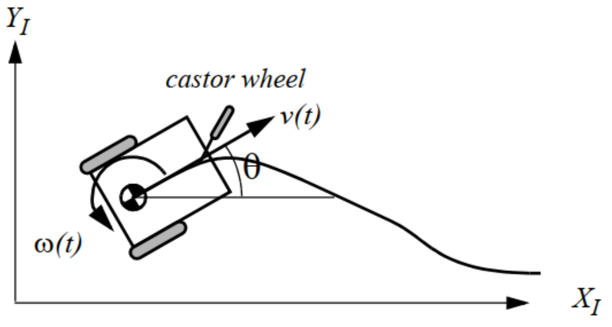


Figure. 10 Differential mobile robot moving in its GRF

The relationship between the motions in the two frames is thus expressed as:

$$\dot{\xi}_R = R(\theta)\dot{\xi}_I \tag{36}$$

Taking turtlebot3 burger as an assumption, turtlebot3 burger has two wheels, each of which is r in diameter. At a distance of l from each wheel, the reference point P is centered. Therefore, given $r, 1, \theta$ and each wheel's rotational speed $\dot{\phi}_1$ and $\dot{\phi}_2$. the robot's overall speed can be estimated using the forward kinematic model in the GFR as:

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f\left(l, r, \theta, \dot{\phi}_1, \dot{\phi}_2\right) \tag{37}$$

The final differential-drive robot's kinematic model is given by:

$$\dot{\xi}_I = R(\theta)^{-1} \begin{bmatrix} \frac{r\dot{\phi}_1}{2} \frac{r\dot{\phi}_2}{2} \\ 0 \\ \frac{r\dot{\phi}_1}{2l} \frac{-r\dot{\phi}_2}{2l} \end{bmatrix} \tag{38}$$

Where $R(\theta)^{-1}$ is calculated as:

$$R(\theta)^{-1} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{39}$$

5. Results and discussion

5.1 Proposed algorithm performance on benchmark test functions

The ICOOT algorithm is assessed in 13 criterion functions (30, 100, and 500 dimensions) in this section. These are standard functions that a lot of researchers have utilized [30]. These tests are used by ICOOT in comparison to the outcomes of the swarm algorithms. Table 1 show these standard functions, where Range denotes the boundary of the function's search space, and f-min denotes the optimal value. The first seven functions are unimodal and the last six are multimodal.

The ICOOT algorithm is compared with five swarm optimization algorithms: the first is particle swarm optimization (PSO); the second is another well-known algorithm called salp swarm algorithm (SSA); and the fitness dependent optimizer (FDO). They are also compared with the conventional COOT optimization algorithm and the enhanced version of COOT to validate its results. The search agents are 30 in number, the maximum number of iterations is 500.

Firstly implemented the algorithm on 13 test functions with 30 dimensions. The proposed ICOOT algorithm achieves the best results in all test functions, as shown in Table 2 (F1-F7). As evidenced by this, the ICOOT algorithm has successfully exploited the search space. Algorithm exploration is measured using multimodal functions. There are several local optimums in these functions, and the algorithm should avoid them. Table 2 shows the statistical results of the algorithms on these functions (F8-F13). The new ICOOT algorithm also performs better results in 4c out of 6 test functions.

The proposed algorithm was then applied to the same 13 test functions from the first round of testing, each with 100 dimensions. Table 3 displays the outcomes of this application. The outcomes demonstrate that the suggested algorithm is stable in most test functions and does not suffer from the growth of the problem size. At this point, the proposed algorithm outperformed all the compared algorithms in all 5 multimodal test functions and 5 unimodal test functions (F9, F10, F11, F12, and F13).

In the final step using 13 test functions with 500 dimensions. Table 6 displays the experiment's

Table 1. Benchmark test functions

Function	Range	fMIN
$f_1(x) = \sum_{i=1}^{n} x_i^2$	[-100, 100]	0
$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	[-10, 10]	0
$f_3(x) = \sum_{i=1}^{n} \left(\sum_{j-1}^{i} x_j \right)^2$	[-100, 100]	0
$f_4(x) = max\{\lvert x_i \rvert, 1 \leqslant i \leqslant n\}$	[-100, 100]	0
$f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_l - 1)^2]$	[-30, 30]	0
$f_6(x) = \sum_{i=1}^{n} ([x_i + 0.5])^2$	[-100, 100]	0
$f_7(x) = max\{\lvert x_i \rvert, 1 \leqslant i \leqslant n\}$	[-1.28, 1.128]	0
$F_8(x) = \sum_{i=1}^{n} - x_i \, sin\left(\sqrt{\lvert x_i \rvert}\right)$	[-500, 500]	0
$F_9(x) = \sum_{i=1}^{n} [x_i^2 - 10 \, cos(2\pi x_i) + 10]$	[-5.12, 5.12]	0
$F_{10}(x) = -20 \, cxp\left(-0.2 \sqrt{\frac{1}{n} \sum_{i-1}^{n} x_i^2} \right) - cxp\left(\frac{1}{n} \sum_{i-1}^{n} cos(2\pi x_i) \right) + 20 + c$	[-32, 32]	0
$F_{11}(x) = \frac{1}{4000} \sum_{i-1}^{n} x_i^2 - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[-600, 600]	0
$F_{12}(x) = \frac{\pi}{n}\left\{10 \, sin(\pi y_1) + \sum_{i-1}^{n-1} (y_i - 1)^2 [1 + 10 \, sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\}$ $+ \sum_{i-1}^{n} u(x_i, 10,100,4) + \sum_{i=1}^{n} u(x_i, 10,100,4) y_i = 1 + \frac{x_i + 1}{4}$ $u(x_i, a, k, m) = \left\{ \begin{array}{cc} k(x_i - a)^m & x_i > a \\ 0 & -a < x_i < a \\ k(-x_i - a)^m & x_i < -a \end{array} \right\}$	[-50, 50]	0
$F_{13}(x) = 0.1\left\{sin^2(3\pi x_1) + \sum_{i=1}^{n} (x_i - 1)^2 [1 + sin^2(3\pi x_i + 1)]\right.$ $\left. + (x_{11} - 1)^2 [1 + sin^2(2\pi x_n)]\right\} + \sum_{i=1}^{n} u(x_i, 5,100,4)$	[-50, 50]	0

outcomes. With the exception of function 8, the COOT algorithm has outperformed all other algorithms. The outcomes demonstrate that the suggested algorithm is stable with increasing dimensions, while other algorithms have performed terribly. The outcomes in Tables 2 to 4 demonstrate how scalable the suggested algorithm is.

5.2 RPP in a complex static environment

The efficiency of the suggested for the mobile robot, a path planning algorithm has been developed and is shown in this case study when the robot is in a static obstacle environment. The static environment

Table 2. The Benchmark Test Functions were run over 1000 times with 30 dimensions.

