mathematics

MDPI

Article

A Metaheuristic Optimization Approach for Trajectory Tracking of Robot Manipulators

Carlos Lopez-Franco, Dario Diaz, Jesus Hernandez-Barragan, Nancy Arana-Daniel and Michel Lopez-Franco *

Computer Sciences Department, Universidad de Guadalajara, Guadalajara 44430, Mexico; carlos.lopez@cucei.udg.mx (C.L.-F.); dario.diaz5983@alumnos.udg.mx (D.D.); josed.hernandezb@academicos.udg.mx (J.H.-B.); nancy.arana@academicos.udg.mx (N.A.-D.)
* Correspondence: michel.lopez@academicos.udg.mx

Abstract: Due to the complexity of manipulator robots, the trajectory tracking task is very challenging. Most of the current algorithms depend on the robot structure or its number of degrees of freedom (DOF). Furthermore, the most popular methods use a Jacobian matrix that suffers from singularities. In this work, the authors propose a general method to solve the trajectory tracking of robot manipulators using metaheuristic optimization methods. The proposed method can be used to find the best joint configuration to minimize the end-effector position and orientation in 3D, for robots with any number of DOF.

Keywords: path tracking; manipulator robot; inverse kinematics; metaheuristic optimization; end-effector pose

MSC: 68T20; 68T40

check for updates

Citation: Lopez-Franco, C.; Diaz, D.; Hernandez-Barragan, J.; Arana-Daniel, N.; Lopez-Franco, M. A Metaheuristic Optimization Approach for Trajectory Tracking of Robot Manipulators. *Mathematics* **2022**, *10*, 1051. https://doi.org/10.3390/math10071051

Academic Editor: Ioannis G. Tsoulos

Received: 11 February 2022
Accepted: 20 March 2022
Published: 25 March 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

During the last decades, robot manipulators have seen a massive increase in industrial applications. This increment of robots also comes with a tremendous challenge, since each robot has a different structure and therefore each one requires a different algorithm to achieve its tasks. One of the most important of such tasks is trajectory tracking. In trajectory tracking the robot must follow a given path with a desired position and orientation. In this work, we propose a general approach to solve the trajectory path tracking problem for manipulator robots using metaheuristic optimization algorithms. The proposed method is independent of the structure of the robot and independent of the DOF of the robot.

Manipulator robots work in the joint space. However, to interact with the 3D world its trajectory is defined with respect to a 3D coordinate frame, Figure 1. For such reason, a transformation between the desired 3D trajectory in the 3D space to the joint space of the joint variables is required. To solve the trajectory tracking problem robots must be able to solve the inverse kinematic problem. The objective of inverse kinematics is to obtain the required manipulator joint values for a given end-effector position and orientation. The inverse kinematic problem is complex to solve since it depends on the robot structure. If the robot has more degrees of freedom than the required degrees of freedom to perform a task (redundant robot) then there could be infinite solutions. Traditionally there are three methods used to solve the inverse kinematics problem: algebraic, geometric, and iterative. These methods have some disadvantages, for example in the case of the geometric method, a closed-form solution for the first three joints must exist. There is no guarantee to find a closed-form solution with the algebraic methods. The problem with iterative methods is that they converge to a single solution, and the solution depends on the starting point.

The most common approach to solve the inverse kinematic problem is by obtaining a closed-form solution. Unfortunately, this approach is only applicable to robots with a simple kinematics structure. When the inverse kinematics of a robot are unsolvable with

a closed form, traditional numeric techniques are used. However, these methods can not converge to the correct solution if the initialization of the method is not properly estimated. In addition, multiple solutions may exist or no solution could be found if the Jacobian matrix has a singular configuration [1].

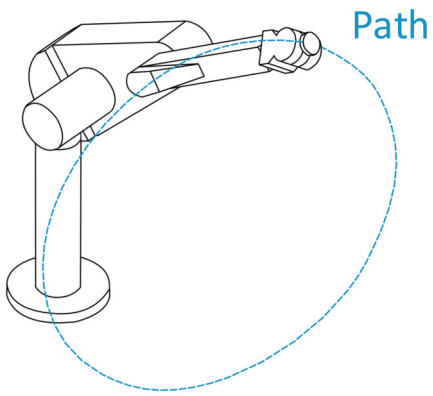

Figure 1. Trajectory tracking with a robot manipulator.

Overall, our work makes the following contribution:

- We present a novel path tracking method (MetaPAT) based on metaheuristic optimization methods for manipulator robots. The proposed method does not suffer from singularities when solving the inverse kinematics problem.
- The proposed approach is a general method, that can be used to solve the path tracking task, with robots of any number of DOF. The experiments were performed with robots with 5 DOF, 6 DOF, and a redundant robot with 7 DOF.
- We propose a novel objective function that combines effectively the position, orientation, and joint angles objectives functions. We show how to normalize each of these objective functions in order to define a weight objective function.
- To the best of our knowledge, this is the first work that uses quaternions in the objective function to solve the end-effector orientation error with metaheuristic optimization algorithms. Quaternions allow us to represent the orientation error effectively and can be combined with the position and joint objectives to solve the path tracking problem without singularities.

Related Work

There are different methods to solve the inverse kinematic (IK) problem. However, the most popular numeric approach is to use the Jacobian matrix to find a linear approximation to the IK problem. Several methodologies have been proposed for the computation or approximation of the Jacobian inverse, such as the Jacobian Transpose [2,3], the Damped Least Squares [4,5], Damped Least Squares with Singular Value Decomposition (SVD-DLS), and Selectively Damped Least Squares (SDLS) [4–7]. These solutions produce smooth postures, however many of these approaches suffer from high computational cost and singularity problems. Geometric methods depend on the structure of the robot and the number of its DOFs [8]. Another approach is the use of iterative methods, these methods suffer from redundant solutions or singularity problems. Due to such limitations, metaheuristic optimization methods appear as an interesting solution for the IK problem.

In [9–12] the authors propose the solution of the IK problem using metaheuristic methods. In [13], the authors propose a method called DEMPSO, which is a hybrid strategy based on Differential Evolution (DE) and Particle Swarm Optimization (PSO) for a 3-RPS parallel manipulator. In [14], the authors present a hybrid biogeography-based optimization (HBBO) algorithm, the proposed method is based on BBO and differential evolution (DE).

2. Problem Definition

A manipulator robot is created from a series of rigid bodies (links), which are connected through joints. The whole structure of a manipulator robot forms a kinematic chain. The beginning of the chain is constrained to the base, and the end-effector (tool or gripper) is connected to the end of the chain, Figure 2.

The structure of the manipulator robot is defined by the number of degrees of freedom (DOFs). Each DOF is associated with a robot joint, and each joint represents a joint variable. The pose (position and orientation) of the end-effector can be computed from the joint variables, this process is known as forward kinematics, see Figure 2. The opposite problem is known as the inverse kinematics problem which consists of the computation of the joint variables that correspond to a desired position and orientation. In the case of forward kinematics, a configuration of joint variables produces a unique pose. On the other hand, the inverse kinematic problem is more complex since there can be multiple joint configurations for the same pose. In the case of redundant robots, there can be infinite solutions. The equations to solve the inverse kinematics are in general nonlinear, and therefore is not always possible to find a closed-form solution.

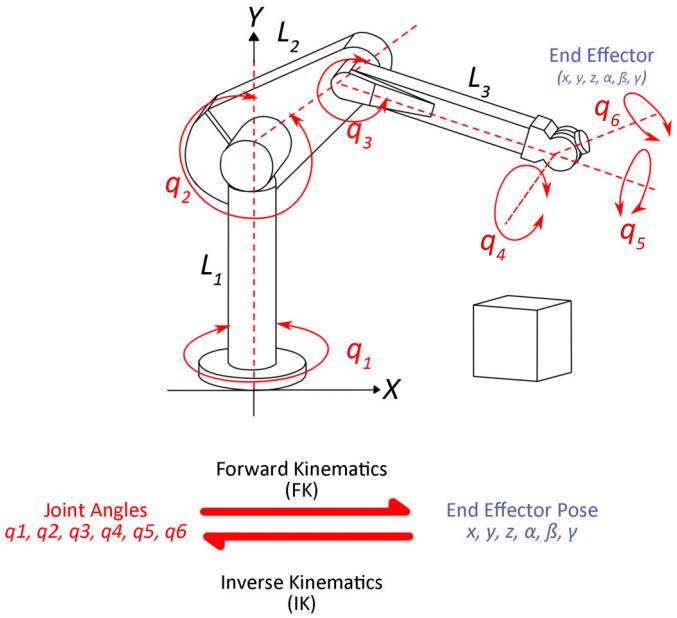

Figure 2. Manipulator robot, with relationship between forward and inverse kinematics.

2.1. Forward Kinematics

The forward kinematics for the joints parameters $\mathbf{r} = [r_1, r_2, r_3, \ldots, r_n]$ can be computed as

$$ {}^{0}T_n(\mathbf{r}) = {}^{0}A_1(r_1)\ {}^{1}A_2(r_2)\ {}^{2}A_3(r3)\ \cdots\ {}^{n-1}A_n \tag{1} $$

where n is the total number of degrees of freedom, and the ${}^{i-1}A_i(r_i)$ is a transformation matrix for each i-joint. The ${}^{i-1}T_n$ represents a homogeneous matrix that contains the position and orientation of the end effector pose.

Based on the Denavit-Hartenberg convention, the transformation matrix ${}^{i-1}A_i(r_i)$ is the transformation from the coordinate frame B_i to the coordinate frame B_{i-1}, this transformation is defined using the parameters of the joint i and link i, the transformation matrix is

$$ {}^{i-1}A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

where the variables θ_i, a_i, d_i and α_i are the Denavit-Hartenberg (DH) parameters associated with each joint and link i. The parameter θ_i represents the joint angle, the parameter a_i represents de link length, the parameter d_i represents the link offset, and α_i represents the link twist.

The pose of the end-effector can be written as

$$ {}^{0}T_n = \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3} $$

where $\mathbf{t}$ is a vector representing the position of the end-effector and R is a rotation matrix representing the orientation of the end-effector.

2.2. Inverse Kinematics

The inverse kinematic problem can be defined as the computation of the joint parameters given a desired end-effector pose (position and orientation). Mathematically, inverse kinematics can be defined as the search for the elements of the vector $\mathbf{r}$ when a transformation is given as a function of the joint variables $r_1, r_2, r_3 \ldots, r_n$, that is

$$ {}^{0}T_n(\mathbf{r}) = {}^{0}A_1(r_1)\,{}^{1}A_2(r_2)\,{}^{2}A_3(r3)\ \cdots\ {}^{n-1}A_n \tag{4} $$

Manipulator robots are commonly actuated in the joint variable space, whereas trajectories to be followed, or objects to be manipulated are expressed in the global coordinate frame. Therefore, the transformation between 3D pose information and joint variables is a fundamental process for any manipulator robot.