Fun.	Fit.	**PSO** [31] **30 Dim**	**SSA**[32] **30 Dim**	**FDO** [33] **30 Dim**	**mCOOT** [34] **30 Dim**	**COOT**[27] **30 Dim**	**ICOOT 30 Dim**
F1	min	7.5124×10^{-08}	1.6368×10^{-08}	$3.2270 \times 10^{+03}$	6.1833×10^{-56}	9.8206×10^{-58}	3.4334×10^{-52}
	max	4.0190×10^{-05}	6.6116×10^{-07}	$1.0719 \times 10^{+04}$	5.4187×10^{-11}	8.6061×10^{-08}	1.7486×10^{-15}
	avg	2.2965×10^{-06}	1.3651×10^{-07}	$6.3139 \times 10^{+03}$	5.4838×10^{-12}	8.7095×10^{-11}	1.7507×10^{-18}
	std	7.2554×10^{-06}	1.5491×10^{-07}	$1.8230 \times 10^{+03}$	1.7135×10^{-13}	2.7215×10^{-09}	**5.5295×10^{-17}**
F2	min	9.0072×10^{-05}	1.0536×10^{-01}	$2.6008 \times 10^{+01}$	1.8702×10^{-28}	2.9703×10^{-30}	3.1098×10^{-28}
	max	8.6400×10^{-03}	$3.1663 \times 10^{+00}$	$4.9082 \times 10^{+01}$	5.3945×10^{-07}	8.5677×10^{-05}	3.3093×10^{-10}
	avg	1.4400×10^{-03}	$1.3328 \times 10^{+00}$	$3.3957 \times 10^{+01}$	2.4047×10^{-09}	3.8193×10^{-07}	6.9326×10^{-13}
	std	1.9200×10^{-03}	9.3288×10^{-01}	$5.7193 \times 10^{+00}$	2.9066×10^{-08}	4.6164×10^{-06}	**1.4448×10^{-11}**
F3	min	$2.4899 \times 10^{+01}$	$1.9521 \times 10^{+02}$	$7.1684 \times 10^{+03}$	3.1046×10^{-55}	4.9308×10^{-57}	4.4776×10^{-53}
	max	$3.6510 \times 10^{+02}$	$2.8737 \times 10^{+03}$	$2.9706 \times 10^{+04}$	9.1290×10^{-08}	1.4499×10^{-07}	1.0897×10^{-18}
	avg	$1.2401 \times 10^{+02}$	$1.2921 \times 10^{+03}$	$1.9184 \times 10^{+04}$	1.0900×10^{-12}	1.7311×10^{-10}	1.7000×10^{-21}
	std	$8.1552 \times 10^{+01}$	$7.2173 \times 10^{+02}$	$6.0544 \times 10^{+03}$	2.9175×10^{-13}	4.6336×10^{-09}	**3.7661×10^{-20}**
F4	min	8.7640×10^{-01}	$3.5141 \times 10^{+00}$	$2.4018 \times 10^{+01}$	1.0303×10^{-29}	1.6363×10^{-34}	1.3285×10^{-27}
	max	$4.9311 \times 10^{+00}$	$1.7169 \times 10^{+01}$	$3.9262 \times 10^{+01}$	3.7716×10^{-07}	5.9902×10^{-04}	9.9811×10^{-09}
	avg	$2.0211 \times 10^{+00}$	$9.5648 \times 10^{+00}$	$3.2439 \times 10^{+01}$	3.9409×10^{-08}	6.2591×10^{-07}	2.3503×10^{-11}
	std	8.7088×10^{-01}	$3.1342 \times 10^{+00}$	$3.7864 \times 10^{+00}$	1.1930×10^{-06}	1.8948×10^{-05}	**4.0821×10^{-10}**
F5	min	$1.2929 \times 10^{+01}$	$2.0042 \times 10^{+01}$	$2.0875 \times 10^{+02}$	$1.6882 \times 10^{+01}$	$2.6813 \times 10^{+01}$	$2.6941 \times 10^{+01}$
	max	$8.4144 \times 10^{+01}$	$1.2065 \times 10^{+03}$	$9.4832 \times 10^{+06}$	$1.0561 \times 10^{+02}$	$1.6774 \times 10^{+03}$	$2.8691 \times 10^{+01}$
	avg	$3.0213 \times 10^{+01}$	$2.0168 \times 10^{+02}$	$4.9340 \times 10^{+06}$	$2.8439 \times 10^{+01}$	$4.5168 \times 10^{+01}$	$2.7950 \times 10^{+01}$
	std	$1.9874 \times 10^{+01}$	$2.7394 \times 10^{+02}$	$1.8511 \times 10^{+06}$	$4.5214 \times 10^{+01}$	$7.1811 \times 10^{+01}$	**3.0230×10^{-01}**
F6	min	3.1085×10^{-08}	2.2926×10^{-08}	$2.9047 \times 10^{+03}$	7.6815×10^{-03}	1.2200×10^{-02}	1.0500×10^{-02}
	max	7.8797×10^{-06}	6.5254×10^{-07}	$1.1198 \times 10^{+04}$	7.9100×10^{-01}	$1.2563 \times 10^{+00}$	2.0190×10^{-01}
	avg	1.2411×10^{-06}	1.6204×10^{-07}	$6.0501 \times 10^{+03}$	9.1800×10^{-02}	1.4580×10^{-01}	5.0200×10^{-02}
	std	1.9515×10^{-06}	1.5808×10^{-07}	$1.7282 \times 10^{+03}$	7.5430×10^{-02}	1.1980×10^{-01}	**2.3500×10^{-02}**
F7	min	8.1600×10^{-03}	4.9200×10^{-02}	$1.0362 \times 10^{+00}$	3.3827×10^{-05}	5.3725×10^{-05}	8.4180×10^{-06}
	max	4.4880×10^{-02}	3.9160×10^{-01}	$6.5583 \times 10^{+00}$	2.6759×10^{-02}	4.2500×10^{-02}	1.6300×10^{-02}
	avg	2.0240×10^{-02}	1.3672×10^{-01}	$2.7259 \times 10^{+00}$	3.2111×10^{-03}	5.1000×10^{-03}	3.0000×10^{-03}
	std	8.2400×10^{-03}	7.2800×10^{-02}	$1.2317 \times 10^{+00}$	2.9074×10^{-03}	4.3000×10^{-03}	**2.8000×10^{-03}**
F8	min	$-6.3116\times 10^{+03}$	$-7.1509 \times 10^{+03}$	$-3.2318 \times 10^{+03}$	$-7.6720 \times 10^{+03}$	$-1.2185 \times 10^{+04}$	$-1.0322 \times 10^{+04}$
	max	$-3.8802 \times 10^{+03}$	$-4.8477 \times 10^{+03}$	$-1.7393 \times 10^{+03}$	$-3.1031 \times 10^{+03}$	$-4.9284 \times 10^{+03}$	$-3.1085 \times 10^{+03}$
	avg	$-5.2250 \times 10^{+03}$	$-5.9466 \times 10^{+03}$	$-2.2258 \times 10^{+03}$	$-4.6161 \times 10^{+03}$	$-7.3315 \times 10^{+03}$	$-6.5649 \times 10^{+03}$
	std	$6.4182 \times 10^{+02}$	$6.1953 \times 10^{+02}$	**$3.2376 \times 10^{+02}$**	$5.7506 \times 10^{+02}$	$9.1334 \times 10^{+02}$	$1.1677 \times 10^{+03}$
F9	min	$7.6413 \times 10^{+01}$	$2.3083 \times 10^{+01}$	$1.2846 \times 10^{+02}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$
	max	$7.6413 \times 10^{+01}$	$7.9597 \times 10^{+01}$	$1.8942 \times 10^{+02}$	2.1669×10^{-07}	3.4416×10^{-06}	6.2528×10^{-13}
	avg	$3.8604 \times 10^{+01}$	$4.7476 \times 10^{+01}$	$1.6083 \times 10^{+02}$	2.1742×10^{-10}	3.4531×10^{-09}	1.0573×10^{-14}
	std	$1.2983 \times 10^{+01}$	$1.4882 \times 10^{+01}$	$1.5238 \times 10^{+01}$	6.8523×10^{-11}	1.0883×10^{-07}	**5.3272×10^{-14}**
F10	min	3.6847×10^{-05}	$1.3170 \times 10^{+00}$	$8.1832 \times 10^{+00}$	5.5922×10^{-16}	8.8818×10^{-16}	8.8818×10^{-16}
	max	$2.2509 \times 10^{+00}$	$3.5064 \times 10^{+00}$	$1.1314 \times 10^{+01}$	1.5663×10^{-06}	2.4877×10^{-05}	1.5657×10^{-09}
	avg	$1.0174 \times 10^{+00}$	$2.0904 \times 10^{+00}$	$1.0321 \times 10^{+01}$	4.6520×10^{-09}	7.3884×10^{-08}	1.7670×10^{-12}
	std	6.5720×10^{-01}	5.0912×10^{-01}	7.7416×10^{-01}	6.8711×10^{-07}	1.0913×10^{-06}	**4.9677×10^{-11}**
F11	min	3.0870×10^{-08}	7.1483×10^{-04}	$2.7147 \times 10^{+01}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$
	max	5.6720×10^{-02}	3.6400×10^{-02}	$8.6144 \times 10^{+01}$	2.7690×10^{-08}	4.3979×10^{-07}	7.9714×10^{-14}
	avg	1.1840×10^{-02}	1.3520×10^{-02}	$5.5598 \times 10^{+01}$	2.7697×10^{-11}	4.3990×10^{-10}	1.5266×10^{-16}
	std	1.3360×10^{-02}	9.3600×10^{-03}	$1.5810 \times 10^{+01}$	8.7563×10^{-09}	1.3907×10^{-08}	**2.5573×10^{-15}**
F12	min	2.4686×10^{-09}	$2.2231 \times 10^{+00}$	$5.4161 \times 10^{+04}$	3.3693×10^{-04}	5.3513×10^{-04}	4.6113×10^{-04}
	max	7.4704×10^{-01}	$8.9368 \times 10^{+00}$	$8.2632 \times 10^{+06}$	$2.8676 \times 10^{+00}$	$4.5544 \times 10^{+00}$	1.6140×10^{-01}
	avg	1.4120×10^{-01}	$4.8889 \times 10^{+00}$	$2.8270 \times 10^{+06}$	1.2630×10^{-02}	2.0060×10^{-01}	6.7000×10^{-03}
	std	1.9880×10^{-01}	$1.9097 \times 10^{+00}$	$1.9333 \times 10^{+06}$	3.0222×10^{-01}	4.8000×10^{-01}	**8.0000×10^{-03}**
F13	min	2.7983×10^{-07}	7.2400×10^{-02}	$3.2630 \times 10^{+06}$	1.3411×10^{-02}	2.1300×10^{-02}	5.4400×10^{-02}
	max	4.9800×10^{-01}	$3.4803 \times 10^{+01}$	$2.2232 \times 10^{+07}$	$2.6296 \times 10^{+00}$	$4.1765 \times 10^{+00}$	$2.9690 \times 10^{+00}$
	avg	2.7680×10^{-02}	$1.3999 \times 10^{+01}$	$1.0170 \times 10^{+07}$	2.8705×10^{-01}	4.5590×10^{-01}	$1.7224 \times 10^{+00}$
	std	9.1360×10^{-02}	$1.1228 \times 10^{+01}$	$5.3626 \times 10^{+06}$	**2.9593×10^{-02}**	4.7000×10^{-01}	$1.1820 \times 10^{+00}$

Received: May 28, 2022. Revised: July 17, 2022.

Table 3. The Benchmark Test Functions were run over 1000 times with 100 dimensions

Fun.	Fit.	**PSO** [31] **100 Dim**	**SSA**[32] **100 Dim**	**FDO** [33] **100 Dim**	**mCOOT** [34] **100 Dim**	**COOT**[27] **100 Dim**	**ICOOT 100 Dim**
F1	min	$5.5722 \times 10^{+01}$	$7.5314 \times 10^{+02}$	$3.0089 \times 10^{+04}$	5.5845×10^{-61}	8.8695×10^{-65}	1.4110×10^{-55}
	max	$3.3461 \times 10^{+02}$	$1.7282 \times 10^{+03}$	$5.3646 \times 10^{+04}$	9.8587×10^{-07}	1.5658×10^{-06}	2.0706×10^{-16}
	avg	$1.3911 \times 10^{+02}$	$1.1375 \times 10^{+03}$	$3.8606 \times 10^{+04}$	1.0579×10^{-12}	1.6802×10^{-09}	2.2919×10^{-19}
	std	$6.1808 \times 10^{+01}$	$2.6206 \times 10^{+02}$	$5.5506 \times 10^{+03}$	3.1250×10^{-13}	4.9632×10^{-08}	**6.5694×10^{-18}**
F2	min	$5.1531 \times 10^{+00}$	$2.8970 \times 10^{+01}$	$1.6734 \times 10^{+02}$	1.4755×10^{-29}	2.3435×10^{-27}	1.4304×10^{-26}
	max	$1.6889 \times 10^{+01}$	$5.1411 \times 10^{+01}$	$8.0480 \times 10^{+10}$	1.5741×10^{-03}	2.5000×10^{-02}	1.5027×10^{-09}
	avg	$8.2088 \times 10^{+00}$	$3.7718 \times 10^{+01}$	$4.5337 \times 10^{+09}$	1.7271×10^{-06}	2.7431×10^{-05}	3.2607×10^{-12}
	std	$2.6060 \times 10^{+00}$	$5.3453 \times 10^{+00}$	$1.6450 \times 10^{+10}$	4.9920×10^{-06}	7.9285×10^{-04}	**6.3071×10^{-11}**
F3	min	$1.2368 \times 10^{+04}$	$1.2150 \times 10^{+04}$	$9.3800 \times 10^{+04}$	1.8742×10^{-58}	2.9766×10^{-60}	4.3552×10^{-52}
	max	$3.6283 \times 10^{+04}$	$7.5970 \times 10^{+04}$	$3.4903 \times 10^{+05}$	6.0307×10^{-11}	9.5782×10^{-08}	2.4988×10^{-15}
	avg	$2.3336 \times 10^{+04}$	$3.8695 \times 10^{+04}$	$2.4800 \times 10^{+05}$	6.2011×10^{-12}	9.8488×10^{-11}	2.5536×10^{-18}
	std	$6.2766 \times 10^{+03}$	$1.6246 \times 10^{+04}$	$6.3892 \times 10^{+04}$	1.9073×10^{-12}	3.0293×10^{-09}	**7.9026×10^{-17}**
F4	min	$1.5223 \times 10^{+01}$	$1.5098 \times 10^{+01}$	$4.7058 \times 10^{+01}$	1.0224×10^{-27}	1.6238×10^{-31}	1.0332×10^{-26}
	max	$2.6774 \times 10^{+01}$	$2.7810 \times 10^{+01}$	$5.7889 \times 10^{+01}$	2.9039×10^{-02}	4.6120×10^{-01}	1.4662×10^{-06}
	avg	$1.9118 \times 10^{+01}$	$2.1805 \times 10^{+01}$	$5.3201 \times 10^{+01}$	2.9129×10^{-05}	4.6264×10^{-04}	1.5419×10^{-09}
	std	$2.4487 \times 10^{+00}$	$2.9840 \times 10^{+00}$	$2.7074 \times 10^{+00}$	9.1926×10^{-03}	1.4600×10^{-02}	**4.6417×10^{-08}**
F5	min	$3.4309 \times 10^{+03}$	$4.6377 \times 10^{+04}$	$1.6928 \times 10^{+07}$	$6.1635 \times 10^{+01}$	$9.7891 \times 10^{+01}$	$9.7556 \times 10^{+01}$
	max	$3.6166 \times 10^{+04}$	$5.9739 \times 10^{+05}$	$9.7808 \times 10^{+07}$	$5.0185 \times 10^{+04}$	$7.9705 \times 10^{+03}$	$9.8444 \times 10^{+01}$
	avg	$1.0002 \times 10^{+04}$	$1.5718 \times 10^{+05}$	$4.3122 \times 10^{+07}$	$2.0153 \times 10^{+01}$	$3.2007 \times 10^{+02}$	$9.8256 \times 10^{+01}$
	std	$8.1664 \times 10^{+03}$	$1.1650 \times 10^{+05}$	$1.9834 \times 10^{+07}$	$4.4796 \times 10^{+01}$	$7.1146 \times 10^{+02}$	**1.1180×10^{-01}**
F6	min	$4.3837 \times 10^{+01}$	$7.4813 \times 10^{+02}$	$2.5374 \times 10^{+04}$	$2.5079 \times 10^{+00}$	$3.9831 \times 10^{+00}$	$3.3907 \times 10^{+00}$
	max	$5.4170 \times 10^{+02}$	$1.9870 \times 10^{+03}$	$5.3297 \times 10^{+04}$	$1.1917 \times 10^{+02}$	$1.8927 \times 10^{+02}$	$7.9542 \times 10^{+00}$
	avg	$1.6494 \times 10^{+02}$	$1.1957 \times 10^{+03}$	$3.8327 \times 10^{+04}$	$8.9365 \times 10^{+00}$	$1.4193 \times 10^{+01}$	$5.2465 \times 10^{+00}$
	std	$1.1005 \times 10^{+02}$	$3.2285 \times 10^{+02}$	$7.4018 \times 10^{+03}$	$1.2831 \times 10^{+01}$	$2.0378 \times 10^{+01}$	**7.7500×10^{-01}**
F7	min	2.5064×10^{-01}	$1.0387 \times 10^{+00}$	$3.0427 \times 10^{+01}$	7.5310×10^{-05}	1.1961×10^{-04}	4.4664×10^{-06}