The aim of this work is the computation of the joint variables given the desired 3D pose, i.e., the computation of the inverse kinematics of a manipulator robot.

2.3. Metaheuristic Algorithms

As we can notice, the computation of the inverse kinematic for a robot is a complex problem. This complexity can be increased by the kinematic structure of the robot and also by the constraints of the desired task. In our case, we want to compute the joint variables of the robot for the desired pose and in addition, as the robot is following a path we will require that the new pose of the end-effector is also close to the previous pose. This last, requirement will help to avoid abrupt changes in robot joint variables.

To solve this problem we propose to use a metaheuristic algorithm. Metaheuristic optimization is an impressive research area for the solution of intractable optimization problems.

The use of metaheuristic algorithms has become more common in recent years. Some of the most used algorithms are: Genetics Algorithms (GA) [15], Particle Swarm Optimization (PSO) [16], Differential Evolution (DE) [17], Artificial Bee Colony (ABC) [18]. There are more recent algorithms like the proposed in [14], in this work the authors use an approach which is a hybrid combination of the Biogeography-Based Optimization (BBO) and DE, the approach is named Hybrid Biogeography-Based Optimization (HBBO). Another method used to solve inverse kinematics is the imperialist competitive algorithm (ICA) [19]. The Whale Optimization Algorithm (WOA), is a metaheuristic optimization algorithm that mimics the social behavior of humpback whales [20]. In [21], the authors present a Sine Cosine Algorithm (SCA) for solving optimization problems.

The time complexity of evolutionary algorithms can be defined as $O(Nd)$, where N is the number of iterations and d is the population size. Therefore, the time complexity for evolutionary algorithms is in the order of $O(n^2)$ [22]. In the computational intelligence

field, it is generally agreed to use nonparametric tests to analyze the results obtained by swarm intelligence or evolutionary algorithms.

In this work, we compare the DE, PSO, DEMPSO, HBBO, ICA, OE, WOA and SCA algorithms for robot manipulator trajectory tracking.

3. Methodology

To solve the trajectory tracking problem we have to take into account three components. The first component is the distance from the end-effector to the desired position. The second component is the difference between the end-effector orientation and the desired orientation. Finally, the third component is the difference between the current joint variables and the previous joint variables. Is clear, that the first component can be computed using the Euclidean distance between the two positions. On the other hand, the computation of the orientation difference has many more options. The last component is defined in the joint variable space, and it is different from the two previous components.

As we can notice, there are different metrics in each of the three components that we want to solve. This difference in the metrics must be taken into account when we define the objective function.

3.1. Position Error

In order to compute the difference between the current end-effector position t and the desired position t_d we use the Euclidean distance

$$d_{err} = \|t - t_d\|_2 \tag{5}$$

where $\|\|_2$ denotes a 2-norm.

3.2. Orientation Error

The selection of the orientation error is not clear as the position error. There are many representations

Given two coordinates systems B_1 and B_2 and let R_1 and R_2 denote the rotation matrices that describe the orientation of each system referenced to a common base frame B_0. The relative orientation between the frames B_1 and B_2 is defined by the rotation matrix

$$R_{21} = R_2 R_1^\top \tag{6}$$

The rotation matrix R_{21} has 9 elements while only 3 are independent, this representation is hard to manipulate, and for this reason, there are representations with fewer parameters. One effective way to represent a rotation is a quaternion. A quaternion [23] can be defined as

$$\eta = \cos(\phi/2) \tag{7}$$
$$q = \sin(\phi/2)\mathbf{u} \tag{8}$$

where ϕ represents the rotation angle, and $\mathbf{u}$ represents the axis of rotation [23].

The quaternion obeys [23] the following

$$\eta^2 + q^\top q = 1 \tag{9}$$

Now, let us define the two corresponding quaternions for each rotation as $\{\eta_1, q_1\}$ and $\{\eta_2, q_2\}$, respectively. The quaternion of the rotation matrix R_{21} is given by

$$\{\eta, q\} \tag{10}$$

where

$$\eta = \eta_1\eta_2 + q_1^\top q_2 \tag{11}$$

and where

$$q = \eta_1 q_2 - \eta_2 q_1 + q_1 \times q_2 \tag{12}$$

When the two frames coincide then $\eta_1 = \eta_2$ and $q_1 = q_2$ and from (9) we get

$$\eta = 1 \tag{13}$$
$$q = 0 \tag{14}$$

From the previous equation, we can notice that the element q can be used to compute the orientation error. In particular, the norm of q is 0 when the frames coincide and is 1 when they do not coincide. Therefore, we can define the orientation error with the following equation

$$q_{err} = \|\mathbf{q}\|_2 \tag{15}$$

3.3. Joint Variables Error

The last component to take into account is the difference between the previous joint variables $\mathbf{r}_{i-1}$ and the current joint variables $\mathbf{r}_i$. The objective of this component is to avoid abrupt changes between the previous joint variables and the current joint variables. To achieve this, let us analyze how the joint variable r_i is defined. In general, the joint variable is defined within a range from $-\pi$ to π. The difference between the joint variables $\mathbf{r}_{i-1}$ and $\mathbf{r}_i$ can be defined as

$$r_{abs} = abs(\mathbf{r}_i - \mathbf{r}_{i-1}).$$

From this equation we can notice that the maximum difference between the joint variables $\mathbf{r}_{i-1}$ and $\mathbf{r}_i$ is $r_{abs} = [\pi_1, \pi_2, \pi_3, \ldots, \pi_N]$. With that in mind, we propose to compute the joint variables error with the following equation

$$r_{err} = \frac{\|\mathbf{r}_i - \mathbf{r}_{i-1}\|_1}{n\pi}. \tag{16}$$

where $\|\|_1$ denotes a 1-norm, and where n represents the number of DOF of the robot. This equation provides us a normalized measurement of the difference between consecutive joint variables in a range from 0 to 1.

3.4. Objective Function

The objective function for the path tracking task will be defined by using the three components of the previous subsections. As we can notice the tracking problem has three functions to be optimized. To solve this, we use a weighted sum method, selecting scalar weights and minimizing the composite objective function. The proposed composite objective function is the following:

$$f = \kappa d_{err} + \lambda q_{err} + \mu r_{err} \tag{17}$$

where d_{err} is the position error defined in (5), q_{err} is the orientation error (15), and r_{err} is the error between consecutive joint variables (16). The constants κ, λ, and μ represent the weight scalar factors. The three weight scalar factors are positive and are chosen to give more weight to the position, then to the orientation, and finally to the previous joint variables difference. The reason for choosing such weights is because the highest priority of the robot is to position the end-effector on the desired position, then the robot should reduce the orientation error, and finally, if it is possible, we may choose the new joint variables close to the previous joints variables. If for example, the value of μ is bigger than the other weights then the robot will not move. If the value of *lambda* is bigger than the other two weights then the robot end-effector will match the current orientation with the desired orientation. However, it can be far from the desired position, which is not what we want. Therefore, for the path tracking problem that we want to solve in this work, the weights must satisfy $\mu < \lambda < \kappa$.

There are several methods to solve the inverse kinematics problem using metaheuristic optimizations algorithms like [24–29]. However, most of these methods are designed for a specific robot. In addition, and more important the objective function of these methods only considers the position error, which is only one term of our proposed objective function. In our approach we can set $\kappa = 1$, $\lambda = 0$, and $\mu = 0$, to consider only the position error. However, we are interested in a more general problem that not only considers the position error. In the results section, we show that the proposed approach can effectively solve the path tracking problem, which not only takes into account the position error, but also the orientation and joint errors. In addition, we present a general methodology to apply the proposed approach to robot manipulators with different structures and DOFs, using different metaheuristic optimization algorithms, see Algorithms 1 and 2.

3.5. Solution of the Path Tracking Problem

To solve the path tracking problem every metaheuristic algorithm will optimize the objective function (17). Where each individual of the metaheuristic algorithm will represent a possible joint variable solution $\mathbf{r} = [r_1, r_2, r_3, \ldots, r_n]$. The metaheuristic algorithm will find the best solution to minimize the objective function (17).

Before the computation of the objective function, we compute the forward kinematics (4) of the proposed individual. This operation will provide us with the current pose of the end-effector, which is what we want to minimize with respect to a desired pose.

We also must take into account that the range of a joint variable is constrained to the robot joint range constraints. For this reason, we must define the joints constraints as

$$\begin{aligned} \mathbf{r}_{low} &= [\theta_{min1}, \theta_{min2}, \theta_{min3}, \ldots, \theta_{min\,1}] \\ \mathbf{r}_{up} &= [\theta_{max1}, \theta_{max2}, \theta_{max3}, \ldots, \theta_{max\,1}] \end{aligned}$$

The optimal joint configuration can be found by solving the following constrained optimization:

$$\begin{aligned} &\min_{\mathbf{r}} f(\mathbf{r}), \\ \text{subject to } &\mathbf{r}_{low} < \mathbf{r} < \mathbf{r}_{up} \end{aligned}$$

4. Proposed Approach

To solve the path tracking problem we use Algorithm 1. The algorithm obtains the desired pose from the path, then it passes the desired pose to the metaheuristic algorithm which will compute the corresponding joint variables.

Algorithm 1 Trajectory following algorithm.

```
for each pose T_d in ∈ trajectory 𝒯 do
    r ← metaheuristic_inverse_kinematics(T_d)
    Move robot to desired pose using joints variables r
end
Follow desired trajectory
```

In order to compute the inverse kinematics, we propose the use Algorithm 2. This algorithm is a general scheme for the solution of the inverse kinematic problem that can be adapted to any metaheuristic optimization algorithm.

We can observe in Algorithm 2 that before evaluating the fitness function we must compute the forward kinematics using the current individual solution $\mathbf{x}_i$ and $\mathbf{x}'_i$. This is because the solutions provide us with the joint variables, and then the forward kinematics transform these joint variables into a 3D pose. Using the computed 3D pose and the desired 3D pose the fitness function evaluates how close is the current solution for the robot joint angles that generate the desired pose. Therefore, the result of the metaheuristic optimization is the joint variables $\mathbf{r}$ that solve the inverse kinematic problem.

Algorithm 2 Inverse kinematics algorithm.

```
Initialize the metaheurstic parameters, and population 𝒫
while stopping conditions not true do
    for each individual x_i ∈ 𝒫 do
        T_i ← compute_forward_kinematics(x_i)
        Evaluate the fitness, f(T_i)
        Generate the new vector x'_i
        T'_i ← compute_forward_kinematics(x'_i)
        if f(T'_i) < f(T_i) then
            x_i ← x'_i
        end
    end
    end
end
Return the individual with the best fitness as the solution
```

In Figure 3, we summarize the proposed approach for manipulator trajectory tracking. The proposed approach uses Algorithm 1 to follow the desired path. This algorithm, requires the computation of the inverse kinematics (Algorithm 2). To solve the inverse kinematic problem Algorithm 2 uses a metaheuristic algorithm, which objective is to find the best joint variables that minimize the cost function (17).