	max	9.6000×10^{-01}	$4.0376 \times 10^{+00}$	$1.5444 \times 10^{+02}$	5.6981×10^{-02}	9.0500×10^{-02}	2.5500×10^{-02}
	avg	4.0472×10^{-01}	$2.2578 \times 10^{+00}$	$6.3160 \times 10^{+01}$	3.8407×10^{-03}	6.1000×10^{-03}	3.4000×10^{-03}
	std	1.5560×10^{-01}	6.8144×10^{-01}	$2.7332 \times 10^{+01}$	4.5963×10^{-03}	7.3000×10^{-03}	**3.2000×10^{-03}**
F8	min	$-1.8386 \times 10^{+04}$	$-2.0976 \times 10^{+04}$	$-5.9428 \times 10^{+03}$	$-1.8028 \times 10^{+04}$	$-2.8632 \times 10^{+04}$	$-2.2321 \times 10^{+04}$
	max	$-1.2145 \times 10^{+04}$	$-1.4594 \times 10^{+04}$	$-3.4410 \times 10^{+03}$	$-6.8919 \times 10^{+03}$	$-1.0946 \times 10^{+04}$	$-6.0861 \times 10^{+03}$
	avg	$-1.5563 \times 10^{+04}$	$-1.7487 \times 10^{+04}$	$-4.4265 \times 10^{+03}$	$-1.1925 \times 10^{+04}$	$-1.8939 \times 10^{+04}$	$-1.3756 \times 10^{+04}$
	std	$1.5734 \times 10^{+03}$	$1.4071 \times 10^{+03}$	**$6.0086 \times 10^{+02}$**	$1.9219 \times 10^{+03}$	$3.0524 \times 10^{+03}$	$2.7817 \times 10^{+03}$
F9	min	$1.1416 \times 10^{+02}$	$1.3990 \times 10^{+02}$	$5.8660 \times 10^{+02}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$
	max	$2.2095 \times 10^{+02}$	$2.9538 \times 10^{+02}$	$7.5527 \times 10^{+02}$	5.3363×10^{-08}	8.4753×10^{-07}	3.5243×10^{-12}
	avg	$1.5189 \times 10^{+02}$	$1.9779 \times 10^{+02}$	$6.8793 \times 10^{+02}$	5.8313×10^{-11}	9.2614×10^{-10}	3.2855×10^{-14}
	std	$2.6688 \times 10^{+01}$	$3.3480 \times 10^{+01}$	$3.6026 \times 10^{+01}$	1.6940×10^{-09}	2.6905×10^{-08}	**1.9214×10^{-13}**
F10	min	$3.2842 \times 10^{+00}$	$6.2140 \times 10^{+00}$	$1.1792 \times 10^{+01}$	5.5922×10^{-16}	8.8818×10^{-16}	8.8818×10^{-16}
	max	$5.6556 \times 10^{+00}$	$9.5208 \times 10^{+00}$	$1.4675 \times 10^{+01}$	1.1134×10^{-06}	1.7683×10^{-05}	5.5582×10^{-08}
	avg	$4.3490 \times 10^{+00}$	$8.1384 \times 10^{+00}$	$1.2970 \times 10^{+01}$	2.3612×10^{-09}	3.7501×10^{-08}	5.8108×10^{-11}
	std	6.1408×10^{-01}	7.9216×10^{-01}	5.9008×10^{-01}	4.2042×10^{-07}	6.6772×10^{-07}	**1.7592×10^{-09}**
F11	min	$1.2383 \times 10^{+00}$	$6.2030 \times 10^{+00}$	$2.4270 \times 10^{+02}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$
	max	$7.9671 \times 10^{+00}$	$1.7892 \times 10^{+01}$	$4.7155 \times 10^{+02}$	2.0802×10^{-08}	3.3038×10^{-08}	9.8601×10^{-11}
	avg	$2.3921 \times 10^{+00}$	$1.1797 \times 10^{+01}$	$3.5998 \times 10^{+02}$	2.4523×10^{-11}	3.8949×10^{-11}	1.0061×10^{-13}
	std	$1.2688 \times 10^{+00}$	$3.3246 \times 10^{+00}$	$5.4707 \times 10^{+01}$	6.6659×10^{-10}	1.0587×10^{-09}	**3.1186×10^{-12}**
F12	min	$3.1538 \times 10^{+00}$	$1.3594 \times 10^{+01}$	$7.0627 \times 10^{+06}$	3.0348×10^{-02}	4.8200×10^{-02}	5.0800×10^{-02}
	max	$1.8358 \times 10^{+01}$	$4.3185 \times 10^{+01}$	$6.2160 \times 10^{+07}$	$4.4974 \times 10^{+00}$	$7.1429 \times 10^{+00}$	2.2810×10^{-01}
	avg	$6.9790 \times 10^{+00}$	$2.9772 \times 10^{+01}$	$2.7158 \times 10^{+07}$	2.2736×10^{-01}	3.6110×10^{-01}	1.0950×10^{-01}
	std	$2.7466 \times 10^{+00}$	$8.0168 \times 10^{+00}$	$1.1500 \times 10^{+07}$	3.9881×10^{-01}	6.3340×10^{-01}	**2.3400×10^{-02}**
F13	min	$5.6178 \times 10^{+01}$	$1.6066 \times 10^{+02}$	$4.2209 \times 10^{+07}$	$3.1910 \times 10^{+00}$	$5.0681 \times 10^{+00}$	$6.8748 \times 10^{+00}$
	max	$1.6312 \times 10^{+03}$	$1.4339 \times 10^{+04}$	$1.7979 \times 10^{+08}$	$4.1616 \times 10^{+01}$	$6.6096 \times 10^{+01}$	$9.9422 \times 10^{+00}$
	avg	$1.5526 \times 10^{+02}$	$3.2519 \times 10^{+03}$	$1.0042 \times 10^{+08}$	$7.7912 \times 10^{+00}$	$1.2374 \times 10^{+01}$	$9.9090 \times 10^{+00}$
	std	$2.8248 \times 10^{+02}$	$4.3759 \times 10^{+03}$	$3.9026 \times 10^{+07}$	$4.0171 \times 10^{+00}$	$6.3801 \times 10^{+00}$	**1.1620×10^{-01}**

Table 4. The Benchmark Test Functions were run over 1000 times with 500 dimensions

Fun.	Fit.	**PSO** [31] **500 Dim**	**SSA**[32] **500 Dim**	**FDO** [33] **500 Dim**	**mCOOT** [34] **500 Dim**	**COOT**[27] **500 Dim**	**ICOOT** **500 Dim**
F1	min	$2.9446 \times 10^{+04}$	$6.3491 \times 10^{+04}$	$2.4682 \times 10^{+05}$	2.8224×10^{-56}	4.4826×10^{-60}	3.3503×10^{-58}
	max	$5.5507 \times 10^{+04}$	$8.5472 \times 10^{+04}$	$3.8145 \times 10^{+05}$	1.2338×10^{-09}	1.9595×10^{-06}	9.9796×10^{-16}
	avg	$4.0022 \times 10^{+04}$	$7.5171 \times 10^{+04}$	$3.0623 \times 10^{+05}$	1.2963×10^{-11}	2.0589×10^{-09}	1.2798×10^{-18}
	std	$5.3768 \times 10^{+03}$	$5.1430 \times 10^{+03}$	$3.3204 \times 10^{+04}$	3.9063×10^{-10}	6.2042×10^{-08}	**3.2092×10^{-17}**
F2	min	$2.5901 \times 10^{+02}$	$3.8414 \times 10^{+02}$	$1.1302 \times 10^{+43}$	2.1855×10^{-32}	3.4711×10^{-34}	1.5704×10^{-27}
	max	$7.9103 \times 10^{+02}$	$4.5681 \times 10^{+02}$	$1.3960 \times 10^{+82}$	3.1481×10^{-04}	5.0000×10^{-03}	1.5360×10^{-08}
	avg	$3.1255 \times 10^{+02}$	$4.2923 \times 10^{+02}$	$8.7456 \times 10^{+80}$	4.2550×10^{-07}	6.7579×10^{-06}	2.9204×10^{-11}
	std	$1.2921 \times 10^{+02}$	$1.6948 \times 10^{+01}$	$3.4895 \times 10^{+81}$	1.0141×10^{-06}	1.6106×10^{-04}	**6.0164×10^{-10}**
F3	min	$4.4210 \times 10^{+05}$	$3.9886 \times 10^{+05}$	$3.1061 \times 10^{+06}$	7.1450×10^{-54}	1.1348×10^{-56}	6.0385×10^{-52}
	max	$8.4808 \times 10^{+05}$	$2.7517 \times 10^{+06}$	$9.0120 \times 10^{+06}$	5.6437×10^{-01}	$8.9635 \times 10^{+01}$	1.1620×10^{-14}
	avg	$6.6498 \times 10^{+05}$	$1.0046 \times 10^{+06}$	$5.5883 \times 10^{+06}$	5.6415×10^{-03}	8.9600×10^{-02}	2.4494×10^{-17}
	std	$1.0566 \times 10^{+05}$	$5.6095 \times 10^{+05}$	$1.5414 \times 10^{+06}$	1.7847×10^{-01}	$2.8345 \times 10^{+00}$	**5.0743×10^{-16}**
F4	min	$3.3074 \times 10^{+01}$	$2.7379 \times 10^{+01}$	$6.7901 \times 10^{+01}$	2.7104×10^{-31}	4.3048×10^{-32}	3.9154×10^{-29}
	max	$4.6635 \times 10^{+01}$	$3.9255 \times 10^{+01}$	$7.3807 \times 10^{+01}$	7.2407×10^{-03}	1.1500×10^{-02}	2.4044×10^{-09}
	avg	$3.7789 \times 10^{+01}$	$3.9255 \times 10^{+01}$	$7.1660 \times 10^{+01}$	7.5952×10^{-06}	1.2063×10^{-05}	3.3610×10^{-12}
	std	$2.9033 \times 10^{+00}$	$2.7189 \times 10^{+00}$	$1.6585 \times 10^{+00}$	2.2971×10^{-05}	3.6483×10^{-04}	**7.9552×10^{-11}**
F5	min	$1.0208 \times 10^{+07}$	$2.2798 \times 10^{+07}$	$3.4090 \times 10^{+08}$	$3.1353 \times 10^{+02}$	$4.9795 \times 10^{+02}$	$4.9764 \times 10^{+02}$
	max	$2.6888 \times 10^{+07}$	$3.7563 \times 10^{+07}$	$1.0330 \times 10^{+09}$	$1.0941 \times 10^{+04}$	$1.7377 \times 10^{+05}$	$4.9830 \times 10^{+02}$
	avg	$1.5187 \times 10^{+07}$	$3.0294 \times 10^{+07}$	$6.3648 \times 10^{+08}$	$2.6719 \times 10^{+03}$	$4.2436 \times 10^{+03}$	$4.9799 \times 10^{+02}$
	std	$4.3703 \times 10^{+06}$	$3.8318 \times 10^{+06}$	$1.7839 \times 10^{+08}$	$9.4507 \times 10^{+03}$	$1.5010 \times 10^{+04}$	**1.1600×10^{-01}**
F6	min	$3.0738 \times 10^{+04}$	$6.5766 \times 10^{+04}$	$2.3929 \times 10^{+05}$	$5.5376 \times 10^{+01}$	$8.7949 \times 10^{+01}$	$8.3821 \times 10^{+01}$
	max	$6.2133 \times 10^{+04}$	$8.6824 \times 10^{+04}$	$3.8058 \times 10^{+05}$	$2.2568 \times 10^{+03}$	$3.5844 \times 10^{+03}$	$1.0018 \times 10^{+02}$
	avg	$4.1158 \times 10^{+04}$	$7.5923 \times 10^{+04}$	$3.0010 \times 10^{+05}$	$1.3763 \times 10^{+01}$	$2.1858 \times 10^{+02}$	$9.2041 \times 10^{+01}$
	std	$6.9138 \times 10^{+03}$	$4.7918 \times 10^{+03}$	$3.6375 \times 10^{+04}$	$2.4408 \times 10^{+02}$	$3.8765 \times 10^{+02}$	**$2.8448 \times 10^{+00}$**
F7	min	$7.3681 \times 10^{+01}$	$1.6387 \times 10^{+02}$	$2.0082 \times 10^{+03}$	2.6109×10^{-05}	4.1468×10^{-05}	1.7397×10^{-05}
	max	$1.5434 \times 10^{+02}$	$2.8490 \times 10^{+02}$	$8.2848 \times 10^{+03}$	1.8448×10^{-01}	2.9300×10^{-01}	3.4800×10^{-02}
	avg	$1.0843 \times 10^{+02}$	$2.2242 \times 10^{+02}$	$4.6495 \times 10^{+03}$	4.8481×10^{-03}	7.7000×10^{-03}	3.5000×10^{-03}
	std	$1.9512 \times 10^{+01}$	$2.8015 \times 10^{+01}$	$1.7100 \times 10^{+03}$	9.2556×10^{-03}	1.4700×10^{-02}	**3.4000×10^{-03}**
F8	min	$-7.0866 \times 10^{+04}$	$-5.6095 \times 10^{+04}$	$-1.2157 \times 10^{+04}$	$-4.6130 \times 10^{+04}$	$-7.3266 \times 10^{+04}$	$-5.3359 \times 10^{+04}$
	max	$-4.4052 \times 10^{+04}$	$-3.6572 \times 10^{+04}$	$-7.0314 \times 10^{+03}$	$-1.5068 \times 10^{+04}$	$-2.3932 \times 10^{+04}$	$-1.3337 \times 10^{+04}$
	avg	$-5.7190 \times 10^{+04}$	$-4.8169 \times 10^{+04}$	$-9.5400 \times 10^{+03}$	$-3.0905 \times 10^{+04}$	$-4.9085 \times 10^{+04}$	$-3.2469 \times 10^{+04}$
	std	$5.6096 \times 10^{+03}$	$4.1384 \times 10^{+03}$	**$1.0932 \times 10^{+03}$**	$5.8077 \times 10^{+03}$	$9.2240 \times 10^{+03}$	$6.5859 \times 10^{+03}$
F9	min	$1.8818 \times 10^{+03}$	$2.3410 \times 10^{+03}$	3.9219 × 10+03	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$
	max	$2.5033 \times 10^{+03}$	$2.6965 \times 10^{+03}$	4.2545 × 10+03	4.3286×10^{-05}	6.8749×10^{-05}	1.6553×10^{-10}
	avg	$2.0918 \times 10^{+03}$	$2.5250 \times 10^{+03}$	4.0872 × 10+03	4.3624×10^{-08}	6.9285×10^{-08}	7.8489×10^{-13}
	std	$1.3734 \times 10^{+02}$	$8.9432 \times 10^{+01}$	8.9560 × 10+01	1.3689×10^{-06}	2.1741×10^{-06}	**7.8184×10^{-12}**
F10	min	$9.2688 \times 10^{+00}$	$1.1018 \times 10^{+01}$	$1.4307 \times 10^{+01}$	5.5922×10^{-16}	8.8818×10^{-16}	8.8818×10^{-16}
	max	$1.1663 \times 10^{+01}$	$1.1761 \times 10^{+01}$	$1.5081 \times 10^{+01}$	1.3045×10^{-07}	2.0718×10^{-04}	2.4513×10^{-08}
	avg	$1.0089 \times 10^{+01}$	$1.1386 \times 10^{+01}$	$1.4758 \times 10^{+01}$	3.5928×10^{-08}	5.7062×10^{-07}	4.2899×10^{-11}
	std	5.7592×10^{-01}	1.8048×10^{-01}	1.8176×10^{-01}	6.0487×10^{-08}	9.6068×10^{-06}	**9.5612×10^{-10}**
F11	min	$2.7295 \times 10^{+02}$	$5.7293 \times 10^{+02}$	$2.3689 \times 10^{+03}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$	$0.0000 \times 10^{+00}$
	max	$3.9745 \times 10^{+02}$	$8.2312 \times 10^{+02}$	$3.2686 \times 10^{+03}$	2.5808×10^{-06}	4.0989×10^{-05}	5.4512×10^{-13}
	avg	$3.4624 \times 10^{+02}$	$6.8206 \times 10^{+02}$	$2.7928 \times 10^{+03}$	2.7199×10^{-09}	4.3199×10^{-08}	7.7083×10^{-16}
	std	$3.6390 \times 10^{+01}$	$5.1201 \times 10^{+01}$	$2.3477 \times 10^{+02}$	8.1688×10^{-07}	1.2974×10^{-06}	**1.7813×10^{-14}**
F12	min	$2.1103 \times 10^{+05}$	$3.4673 \times 10^{+05}$	$1.4846 \times 10^{+08}$	2.7043×10^{-01}	4.2950×10^{-01}	4.3820×10^{-01}
	max	$4.7669 \times 10^{+06}$	$2.6073 \times 10^{+06}$	$1.8748 \times 10^{+09}$	$4.1987 \times 10^{+00}$	$6.6686 \times 10^{+00}$	6.4950×10^{-01}
	avg	$1.6715 \times 10^{+06}$	$1.0661 \times 10^{+06}$	$1.0715 \times 10^{+09}$	5.1371×10^{-01}	8.1590×10^{-01}	5.4710×10^{-01}
	std	$1.1747 \times 10^{+06}$	$5.0250 \times 10^{+05}$	$4.9917 \times 10^{+08}$	4.7814×10^{-01}	7.5940×10^{-01}	**3.3100×10^{-02}**
F13	min	$8.5888 \times 10^{+06}$	$1.7332 \times 10^{+07}$	$1.0189 \times 10^{+09}$	$3.1417 \times 10^{+01}$	$4.9898 \times 10^{+01}$	$4.9428 \times 10^{+01}$
	max	$4.3985 \times 10^{+07}$	$4.0517 \times 10^{+07}$	$3.9916 \times 10^{+09}$	$2.4258 \times 10^{+03}$	$3.8527 \times 10^{+02}$	$4.9935 \times 10^{+01}$