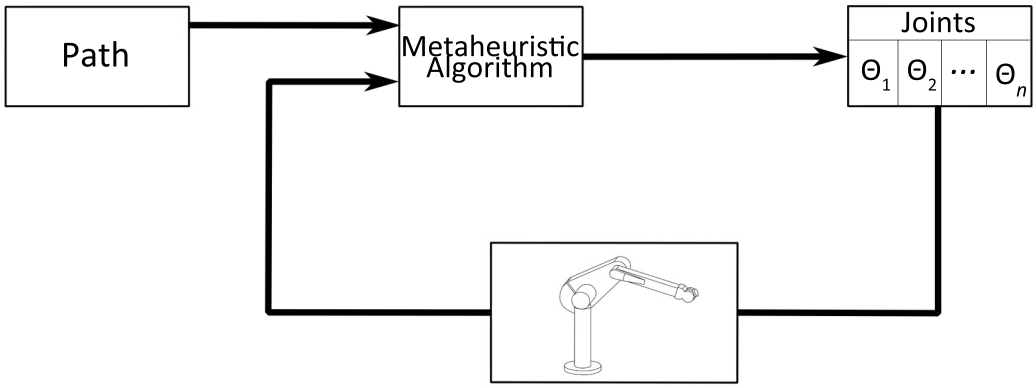

Figure 3. Schematic representation of the path tracking problem for manipulator robots.

5. Results

In this section, we present the performance of the proposed approach, which is analyzed and compared with the metaheuristic algorithms DE, PSO, DEMPSO, HBBO, ICA, WOA, and SCA in terms of accuracy of the estimated position, orientation, and time offsets. We also perform nonparametric statistical tests to contrast the results obtained in the simulation studies of these metaheuristic algorithms. The common parameters for the metaheuristic algorithms are the population size of 30 individuals, maximum iteration of 500, and a tolerance of 1×10^{-4}.

The parameters for the objective function (17), used in the tests, are: $\kappa = 0.7143$, $\lambda = 0.1786$ and $\mu = 0.1071$. These parameters correspond to the position, orientation, and joint variables weights, respectively. The objective function is a weighted sum of the three objectives (i.e., position, orientation, and joint errors). The weights κ, λ, and μ are subject to $\kappa + \lambda + \mu = 1$. In our case, the values for the weights are proportional to the importance of the position (5), orientation (15), and joint (16) errors. As we can notice the position of the end-effector has the highest priority, then the orientation, and lastly the joint variables error. A good rule of thumb is to define the parameters λ and μ as

$$\lambda = (1-\kappa)0.6251 \quad (18)$$
$$\mu = (1-\kappa)0.3749 \quad (19)$$

and the parameter κ with values from 0.6 to 0.8. In this way, the position error has the highest priority, and the parameters will obey the constraint $\mu < \lambda < \kappa$.

The DE algorithm used in these tests is the standard DE with mutation strategy DE/rand/1/bin. The scaling factor F, controls the amplification of the differential variations, and it was set to $F = 0.6$. The cross-over factor C_R, has a direct influence on the diversity of DE, and it was set to $C_R = 0.9$.

The PSO parameters are the cognitive components c_1, which measures the performance of a particle with respect to its previous performance, and the social component c_2, which quantifies the performance of a particle with respect to its neighbors. These parameters were set to $c_1 = 1.1312$ and $c_2 = 2.0149$. The weight factor w is used to stabilize the motion of the particles, and it was set to $w = 0.53514$. In the case of DEMPSO, the weight factor is gradually decreased from 0.9 to 0.1, the cognitive factor c_1 is gradually decreased from 2 to 0.1, and the cognitive factor c_2 is gradually increased from 0.1 to 2.

In the ICA algorithm, the number of imperialists is 10, the selection pressure 1, and the assimilation coefficient 1.5. Finally, the parameters settings for the HBBO algorithm are: amplification factor $F = 0.7$, cross-over $C = 0.8$, the maximum immigration and emigration are 1, the predetermined maximum mutation probability is 0.05. For the case of the WOA algorithm, the parameter a is linearly decreased from 2 to 0. For the SCA the parameter a is set to $a = 2$.

The experiments were performed with the robots Youbot, Puma 560, Baxter, and Fanuc AM120iB. These robots have different kinematic structures which are summarized in the Denavit–Hartenberg tables. Table 1 shows the Youbot DH parameters, this is a robot with 5 DOF. Table 2, shows the Puma 560 DH parameters, this is a robot with 6 DOF. The Baxter DH parameters are shown in Table 3, this is a robot with 7 DOF. The last robot is the Fanuc AM120iB and its DH parameters are shown in Table 4.

Table 1. DH table for KUKA Youbot manipulator.

Joint	a [m]	α [rad]	d [m]	θ [rad]
1	0.033	$\pi/2$	0.147	q_1
2	0.155	0	0	q_2
3	0.135	0	0	q_3
4	0	$\pi/2$	0	q_4
5	0	0	0.2175	q_5

Table 2. DH table for KUKA Youbot manipulator.

Joint	a [m]	α [rad]	d [m]	θ [rad]
1	0	$\pi/2$	0	q_1
2	0.4318	0	0	q_2
3	0.0203	$-\pi/2$	0.15	q_3
4	0	$\pi/2$	0.4318	q_4
5	0	$-\pi/2$	0	q_5
6	0	0	0	q_6

Table 3. DH table for KUKA Youbot manipulator.

Joint	a [m]	α [rad]	d [m]	θ [rad]
1	0.069	$-\pi/2$	0.270	q_1
2	0	$\pi/2$	0	q_2
3	0.069	$-\pi/2$	0.364	q_3
4	0	$\pi/2$	0	q_4
5	0.01	$-\pi/2$	0.374	q_5
6	0.01	$\pi/2$	0	q_6
7	0.01	0	0.28	q_7

Table 4. DH table for KUKA Youbot manipulator.

Joint	a [m]	α [rad]	d [m]	θ [rad]
1	0.1	$-\pi/2$	0	q_1
2	0.7	π	0	q_2
3	0.1	$-\pi/2$	0	q_3
4	0	$\pi/2$	-0.96	q_4
5	0	$-\pi/2$	0	q_5
6	0	0	-0.1	q_6

During the optimization process, the range of the joint variables $\mathbf{r}$ is constrained to the robot joint limits. The robot joint limits are shown in Equations (20)–(27). For clarity, values are shown in degrees, although in the optimization the algorithms use radians.

The joint limits for the Youbot robot are

$$\mathbf{r}_{min} = \begin{bmatrix} -169 & -65 & -150 & -102.2 & -167.5 \end{bmatrix} \quad (20)$$

$$\mathbf{r}_{max} = \begin{bmatrix} 169 & 65 & 150 & 102.2 & 167.5 \end{bmatrix} \quad (21)$$

The joint limits for the Puma 560 robot are

$$\mathbf{r}_{min} = \begin{bmatrix} -160 & -45 & -225 & -110 & -100 & -266 \end{bmatrix} \quad (22)$$

$$\mathbf{r}_{max} = \begin{bmatrix} 160 & 225 & 45 & 170 & 100 & 266 \end{bmatrix} \quad (23)$$

The joint limits for the Baxter robot are

$$\mathbf{r}_{min} = \begin{bmatrix} -141 & -123 & -173 & -3 & -175 & -90 & -175 \end{bmatrix} \quad (24)$$

$$\mathbf{r}_{max} = \begin{bmatrix} 51 & 60 & 173 & 150 & 175 & 120 & 175 \end{bmatrix} \quad (25)$$

The joint limits for the Fanuc AM 120iB robot are

$$\mathbf{r}_{min} = \begin{bmatrix} -185 & -100 & -185 & -200 & -140 & -270 \end{bmatrix} \quad (26)$$

$$\mathbf{r}_{max} = \begin{bmatrix} 185 & 160 & 273 & 200 & 140 & 270 \end{bmatrix} \quad (27)$$

5.1. Results of Test 1

In this study, we define a circle path for all the robots, the parameters of the circle depending on the reachability constraint of the robot. For every pose (position, and orientation) in the path, each metaheuristic algorithm runs 30 times. To qualify the results, we take the median value from the 30 runs.

The parameters for the Youbot robot are the center $C = (0.3, 0.2, 0.5)$ and the radius $r = 0.05$. For the Puma robot $C = (0.5, 0, 0.4)$ and the radius $r = 0.3$. The parameters for

the Fanuc robot are $C = (1.4, 0, 0.25)$ and $r = 0.5$. Finally, the parameters for the Baxter robot are center $C = (0.6, 0.6, 0.5)$ and the radius $r = 0.3$.

The desired orientation for all the robots is

$$R_d = Ry(\frac{\pi}{2}) \tag{28}$$

The metaheuristic algorithms used for this test are DE, PSO, DEMPSO, ICA, HBBO, WOA, and SCA, which were used with the four robots (Youbot, Puma, Fanuc, Baxter) in a circular path tracking tasks. The results of the circle path tracking are shown using boxplots (Figures 4–7); the position error results are also shown in the comparative Table 5.

We will start by analyzing the results of the Yobout robot, Figure 4. With respect to the position error (Figure 4a), we can observe that the DE, ICA and HBBO algorithms provide more consistent performance with smaller variance. From the distribution of the solution, we can observe that SCA has the worst performance than all the other algorithms its median is higher than all the algorithms and it also has the highest distribution. It is important to notice that the Youbot has only 5 DOF, this kind of robot has a lower range of motion. Despite that constraint, we can observe that the metaheuristics algorithms provide good results.

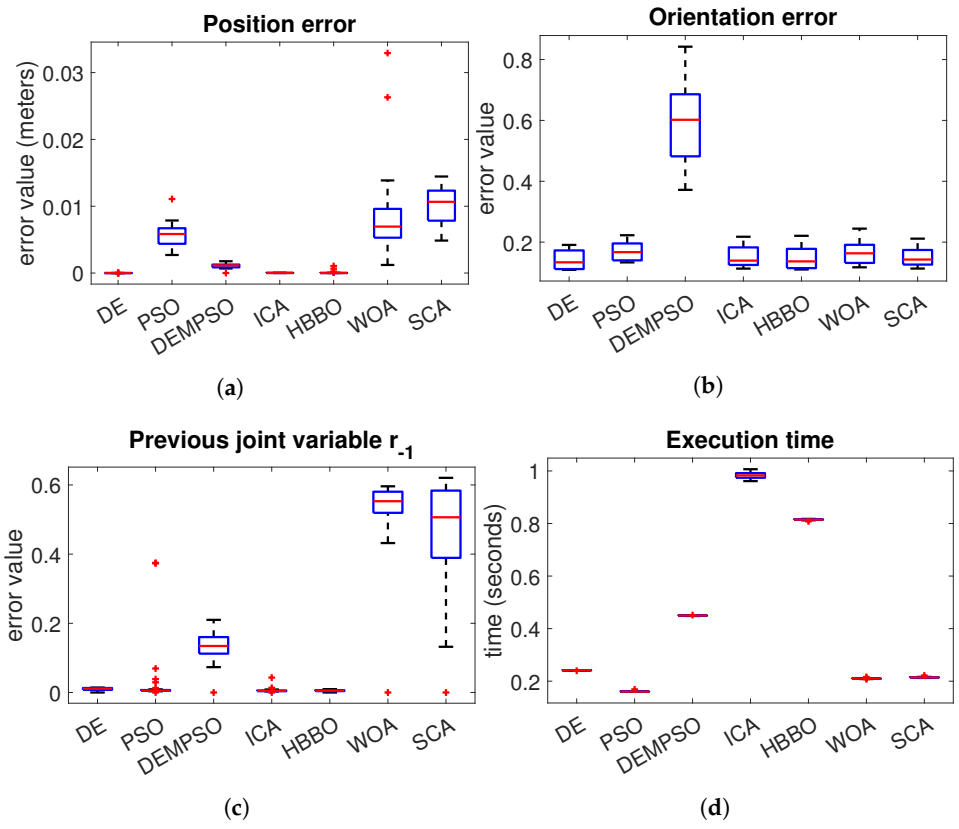

Figure 4. Circle path tracking results for the Youbot robot. (**a**) Youbot end-effector position error (5); (**b**) Youbot end-effector orientation error (15); (**c**) Youbot joint variables error (16); (**d**) Execution time for each point in the path.