	avg	$2.2029 \times 10^{+07}$	$2.6786 \times 10^{+07}$	$2.0358 \times 10^{+09}$	$4.4490 \times 10^{+01}$	$7.0660 \times 10^{+01}$	$4.9890 \times 10^{+01}$
	std	$7.9839 \times 10^{+06}$	$5.5066 \times 10^{+06}$	$7.6943 \times 10^{+08}$	$2.1011 \times 10^{+01}$	$3.3371 \times 10^{+01}$	**3.2400×10^{-02}**

Table 5. The performance in the environment with irregular shaped obstacles (Case 1)

Run	**Map 1**			**Map 2**		
	Distance (m)	**Error (m)**	**Elapsed time (sec)**	**Distance (m)**	**Error (m)**	**Elapsed time (sec)**
1	13.2793	9.8505×10^{-04}	40.2606	14.1892	6.8188×10^{-04}	39.3076
2	13.1348	1.2000×10^{-03}	37.185	13.0472	9.5682×10^{-04}	47.4014
3	13.1557	5.5994×10^{-04}	39.1394	13.2252	1.3000×10^{-03}	43.6256
4	12.9956	1.0000×10^{-03}	45.0583	13.4820	9.9570×10^{-04}	56.9447
5	13.1023	6.6753×10^{-04}	49.6354	13.4929	6.8757×10^{-04}	48.1613
6	13.6819	9.8490×10^{-04}	39.7836	13.5341	8.6734×10^{-04}	38.1024
7	13.1572	5.3466×10^{-04}	36.9746	13.4909	1.4000×10^{-03}	43.6487
8	14.0919	7.3766×10^{-04}	47.2728	13.3086	5.1325×10^{-04}	37.7038
9	13.8775	7.4151×10^{-04}	37.4747	13.2252	1.3000×10^{-03}	37.9535
10	13.3207	6.6267×10^{-04}	53.4604	14.1691	1.3000×10^{-03}	57.4533
Mean	**13.3797**	**8.0739×10^{-04}**	**42.6245**	**13.5164**	**1.0000×10^{-03}**	**45.0302**

is made up of four irregular static obstacles (Map 1) and seven irregular static obstacles (Map 2) of different sizes. The starting position was (0, 0), and the goal position was (10, 10). The proposed ICOOT algorithm was used in static environments with the following settings: The number of coots is 80, the number of leaders is 2, and the minimum acceptance error is 0.2 m (error is defined as norm(robotCurrentPose (1:2)-GoalPosition(:)).where robotCurrentPose = [robotInitialLocation initialOrientation] and initialOrientation = 0, desired linear velocity for robot = 0.5 m/sec, robot wheel radius = 0.034 m, and Max Angular Velocity = Linear Velocity / Wheel Radius. The optimized function is given by Eq. (4)

(1) Case 1: The target is static

In this case, the mean distance for Map 1 is 13.3797m and the error is 8.0739×10^{-04} m and the elapsed time is 42.6245 sec; the mean distance for Map 2 is 13.5164 m and the mean error is 1.0000×10^{-03} m and the elapsed time is 45.0302 sec as shown in Fig. 11 and in Table 5.

The first map contains four irregularly shaped obstacles, while the second map contains seven irregularly shaped obstacles. The green dashed line represents the path that the robot took and avoided collision, while the robot is shown at the goal position in red. Later, the target will be fixed in the first and second maps, while the target will be moving in the third map, and the fourth map will contain ten obstacles moving at a variable speed.

Ten runs were made for each map, and the distance, the error distance from the target, and the time taken were measured.

(2) Case 2: The goal is dynamic

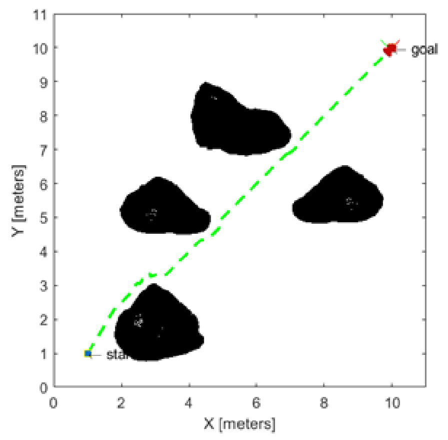
In this case, the goal is a dynamic that moves from one location to another at each successive time step interval. The velocity and direction of the dynamic goal are assumed to be random. The settings are the same as in case 1. As shown in Fig. 12 (a) and in Table 6, the mean distance for Map 3 is 11.9312m, the error is 9.9992×10^{-04} m, and the elapsed time is 42.5964 sec.

(3) RPP in a dynamic environment

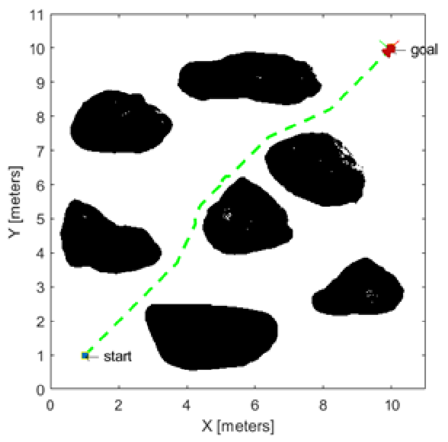
The proposed algorithm was put to the test in a dynamic environment with ten dynamic obstacles. The settings are the same as in static environment. The velocity (v_{obs}) and direction (θ_{obs}) of the dynamic obstacles are assumed to be random according to Eqs. (5) and (6). As shown in Fig. 12 (b) and in Table 6, the mean distance for Map 4 is 16.3937 m**,** the error is 9.5249×10^{-04} m**,** and the elapsed time is 49.1190 sec**.**

6. Conclusions

This paper came up with the ICOOT swarm optimization algorithm, and thirteen benchmark test functions are used to compare the proposed method to the standard algorithm. Also, using an ICOOT local search is an integrated strategy for the detection and avoidance of obstacles. Take into consideration the mobile robot's actual size and the kinematic model with specifications for robots. Algorithms were tested in both dynamic and static environments with various scenarios. They tried to minimize a multi-objective measure of path length and smoothness in these environments. According to simulation results, it shows that the ICOOT generates the best path for avoiding static and dynamic



(a)

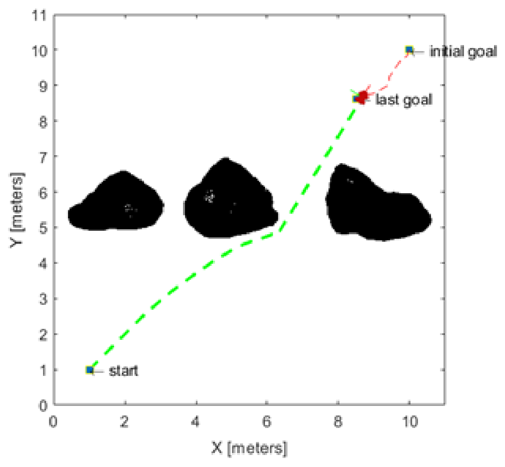


(b)

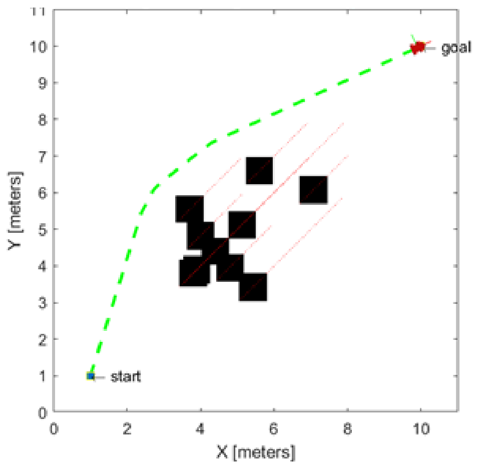
Figure. 11: (a) Map 1 and (b) Map2

Table 6. Cases 2 and 3's performance.

Run	Map 3			Map 4		
	Distance (m)	Error (m)	Elapsed time (sec)	Distance (m)	Error (m)	Elapsed time (sec)
1	14.0403	5.2472×10^{-04}	61.402138	16.0096	1.1000×10^{-03}	52.886948
2	10.9095	6.6049×10^{-04}	33.968483	15.5510	1.1000×10^{-03}	50.835557
3	11.5419	1.4000×10^{-03}	48.351470	17.6560	6.8392×10^{-04}	45.930273
4	10.9759	1.2000×10^{-03}	55.319431	16.6860	1.1000×10^{-03}	52.768155
5	11.4489	9.8912×10^{-04}	46.056632	15.4849	7.5757×10^{-04}	47.944359
6	11.7831	5.8099×10^{-04}	35.623026	16.3143	7.2201×10^{-04}	44.913749
7	11.9819	1.4000×10^{-03}	35.306464	17.6049	6.6138×10^{-04}	48.154225
8	11.7622	8.4387×10^{-04}	35.703954	16.4441	1.0000×10^{-03}	51.582104
9	12.1407	1.2000×10^{-03}	36.017066	15.9843	1.0000×10^{-03}	44.880337
10	12.7279	1.2000×10^{-03}	38.215630	16.2018	1.4000×10^{-03}	51.294447
Mean	**11.9312**	**9.9992×10^{-04}**	**42.5964**	**16.3937**	**9.5249×10^{-04}**	**49.1190**



(a)



(b)

Figure. 12: (a) Map 3 and (b) Map4

obstacles in an unknown complex environment with irregular static obstacles and according to the objective that is needed by the mobile robot to reach the goal position. The proposed method is compared to five swarm optimization strategies. The proposed algorithm gave the results were better than 84% in 30 dimensions. While it was 92% in 100 and 500 dimensions, it means the suggested algorithm is stable in most test functions and does not suffer from the growth of the problem size. The mean results show that this method is highly useful for robot paths from the start to the destination. The average mean distance for four complex arenas with 100m distance and three scenarios (fixed obstacle, fixed target, fixed obstacle, dynamic target, and dynamic obstacle and fixed target) is 13.8052.

The Matlab 2021a programming language is used to write the simulation code and run it on a computer with an Intel (R) Core (TM) i7-9750HF CPU @ 2.60GHz and 16.0 GB of RAM. It will be interesting to think about how the proposed ICOOT algorithm-based path planning will work on real mobile robots in future work.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

Jaafar Ahmed Abdulsaheb PhD candidate contributed to methodology, classification model proposed, software, and writing review and editing. Dheyaa Jasim Kadhim contributed to supervising the overall work and editing the paper.

References

[1] B. K. Patle, G. L. Babu, A. Pandey, D. R. K. Parhi, and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot", *Defence Technology, China Ordnance Society*, Vol. 15, No. 4. pp. 582–606, 2019, doi: 10.1016/j.dt.2019.04.011.

[2] P. A. M. Ehlert, "The use of Artificial Intelligence in Autonomous Mobile Robots", *Report on Research Project, Delft University of Technology*, Netherlands, 1999.

[3] J. M. Keil, "Decomposing a Polygon into Simpler Components", *SIAM Journal on Computing*, Vol. 14, No. 4, pp. 799–817, 1985, doi: 10.1137/0214056.

[4] C. Zhong, S. Liu, B. Zhang, Q. Lu, J. Wang, Q. Wu, and F. Gao, "A Fast On-line Global Path Planning Algorithm Based on Regionalized Roadmap for Robot Navigation," *in IFAC-PapersOnLine*, Vol. 50, No. 1, pp. 319–324, 2017, doi: 10.1016/j.ifacol.2017.08.053.

[5] U. O. Rosas, O. Montiel, and R. Sepúlveda, "Mobile robot path planning using membrane evolutionary artificial potential field", *Applied Soft Computing Journal*, Vol. 77, pp. 236–251, 2019, doi: 10.1016/j.asoc.2019.01.036.

[6] T. T. Mac, C. Copot, D. T. Tran, and R. D. Keyser, "Heuristic approaches in robot path planning: A survey", *Robotics and Autonomous Systems*, Vol. 86, pp. 13–28, 2016, doi: 10.1016/j.robot.2016.08.001.

[7] H. Miao, "Robot Path Planning in Dynamic Environments using Simulated Annealing Based Approach", *Master Thesis, Queensland University of Technology*, Queensland, Australia, March 2009.

[8] J. Ou and M. Wang, "Path planning for omnidirectional wheeled mobile robot by improved ant colony optimization", In: *Proc. of Chinese Control Conf., CCC, IEEE Computer Society*, Guangzhou, China, pp. 2668–2673, 2019, doi: 10.23919/ChiCC.2019.8866228.

[9] H. S. Dewang, P. K. Mohanty, and S. Kundu, "A Robust Path Planning for Mobile Robot Using Smart Particle Swarm Optimization", *Procedia Computer Science*, 2018, Vol. 133, pp. 290–297, doi: 10.1016/j.procs.2018.07.036.