In Figure 4b, a boxplot of the orientation error is shown. From the distribution of the solution in the boxplot, it could be seen that DE, PSO, ICA, HBBO, WOA, and SCO have similar performance, where DE has a slightly better performance than the others. In the orientation, the algorithm with the worst performance is the DEMPSO, which has the highest orientation error, and the larger distribution.

Figure 4c shows the difference between the current $\mathbf{r}_i$ solution and the previous solution $\mathbf{r}_{i-1}$. Similar to the orientation error, the algorithms DE, PSO, ICA, and HBBO

have similar performance, with some outliers in the PSO and ICA. The algorithm with the largest distribution is the SCA. The algorithm with the largest median error is the WOA.

The execution time of the trajectory following for the Youbot robot is shown in Figure 4d. From this plot, we can observe that the best performance was obtained by the PSO, where the PSO algorithm is slightly better than DE, WOA and SCA. We also have to notice that PSO could be slightly faster than DE. However, its position error is much higher. From the results of Figure 4, we can conclude that for the Youbot robot the DE algorithm has the best execution time with the lowest pose errors.

The results for the Puma robot are shown in Figure 5. In this case, we can observe that in the position error Figure 5a, the DE, and DEMPSOs have a more consistent performance with smaller variance. The HBBO algorithm has a fewer distribution than the PSO and ICA algorithms, but is higher than the DE and DEMPSO. The algorithm with the worst performance is the SCA, which has the highest error, and the larger distribution.

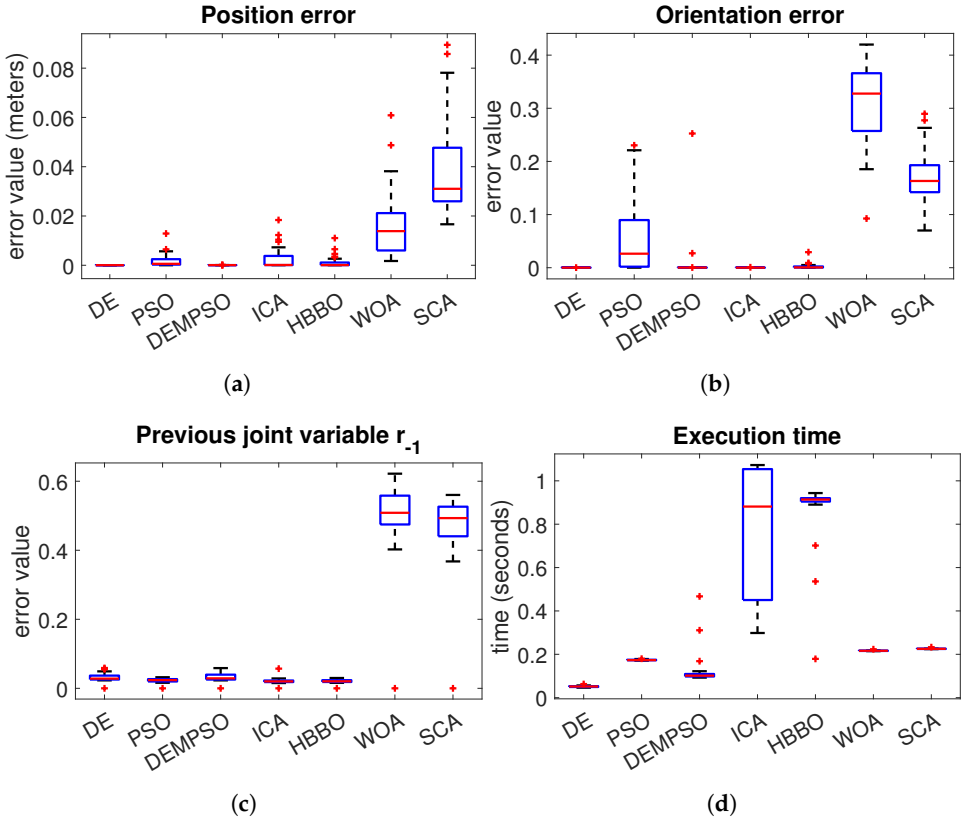

Figure 5. Circle path tracking results for the Puma robot. (**a**) Puma end-effector position error (5); (**b**) Puma end-effector orientation error (15); (**c**) Puma joint variables error (16); (**d**) Execution time for each point in the path.

The orientation error of the Puma robot is shown in Figure 5b. In this case, the algorithms DE, DEMPSO and ICA have the lowest median and lowest distribution with a few outliers. The HBBO has a median very close to the median of those three algorithms, but it has a higher distribution. In this case, the WOA algorithm has the worst performance, with the highest median and distribution.

In the case of the joint variables error for the Puma robot, we can observe in Figure 5c that the ICA and HBBO algorithms are the methods with lower median and lower distribution. The algorithms with the worst performance are the WOA and SCA. With respect to the execution time Figure 5d, we can observe that the algorithm with the best performance is the DE. From these boxplots, we can say that the DE algorithm has in general the best results with the Puma robot.

In the case of the Fanuc robot Figure 6a, we can notice that DE, ICA and HBBO have the best performance in the position error. With respect to the orientation error Figure 6b, the algorithm with the best performance is the DE. In the joint variables error Figure 6c, we have a very similar performance between the algorithms DE, PSO, ICA and HBBO, where the median of the DE is slightly lower. The algorithms with the worst performance are the WOA and SCA, which are the algorithms with the highest median error. In the execution time Figure 6d, the DE and PSO have the lowest median, where PSO has the lowest distribution. However, DE evolution performs better than PSO in the position and orientation errors.

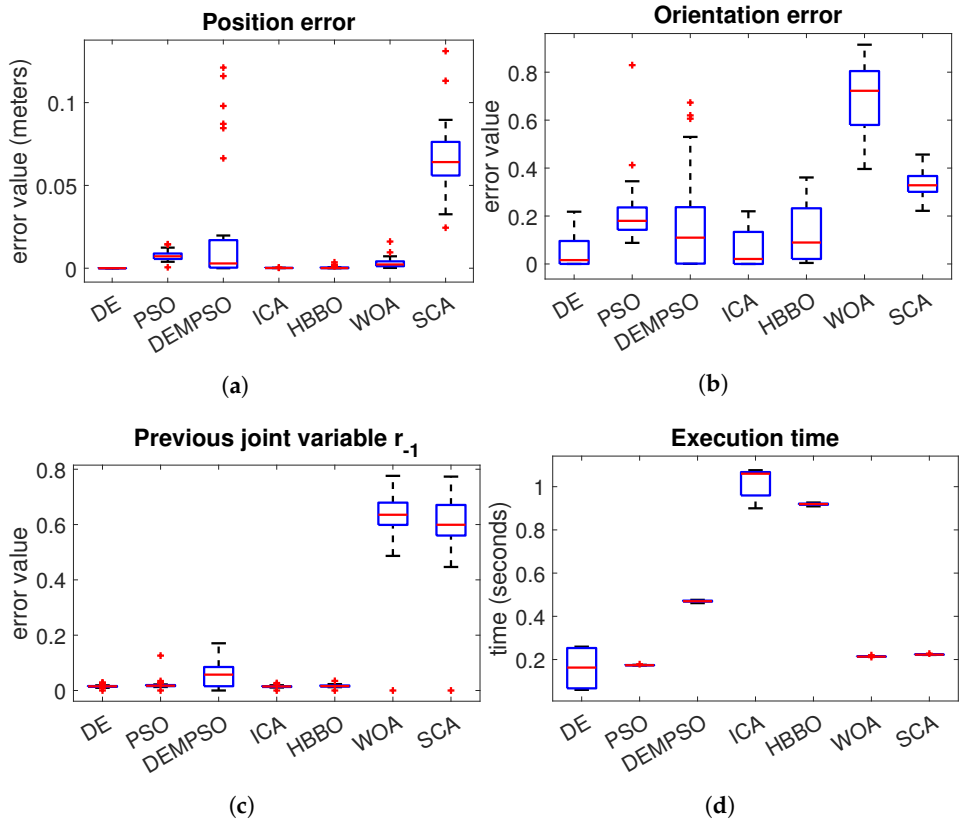

Figure 6. Circle path tracking results with the Fanuc robot. (**a**) Fanuc end-effector position error (5); (**b**) Fanuc end-effector orientation error (15); (**c**) Fanuc joint variables error (16); (**d**) Execution time for each point in the path.

Figure 7 shows the Baxter robot results. In Figure 7a we can observe that the performance for the position error is very similar for all the algorithms, with the exception of the PSO and SCA. In this case, the algorithm with the worst performance is the SCA. With respect to the orientation error Figure 7b, the DE has the lowest median, and the lowest dispersion, with some outliers that are lower than the median of all the others algorithms.

With respect to the joint variables error Figure 7c, we can observe that the algorithm with the lowest median and lowest dispersion is ICA. In this case the algorithms with the worst performance are the WOA and SCA. In the execution time for the Baxter robot tests Figure 7d, we can observe that DE has the best performance. From these results, DE has in general the best performance.

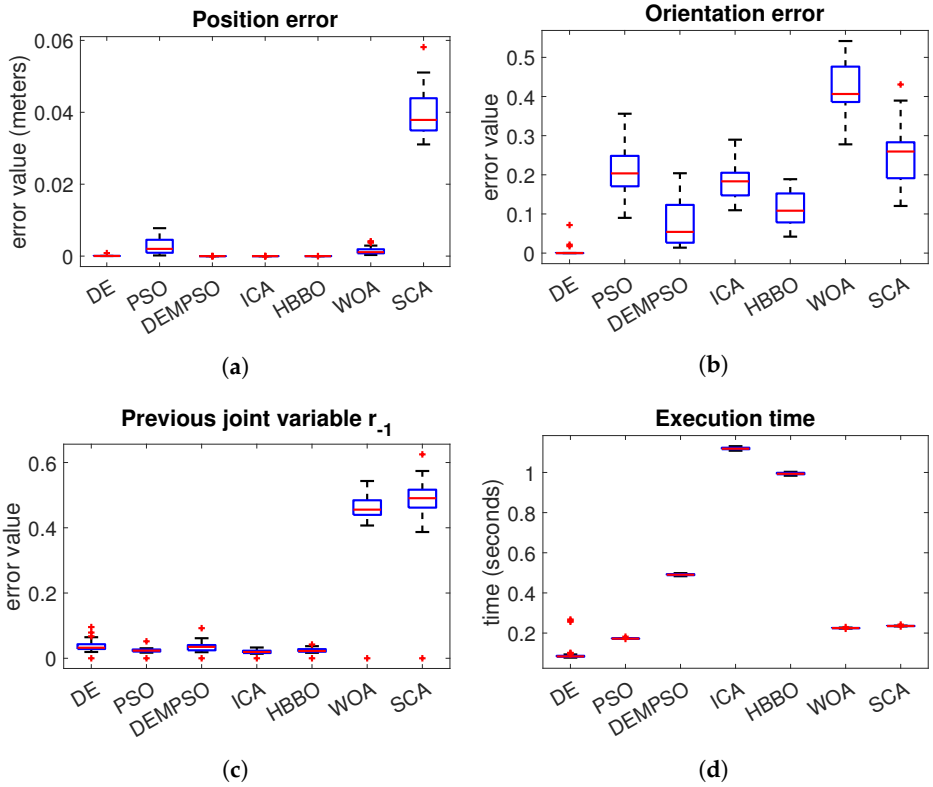

Figure 7. Circle path tracking results with the Baxter robot. (**a**) Baxter end-effector position error (5); (**b**) Baxter end-effector orientation error (15); (**c**) Baxter joint variables error (16); (**d**) Execution time for each point in the path.

Table 5. Position error comparative results for the circle path.

Robot		DE	PSO	DEMPSO	ICA	HBBO	WOA	SCA
Youbot	Mean	4.34×10^{-18}	5.74×10^{-3}	1.09×10^{-3}	1.44×10^{-5}	7.14×10^{-5}	8.55×10^{-3}	1.00×10^{-2}
	Std	9.61×10^{-18}	1.72×10^{-3}	3.37×10^{-4}	1.02×10^{-5}	2.16×10^{-4}	6.17×10^{-3}	2.80×10^{-3}
	Best	0	2.70×10^{-3}	0	2.10×10^{-6}	1.62×10^{-16}	1.21×10^{-3}	4.86×10^{-3}
	Worst	2.78×10^{-17}	1.11×10^{-2}	1.77×10^{-3}	3.79×10^{-5}	1.05×10^{-3}	3.29×10^{-2}	1.44×10^{-2}
Puma	Mean	3.44×10^{-5}	1.80×10^{-3}	6.68×10^{-5}	2.59×10^{-3}	1.21×10^{-3}	1.65×10^{-2}	3.97×10^{-2}
	Std	4.15×10^{-6}	2.66×10^{-3}	2.40×10^{-5}	4.52×10^{-3}	2.39×10^{-3}	1.37×10^{-2}	2.10×10^{-2}
	Best	2.64×10^{-5}	7.65×10^{-6}	0	3.05×10^{-5}	1.74×10^{-3}	2.18×10^{-5}	1.66×10^{-2}
	Worst	4.12×10^{-5}	1.29×10^{-2}	1.56×10^{-4}	1.84×10^{-2}	1.10×10^{-2}	6.09×10^{-2}	8.94×10^{-2}
Baxter	Mean	6.11×10^{-5}	2.72×10^{-3}	1.04×10^{-7}	2.33×10^{-9}	2.25×10^{-16}	1.50×10^{-3}	3.95×10^{-2}
	Std	1.43×10^{-4}	2.16×10^{-3}	3.60×10^{-7}	5.10×10^{-9}	1.58×10^{-16}	1.07×10^{-3}	6.37×10^{-3}
	Best	2.37×10^{-5}	1.81×10^{-4}	0	1.10×10^{-13}	0	3.06×10^{-4}	3.11×10^{-2}
	Worst	8.43×10^{-4}	7.77×10^{-3}	1.82×10^{-6}	1.92×10^{-8}	8.43×10^{-16}	4.13×10^{-3}	5.81×10^{-2}
Fanuc	Mean	1.30×10^{-5}	7.62×10^{-3}	2.14×10^{-2}	1.60×10^{-4}	4.74×10^{-4}	3.27×10^{-3}	6.64×10^{-2}
	Std	1.44×10^{-5}	3.08×10^{-3}	3.75×10^{-2}	6.84×10^{-5}	8.40×10^{-4}	3.21×10^{-3}	2.12×10^{-2}
	Best	1.98×10^{-15}	5.64×10^{-4}	0	7.97×10^{-5}	1.04×10^{-14}	1.90×10^{-4}	2.45×10^{-2}
	Worst	3.73×10^{-5}	1.45×10^{-2}	1.21×10^{-1}	3.35×10^{-4}	3.58×10^{-3}	1.61×10^{-2}	1.31×10^{-1}

Nonparametric Statistical Tests

Nonparametric statistical tests can be used to compare a significant improvement between algorithms. The main concepts of nonparametric tests are the null hypothesis and alternative hypothesis. The null hypothesis H_0 is a statement of no difference, whereas the alternative hypothesis H_1 represents the presence of a difference. A level of significance α is used to determine at which level the hypothesis may be rejected.

In this section, we present the nonparametric test results of the circle path tracking task. To compare the performance of the algorithms we use the popular sign test. In these test, we compare the DE against PSO, DEMPSO, ICA, HBBO, WOA and SCA with respect to the objective function (17) results.

In Table 6, we can observe that DE shows a significant improvement over all the metaheuristic algorithms with a level of significance of $\alpha = 0.05$. In Table 7, we also present the comparative results, we can observe that the DE mean is better than all the other algorithms.

Table 6. Sign test results for the objective function for the circle path.

Robot	DE	PSO	DEMPSO	ICA	HBBO	WOA	SCA
Youbot	Wins (+)	32	32	32	32	32	32
	Loses (−)	0	0	0	0	0	0
	p value	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Puma	Wins (+)	32	32	32	32	32	32
	Loses (−)	0	0	0	0	0	0
	p value	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Baxter	Wins (+)	32	32	32	32	32	32
	Loses (−)	0	0	0	0	0	0
	p value	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Fanuc	Wins (+)	32	32	31	32	32	32
	Loses (−)	0	0	1	0	0	0
	p value	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

5.2. Results of Test 2

The configuration parameters of the metaheuristics algorithm used for this test are the same values used in test 1. The path used for this test is a lemniscate path, with the equation $r = a\cos(\theta)$. The parameter a varies for each robot. The results of the lemniscate path tracking are shown using boxplots (Figures 8–11), the position error results are also shown in the comparative Table 8.

In the case of the Youbot robot the path center is $C = (0.3, 0.2, 0.5)$ and with $a = 0.05$. The results of the Youbot are shown in Figure 8. Figure 8a shows the results of the position error for the Youbot robot. Here we can observe that the algorithms with the best performance correspond to the DE and ICA algorithms, where DE has a slightly lower median and distribution.

The orientation error of the Youbot is presented in Figure 8b. In this case, the performance of the DE, PSO, ICA and HBBO is very similar. The algorithm with the lowest median is the DE, and the algorithm with the larger median and distribution is the DEMPSO. Similarly, with respect to the joint error Figure 8c, we can observe a similar performance with the DE, PSO, ICA and HBBO. Finally, with respect to the execution time, we can observe that the best algorithm is the PSO. However, we also should notice that the PSO is the algorithm with the worst performance in the position error. The second-best execution time algorithm is the DE. Although the performance of the algorithms DE, ICA and HBBO is similar, we could highlight that DE is the fastest of the three.

Table 7. Objective function comparative results for circle path.

Robot		DE	PSO	DEMPSO	ICA	HBBO	WOA	SCA
Youbot	Mean	2.54×10^{-2}	3.59×10^{-2}	1.08×10^{-1}	2.74×10^{-2}	2.67×10^{-2}	5.12×10^{-2}	3.69×10^{-2}
	Std	5.50×10^{-3}	5.81×10^{-3}	2.31×10^{-2}	6.04×10^{-3}	6.46×10^{-3}	7.19×10^{-3}	6.02×10^{-3}
	Best	1.94×10^{-2}	2.75×10^{-2}	6.67×10^{-2}	2.02×10^{-2}	1.97×10^{-2}	3.69×10^{-2}	2.73×10^{-2}
	Worst	3.41×10^{-2}	4.55×10^{-2}	1.51×10^{-1}	3.89×10^{-2}	3.95×10^{-2}	6.65×10^{-2}	4.84×10^{-2}
Puma	Mean	5.19×10^{-5}	1.32×10^{-2}	1.66×10^{-3}	2.01×10^{-3}	3.21×10^{-3}	7.74×10^{-2}	6.35×10^{-2}
	Std	4.76×10^{-6}	1.45×10^{-2}	8.03×10^{-3}	3.25×10^{-3}	4.15×10^{-3}	1.29×10^{-2}	1.59×10^{-2}
	Best	4.29×10^{-5}	7.00×10^{-4}	8.51×10^{-5}	9.64×10^{-5}	5.10×10^{-2}	9.94×10^{-5}	3.55×10^{-2}
	Worst	6.12×10^{-5}	4.83×10^{-2}	4.54×10^{-2}	1.31×10^{-2}	1.77×10^{-2}	1.04×10^{-1}	1.06×10^{-1}
Baxter	Mean	7.39×10^{-4}	4.24×10^{-2}	1.40×10^{-2}	3.23×10^{-2}	2.03×10^{-2}	8.35×10^{-2}	7.62×10^{-2}
	Std	2.67×10^{-3}	1.34×10^{-2}	1.07×10^{-2}	6.99×10^{-3}	7.11×10^{-3}	1.32×10^{-2}	1.51×10^{-2}
	Best	4.88×10^{-5}	1.62×10^{-2}	2.48×10^{-3}	1.96×10^{-2}	7.50×10^{-3}	5.29×10^{-2}	4.73×10^{-2}
	Worst	1.45×10^{-2}	6.78×10^{-2}	3.74×10^{-2}	5.17×10^{-2}	3.38×10^{-2}	1.05×10^{-1}	1.05×10^{-1}
Fanuc	Mean	9.90×10^{-3}	4.68×10^{-2}	4.69×10^{-2}	1.20×10^{-2}	2.59×10^{-2}	1.48×10^{-1}	1.21×10^{-1}
	Std	1.25×10^{-2}	2.23×10^{-2}	6.37×10^{-2}	1.44×10^{-2}	1.97×10^{-2}	2.34×10^{-2}	2.66×10^{-2}
	Best	4.64×10^{-5}	2.87×10^{-2}	2.09×10^{-4}	9.72×10^{-5}	5.11×10^{-3}	8.26×10^{-2}	7.42×10^{-2}
	Worst	3.89×10^{-2}	1.48×10^{-1}	2.09×10^{-1}	3.94×10^{-2}	6.46×10^{-2}	1.79×10^{-1}	1.99×10^{-1}