[10] W. Wang, M. Cao, S. Ma, C. Ren, X. Zhu, and H. Lu, "Multi-robot odor source search based on Cuckoo search algorithm in ventilated indoor environment", In: *Proc. of 12th World Congress on Intelligent Control and Automation (WCICA)*, Guilin, China, pp. 1496–1501, 2016, doi: 10.1109/WCICA.2016.7578817.

[11] M. A. Hossain and I. Ferdous, "Autonomous robot path planning in dynamic environment using a new optimization technique inspired by bacterial foraging technique", *Robotics and Autonomous Systems*, Vol. 64, pp. 137–141, 2015, doi: 10.1016/j.robot.2014.07.002.

[12] P. K. Das, S. K. Pradhan, S. N. Patro, and B. K. Balabantaray, "Artificial Immune System Based Path Planning of Mobile Robot", *Studies in Computational Intelligence*, pp. 195–207, 2012, doi: 10.1007/978-3-642-25507-6_17.

[13] T. K. Dao, T. S. Pan, and J. S. Pan, "A multi-objective optimal mobile robot path planning based on whale optimization algorithm", In: *Proc. of IEEE 13th International Conference on Signal Processing (ICSP)*, Chengdu, China, pp. 337–342, 2016, doi: 10.1109/ICSP.2016.7877851.

[14] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning", *Procedia Computer Science*, Vol. 127, pp. 180–189, 2018, doi: 10.1016/j.procs.2018.01.113.

[15] D. Davis and P. Supriya, "Implementation of fuzzy-based robotic path planning", *Advances in Intelligent Systems and Computing*, Vol. 380, pp. 375–383, 2016, doi: 10.1007/978-81-322-2523-2_36.

[16] J. H. Zhang, Y. Zhang, and Y. Zhou, "Path planning of mobile robot based on hybrid multi-objective bare bones particle swarm optimization with differential evolution", *IEEE*

Access, Vol. 6, pp. 44542–44555, 2018, doi: 10.1109/ACCESS.2018.2864188.

[17] X. Liang, D. Kou, and L. Wen, "An Improved Chicken Swarm Optimization Algorithm and its Application in Robot Path Planning", *IEEE Access*, Vol. 8, pp. 49543–49550, 2020, doi: 10.1109/ACCESS.2020.2974498.

[18] F. Li, X. Fan, and Z. Hou, "A firefly algorithm with self-adaptive population size for global path planning of mobile robot", *IEEE Access*, Vol. 8, pp. 168951–168964, 2020, doi: 10.1109/ACCESS.2020.3023999.

[19] Y. Quan, H. Ouyang, C. Zhang, S. Li, and L. Q. Gao, "Mobile Robot Dynamic Path Planning Based on Self-Adaptive Harmony Search Algorithm and Morphin Algorithm", *IEEE Access*, Vol. 9, pp. 102758–102769, 2021, doi: 10.1109/ACCESS.2021.3098706.

[20] L. Shao, Q. Li, C. Li, and W. Sun, "Mobile Robot Path Planning Based on Improved Ant Colony Fusion Dynamic Window Approach", In: *Proc. of IEEE International Conference on Mechatronics and Automation (ICMA)*, Takamatsu, Japan, pp. 1100–1105, 2021. doi: 10.1109/ICMA52036.2021.9512795.

[21] O. Wahhab and A. A. Araji, "Path Planning and Control Strategy Design for Mobile Robot Based on Hybrid Swarm Optimization Algorithm", *International Journal of Intelligent Engineering and Systems*, Vol. 14, No. 3, pp. 565–579, 2021, doi: 10.22266/ijies2021.0630.48.

[22] Z. E. Kanoon, A. S. A. Araji, and M. N. Abdullah, "Enhancement of Cell Decomposition Path-Planning Algorithm for Autonomous Mobile Robot Based on an Intelligent Hybrid Optimization Method", *International Journal of Intelligent Engineering and Systems*, Vol. 15, No. 3, pp. 161–175, 2022, doi: 10.22266/ijies2022.0630.14.

[23] M. Fuad, T. Agustinah, and D. Purwanto, "Collision Avoidance of Multi Modal Moving Objects for Mobile Robot Using Hybrid Velocity Obstacles", *International Journal of Intelligent Engineering and Systems*, Vol. 13, No. 3, pp. 407–421, 2020, doi: 10.22266/ijies2020.0630.37.

[24] D. Jasim, K. Omar, and A. Hamad, "Improving IoT Applications Using a Proposed Routing Protocol", *Journal of Engineering*, Vol. 20, No. 11, pp. 50–62, Nov. 2014.

[25] N. Abbas and J. Abdulsaheb, "An Adaptive Multi-Objective Particle Swarm Optimization Algorithm for Multi-Robot Path Planning", *Journal of Engineering*, Vol. 22, No. 7, pp. 164–181, 2016.

[26] W. A. Mahmoud and D. J. Kadhim, "A Proposal Algorithm to Solve Delay Constraint Least Cost Optimization Problem", *Journal of Engineering*, Vol. 19, No. 1, pp. 155–160, 2013.

[27] I. Naruei and F. Keynia, "A new optimization method based on COOT bird natural life model", *Expert Systems with Applications*, Vol. 183, 2021, doi: 10.1016/j.eswa.2021.115352.

[28] H. Trenchard, "American coot collective on-water dynamics", *Nonlinear Dynamics*, Vol. 17, No. 2, pp. 183–203, 2012, doi: https://doi.org/10.48550/arXiv.1205.5929.

[29] R. Siegwart, I. R. Nourbakhsh and D. Scaramuzza, "Introduction to Autonomous Mobile Robots", *2nd Edition, MIT Press*, London, 2011.

[30] S. Mirjalili, "SCA: A Sine Cosine Algorithm for solving optimization problems", *Knowledge-Based Systems*, Vol. 96, pp. 120–133, 2016, doi: 10.1016/j.knosys.2015.12.022.

[31] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory", In: *MHS'95. Proc. of the Sixth International Symposium on Micro Machine and Human Science, Nagoya*, Japan, pp. 39–43, 1995, doi: 10.1109/MHS.1995.494215.

[32] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, and S. M. Mirjalili, "Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems", *Advances in Engineering Software*, Vol. 114, pp. 163–191, 2017, doi: 10.1016/j.advengsoft.2017.07.002.

[33] J. M. Abdullah and T. Ahmed, "Fitness Dependent Optimizer: Inspired by the Bee Swarming Reproductive Process", *IEEE Access*, Vol. 7, pp. 43473–43486, 2019, doi: 10.1109/ACCESS.2019.2907012.

[34] R. R. Mostafa, A. G. Hussien, M. A. Khan, S. Kadry, and F. A. Hashim, "Enhanced COOT optimization algorithm for Dimensionality Reduction", In: *Proc. of Fifth International Conference of Women in Data Science at Prince Sultan University (WiDS PSU)*, Riyadh, Saudi Arabia, pp. 43–48, 2022, doi: 10.1109/WiDS-PSU54548.2022.00020.

Table 7. A list of notations used in this paper's proposed algorithm equations

Symbol	**Meaning**
v_{obs}	Obstacle velocity
θ_{obs}	Obstacle direction
$rand(0,1)$	A random number between 0 and 1.
CP	COOT position
d	search space dimension

Received: May 28, 2022. Revised: July 17, 2022.

l *and* u	lower and upper search space limits
Q	random position
L	current iteration
$Iter$	The maximum number of iterations
NL	leader's number
LP	Number of leaders
$gBest$	The best place to be found
N	The number of coots
$D(j)$	Density
CCP	COOT center position
FCP	The first coot in the swarm
LCP	The last coot in the swarm
acc	Acceleration
α	The number of free points inside the circle with integer coordinates.
$\beta1$	represent the number of integer coordinates contained within the circle
$\beta2$	The minimum difference between two coots in the swarm
η	The number of coordinates accompanied by the coot