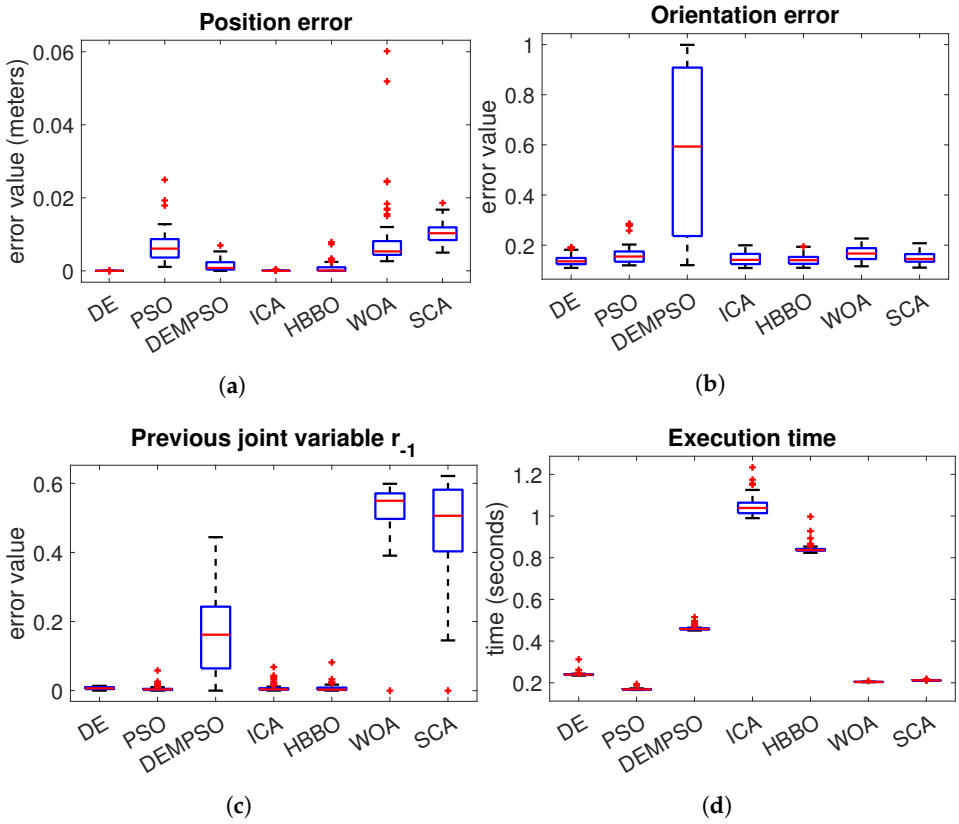

Figure 8. Lemniscate path tracking results for the Youbot robot. (**a**) Youbot end-effector position error (5); (**b**) Youbot end-effector orientation error (15); (**c**) Youbot joint variables error (16); (**d**) Execution time for each point in the path.

The path center for the Puma robot is $C = (0.5, 0, 0.4)$, the parameter of the equation is $a = 0.3$. Figure 9 shows the results for the Puma robot. In the position error Figure 9a we

can observe that DE, DEMPSO and ICA have similar performance. However, we can also notice that the DE and DEMPSO do not have outliers.

In the case of the orientation error Figure 9b, the algorithms DE, DEMPSO and ICA show similar performance, with very low medians, with DE without outliers. For the joint error Figure 9c, we can observe that PSO, ICA, and HBBO have similar performance, the median of DE and DEMPSO are slightly higher than the medians of those three algorithms. Finally, with respect to the execution time the algorithm with the best performance is the DE, Figure 9d.

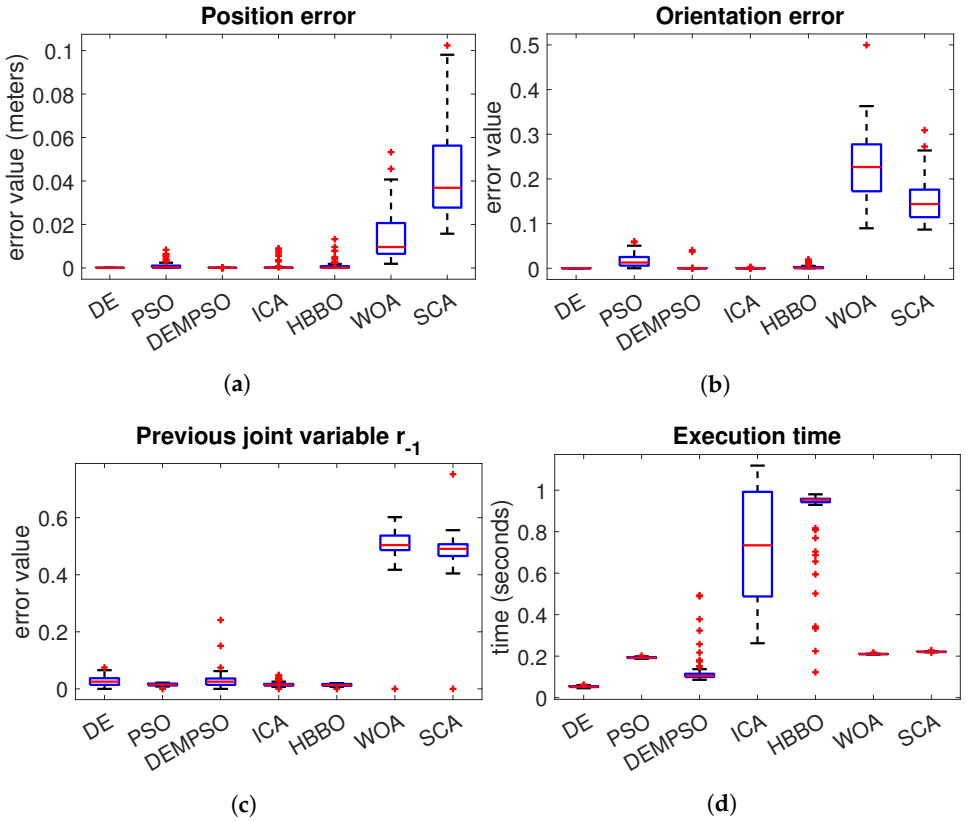

Figure 9. Lemniscate path tracking results for the Puma robot. (**a**) Puma end-effector position error (5); (**b**) Puma end-effector orientation error (15); (**c**) Puma joint variables error (16); (**d**) Execution time for each point in the path.

The center of the path for the Fanuc robot is $C = (1.4; 0; 0.25)$, and the lemniscate parameter $a = 0.5$. The results of the Fanuc robot are shown in Figure 10. In the position error Figure 10a, we can observe that the algorithms with the best performance are the DE and ICA algorithms, which have the lowest median and distribution. In the orientation error Figure 10b, we can notice that the algorithm with the best performance is the DE. With respect to the joint error Figure 10c, all the algorithms, with exception of the DEMPSO algorithm, have a similar performance with DE with a slightly lower median. Finally, the best execution time is obtained by the PSO algorithm. However, the DE, which has the second-best execution time performance, has a better performance in position and orientation.

The path parameters for the Baxter robot are $C = (0.6, 0.6, 0.5)$, and $a = 0.3$. The results for the Baxter robot are shown in Figure 11. In the case of the position error Figure 11a, we can observe that the algorithms DE, DEMPSO, ICA and HBBO have similar performance, the DE has some outliers. In the orientation error the best performance was obtained with the DE, which has the lowest median and a very narrow distribution. With respect to the

joint error Figure 11c, the best performance is obtained by the ICA algorithm. Finally, in the execution time Figure 11d, the best performance was obtained by the DE algorithm.

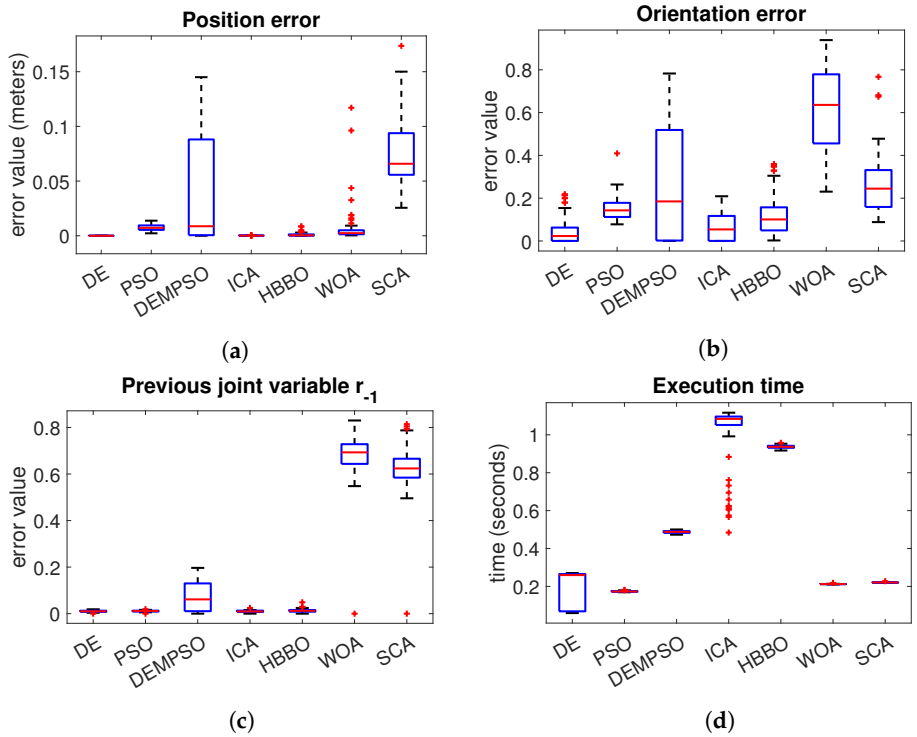

Figure 10. Lemniscate path tracking results with the Fanuc robot. (**a**) Fanuc end-effector position error (5); (**b**) Fanuc end-effector orientation error (15); (**c**) Fanuc joint variables error (16); (**d**) Execution time for each point in the path.

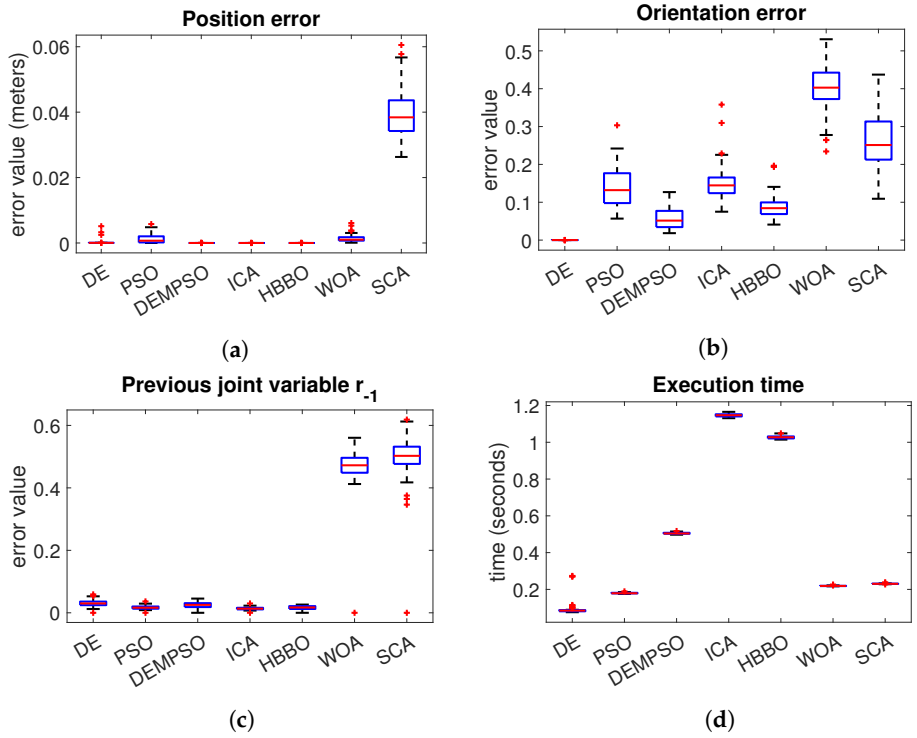

Figure 11. Lemniscate path tracking results with the Baxter robot. (**a**) Baxter end-effector position error (5); (**b**) Baxter end-effector orientation error (15); (**c**) Baxter joint variables error (16); (**d**) Execution time for each point in the path.

Table 8. Position error results for the lemniscate path.

Robot		DE	PSO	DEMPSO	ICA	HBBO	WOA	SCA
Youbot	Mean	2.78×10^{-12}	6.65×10^{-3}	1.48×10^{-3}	4.62×10^{-5}	7.94×10^{-4}	8.42×10^{-3}	1.04×10^{-2}
	Std	2.37×10^{-11}	4.15×10^{-3}	1.49×10^{-3}	7.77×10^{-5}	1.48×10^{-3}	9.26×10^{-3}	2.61×10^{-3}
	Best	0	1.09×10^{-3}	0	5.24×10^{-8}	0	2.64×10^{-3}	4.99×10^{-3}
	Worst	2.03×10^{-10}	2.49×10^{-2}	6.96×10^{-3}	3.80×10^{-4}	7.89×10^{-3}	6.02×10^{-2}	1.86×10^{-2}
Puma	Mean	3.41×10^{-5}	1.01×10^{-3}	5.95×10^{-5}	7.43×10^{-4}	1.05×10^{-3}	1.43×10^{-2}	4.43×10^{-2}
	Std	4.27×10^{-6}	1.69×10^{-3}	1.23×10^{-5}	1.99×10^{-3}	2.36×10^{-3}	1.15×10^{-2}	2.13×10^{-2}
	Best	2.37×10^{-5}	4.49×10^{-6}	0	6.18×10^{-6}	1.92×10^{-3}	2.29×10^{-5}	1.57×10^{-2}
	Worst	4.35×10^{-5}	8.35×10^{-3}	9.29×10^{-5}	8.87×10^{-3}	1.32×10^{-2}	5.33×10^{-2}	1.02×10^{-1}
Baxter	Mean	1.85×10^{-4}	1.23×10^{-3}	2.32×10^{-7}	4.05×10^{-8}	1.55×10^{-7}	1.33×10^{-3}	3.97×10^{-2}
	Std	7.54×10^{-4}	1.36×10^{-3}	1.36×10^{-6}	1.25×10^{-7}	1.00×10^{-6}	1.06×10^{-3}	7.65×10^{-3}
	Best	2.20×10^{-5}	6.50×10^{-8}	0	1.89×10^{-12}	1.11×10^{-16}	8.03×10^{-5}	2.63×10^{-2}
	Worst	5.13×10^{-3}	5.78×10^{-3}	1.13×10^{-5}	9.53×10^{-7}	7.88×10^{-6}	6.10×10^{-3}	6.05×10^{-2}
Fanuc	Mean	1.01×10^{-5}	7.44×10^{-3}	4.28×10^{-2}	1.36×10^{-4}	1.04×10^{-3}	7.48×10^{-3}	7.66×10^{-2}
	Std	1.45×10^{-5}	2.93×10^{-3}	4.95×10^{-2}	5.28×10^{-5}	1.68×10^{-3}	1.82×10^{-2}	3.28×10^{-2}
	Best	2.22×10^{-16}	2.25×10^{-3}	0	4.34×10^{-5}	4.76×10^{-16}	3.22×10^{-4}	2.56×10^{-2}
	Worst	4.24×10^{-5}	1.38×10^{-2}	1.45×10^{-1}	3.66×10^{-4}	8.67×10^{-3}	1.17×10^{-1}	1.74×10^{-1}

Nonparametric Statistical Tests

In this section, we present the nonparametric test results of the lemniscate path tracking task. To compare the performance of the algorithms we use the sign test. In these tests, we compare the DE against PSO, DEMPSO, ICA, HBBO, WOA, and SCA with respect to the objective function (17) results.

In Table 9, we can observe that DE shows a significant improvement over all the metaheuristic algorithms with a level of significance of $\alpha = 0.05$. In Table 10, we also present the comparative results, we can observe that the DE mean is better than all the other algorithms.

Table 9. Sign test results for the objective function for the lemniscate path.

Robot	DE	PSO	DEMPSO	ICA	HBBO	WOA	SCA
Youbot	Wins (+)	73	73	49	51	73	73
	Loses (−)	0	0	24	22	0	0
	p value	0.0000	0.0000	0.0046	0.0009	0.0000	0.0000
Puma	Wins (+)	73	73	73	73	73	73
	Loses (−)	0	0	0	0	0	0
	p value	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Baxter	Wins (+)	73	73	73	73	73	73
	Loses (−)	0	0	0	0	0	0
	p value	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Fanuc	Wins (+)	66	66	54	58	73	73
	Loses (−)	7	7	19	15	0	0
	p value	0.0000	0.0000	0.0001	0.0000	0.0000	0.0000

Table 10. Objective function comparative results for lemniscate path

Robot		DE	PSO	DEMPSO	ICA	HBBO	WOA	SCA
Youbot	Mean	2.58×10^{-2}	3.45×10^{-2}	1.24×10^{-1}	2.68×10^{-2}	2.69×10^{-2}	4.60×10^{-2}	3.64×10^{-2}
	Std	4.09×10^{-3}	7.45×10^{-3}	6.52×10^{-2}	5.00×10^{-3}	4.58×10^{-3}	7.21×10^{-3}	4.30×10^{-3}
	Best	2.01×10^{-2}	2.43×10^{-2}	3.03×10^{-2}	1.98×10^{-2}	1.98×10^{-2}	3.27×10^{-2}	3.06×10^{-2}
	Worst	3.55×10^{-2}	6.39×10^{-2}	3.97×10^{-1}	3.76×10^{-2}	3.97×10^{-2}	6.36×10^{-2}	4.62×10^{-2}
Puma	Mean	5.06×10^{-5}	5.40×10^{-3}	2.96×10^{-4}	5.84×10^{-4}	2.51×10^{-3}	6.48×10^{-2}	6.26×10^{-2}
	Std	4.41×10^{-6}	4.57×10^{-3}	1.24×10^{-3}	1.41×10^{-3}	3.07×10^{-3}	1.71×10^{-2}	1.63×10^{-2}
	Best	3.91×10^{-5}	4.76×10^{-4}	8.35×10^{-5}	9.37×10^{-5}	3.53×10^{-2}	9.45×10^{-5}	3.49×10^{-2}
	Worst	6.09×10^{-5}	2.40×10^{-2}	8.38×10^{-3}	6.33×10^{-3}	1.42×10^{-2}	1.14×10^{-1}	1.04×10^{-1}
Baxter	Mean	1.63×10^{-4}	2.87×10^{-2}	1.04×10^{-2}	2.72×10^{-2}	1.59×10^{-2}	7.82×10^{-2}	7.66×10^{-2}
	Std	5.37×10^{-4}	1.27×10^{-2}	5.35×10^{-3}	8.29×10^{-3}	6.09×10^{-3}	1.23×10^{-2}	1.71×10^{-2}
	Best	4.68×10^{-5}	1.04×10^{-2}	3.27×10^{-3}	1.34×10^{-2}	7.32×10^{-3}	4.68×10^{-2}	4.35×10^{-2}
	Worst	3.67×10^{-3}	6.35×10^{-2}	2.58×10^{-2}	6.40×10^{-2}	3.81×10^{-2}	1.01×10^{-1}	1.10×10^{-1}
Fanuc	Mean	8.44×10^{-3}	3.49×10^{-2}	7.73×10^{-2}	1.18×10^{-2}	2.26×10^{-2}	1.43×10^{-1}	1.19×10^{-1}
	Std	1.12×10^{-2}	1.01×10^{-2}	8.18×10^{-2}	1.06×10^{-2}	1.65×10^{-2}	3.30×10^{-2}	3.60×10^{-2}
	Best	4.50×10^{-5}	2.27×10^{-2}	2.40×10^{-4}	9.85×10^{-5}	3.20×10^{-3}	6.03×10^{-2}	5.87×10^{-2}
	Worst	3.91×10^{-2}	8.90×10^{-2}	2.43×10^{-1}	3.74×10^{-2}	6.44×10^{-2}	2.35×10^{-1}	2.06×10^{-1}

5.3. Results of Test 3

In this section, we present the convergences curves for the metaheuristic algorithms DE, PSO, DEMPSO, HBBO, ICA, WOA and SCA. For these tests, we have chosen an arbitrary position for each robot, with the desired orientation of $R_d = Ry(\frac{\pi}{2})$. Each algorithm runs 30 times for each robot pose; we chose the best solution and plotted its convergence curve.

Figure 12a shows the convergence curves for the Baxter robot. In this case, we can notice that all the algorithms, except for WOA and SCA, show an accelerated convergence behavior. The SCA algorithm converges toward the optimum only in final iterations. The WOA algorithm converges rapidly but it gets stuck in a local minimum. The desired position of the end-effector for this test was $x_d = [0.6000, 0.6000, 0.8000]$.

In Figure 12b, we present the convergence curves for the Fanuc robot. In this case, we can notice that all the algorithms, with the exception of SCA, show an accelerated convergence behavior. The WOA algorithm converges toward the optimum only in final iterations. For the Fanuc robot the desired position was $x_d = [1.4000, 0, 0.7500]$.

The results for the Puma robot are shown in Figure 12c. We can observe that all the algorithms, with the exception of WOA and SCA, show a rapid convergence behavior. The WOA and SCA converge to a local minimum, and the SCA converges at the final iterations. In this test, the desired position was $x_d = [0.5000, 0, 0.7000]$

Figure 12d shows the convergence curves for the Youbot robot. In this case, we can notice that all the algorithms, with the exception of SCA, show an accelerated convergence behavior. The SCA algorithm converges toward the optimum after passing almost half of the iterations. We should notice that the Yobout robot has only five DOFs, and for this reason, the robot is not able to achieve all the position and orientations in the 3D space. However, the algorithm provides an acceptable solution. For this test, the desired position was $x_d = [0.3000, 0.2359, 0.5348]$.

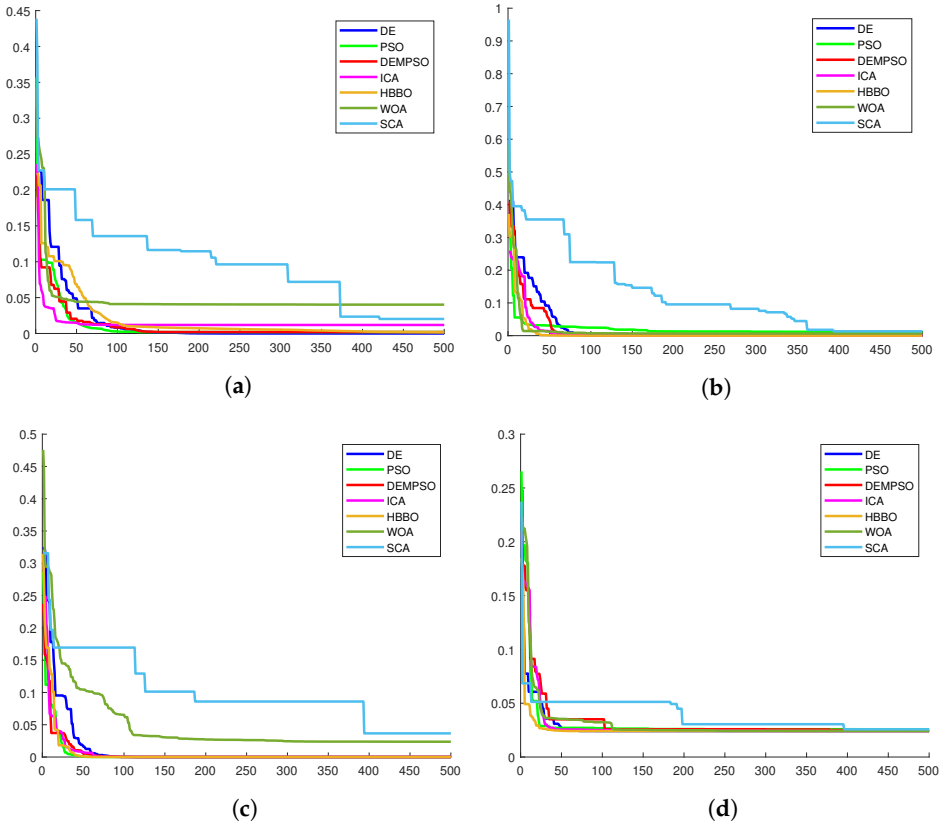

Figure 12. Convergence curves for the four robots. (**a**) Convergence curves for the Baxter robot); (**b**) Convergence curves for the Fanuc robot; (**c**) Convergence curves for the Puma robot; (**d**) Convergence curves for the Youbot robot.

6. Conclusions

In this work, we have presented a new approach to solve the path tracking task for robot manipulators using metaheuristic optimization. The proposed method does not have singularities problems like the Jacobian method. The proposed approach is a general method, that can be used to solve the path tracking task, with robots of any number of DOF. The experiments were performed with robots with 5 DOF, 6 DOF, and a redundant robot with 7 DOF. We have proposed a novel objective function that combines effectively the position, orientation, and joint angles objectives functions. We have shown how to normalize each of these objective functions in order to define a weight objective function. To the best of our knowledge, this is the first work that uses quaternions in the objective function to solve the end-effector orientation error with metaheuristic optimization algorithms. This orientation representation allowed us to effectively combine the position and joint objectives to solve the path tracking problem without singularities.

Author Contributions: Conceptualization, C.L.-F. and J.H.-B.; Data curation, D.D.; Formal analysis, C.L.-F. and N.A.-D.; Investigation, C.L.-F. and J.H.-B.; Methodology, M.L.-F.; Software, M.L.-F.; Validation, N.A.-D. and M.L.-F.; Visualization, D.D.; Writing—original draft, C.L.-F. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been supported by CONACYT México, through Project Cb-258068.

Data Availability Statement: The data analyzed are available from the authors upon request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bingul, Z.; Ertunc, H.; Oysu, C. Comparison of inverse kinematics solutions using neural network for 6R robot manipulator with offset. In Proceedings of the 2005 ICSC Congress on Computational Intelligence Methods and Applications, Istanbul, Turkey, 15–17 December 2005; p. 5. [CrossRef]
2. Balestrino, A.; De Maria, G.; Sciavicco, L. Robust Control of Robotic Manipulators. *IFAC Proc. Vol.* **1984**, *17*, 2435–2440. [CrossRef]
3. Wolovich, W.A.; Elliott, H. A computational technique for inverse kinematics. In Proceedings of the The 23rd IEEE Conference on Decision and Control, Las Vegas, NV, USA, 12–14 December 1984; pp. 1359–1363. [CrossRef]
4. Wampler, C.W. Manipulator Inverse Kinematic Solutions Based on Vector Formulations and Damped Least-Squares Methods. *IEEE Trans. Syst. Man Cybern.* **1986**, *16*, 93–101. [CrossRef]
5. Nakamura, Y.; Hanafusa, H. Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control. *J. Dyn. Syst. Meas. Control.* **1986**, *108*, 163–171. [CrossRef]
6. Buss, S.R.; Kim, J.S. Selectively Damped Least Squares for Inverse Kinematics. *J. Graph. Tools* **2005**, *10*, 37–49. [CrossRef]
7. Baillieul, J. Kinematic programming alternatives for redundant manipulators. In Proceedings of the Proceedings, 1985 IEEE International Conference on Robotics and Automation, St. Louis, MO, USA, 25–28 March 1985; Volume 2, pp. 722–728. [CrossRef]
8. Lee, C.; Ziegler, M. Geometric Approach in Solving Inverse Kinematics of PUMA Robots. *IEEE Trans. Aerosp. Electron. Syst.* **1984**, *AES-20*, 695–706. [CrossRef]
9. Yang, Y.; Peng, G.; Wang, Y.; Zhang, H. A New Solution for Inverse Kinematics of 7-DOF Manipulator Based on Genetic Algorithm. In Proceedings of the 2007 IEEE International Conference on Automation and Logistics, Jinan, China, 18–21 August 2007; pp. 1947–1951. [CrossRef]
10. Pham, D.T.; Castellani, M.; Fahmy, A.A. Learning the inverse kinematics of a robot manipulator using the Bees Algorithm. In Proceedings of the 2008 6th IEEE International Conference on Industrial Informatics, Daejeon, Korea, 13–16 July 2008; pp. 493–498. [CrossRef]
11. Huang, H.C.; Chen, C.P.; Wang, P.R. Particle swarm optimization for solving the inverse kinematics of 7-DOF robotic manipulators. In Proceedings of the 2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Seoul, Korea, 14–17 October 2012; pp. 3105–3110. [CrossRef]
12. Nearchou, A.C. Solving the inverse kinematics problem of redundant robots operating in complex environments via a modified genetic algorithm. *Mech. Mach. Theory* **1998**, *33*, 273–292. [CrossRef]
13. Mao, B.; Xie, Z.; Wang, Y.; Handroos, H.; Wu, H. A Hybrid Strategy of Differential Evolution and Modified Particle Swarm Optimization for Numerical Solution of a Parallel Manipulator. *Math. Probl. Eng.* **2018**, *2018*, 1–10. [CrossRef]
14. Ren, Z.W.; Wang, Z.H.; Sun, L.N. A hybrid biogeography-based optimization method for the inverse kinematics problem of an 8-DOF redundant humanoid manipulator. *Front. Inf. Technol. Electron. Eng.* **2015**, *16*, 607–616. [CrossRef]
15. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, 1st ed.; Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1989.
16. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948. [CrossRef]
17. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
18. Karaboga, D.; Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *J. Glob. Optim.* **2007**, *39*, 459–471. [CrossRef]
19. Bayati, M. Using cuckoo optimization algorithm and imperialist competitive algorithm to solve inverse kinematics problem for numerical control of robotic manipulators. *Proc. Inst. Mech. Eng. Part J. Syst. Control. Eng.* **2015**, *229*, 375–387. [CrossRef]
20. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [CrossRef]
21. Mirjalili, S. SCA: A Sine Cosine Algorithm for solving optimization problems. *Knowl.-Based Syst.* **2016**, *96*, 120–133. [CrossRef]
22. Tahyudin, I.; Haviluddin, H.; Nanb, H. Time Complexity Of A Priori Furthermore, Evolutionary Algorithm For Numerical Association Rule Mining Optimization. *Int. J. Sci. Technol. Res.* **2019**, *8*, 483–485.
23. Kuipers, J.B. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*; Princeton Univ. Press: Princeton, NJ, USA, 1999.
24. Dereli, S.; Köker, R. A Meta-Heuristic Proposal for Inverse Kinematics Solution of 7-DOF Serial Robotic Manipulator: Quantum Behaved Particle Swarm Algorithm. *Artif. Intell. Rev.* **2020**, *53*, 949–964. [CrossRef]
25. dereli, S. IW-PSO approach to the inverse kinematics problem solution of a 7-Dof serial robot manipulator. *Int. J. Nat. Eng. Sci.* **2018**, *36*, 75–85.
26. Çavdar, T.; Milani, M. A New Heuristic Approach for Inverse Kinematics of Robot Arms. *New Heuristic Approach Inverse Kinemat. Robot. Arms* **2012**, *19*, 329–333. [CrossRef]
27. Rokbani, N.; Casals, A.; Alimi, A.M. IK-FA, a New Heuristic Inverse Kinematics Solver Using Firefly Algorithm. In *Computational Intelligence Applications in Modeling and Control*; Azar, A.T., Vaidyanathan, S., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 369–395. [CrossRef]

28. Durmuş, B.; Temurtas, H.; Gün, A. An Inverse Kinematics Solution using Particle Swarm Optimization. In Proceedings of the International Advanced Technologies Symposium, Elazig, Turkey, 16–18 May 2011; pp. 16–18.
29. Rokbani, N. Inverse Kinematics Using Particle Swarm Optimization, A Statistical Analysis. *Procedia Eng.* **2013**, *64*, 1602–1611. [CrossRef]