Inverse Kinematics Optimization Using ACO, MOA, SPOA, and ALO: A Benchmark Study on Industrial Robot Arms

Aziz El Mrabet [1*], Hicham Hihi [2], Mohammed Khalil Laghraib [3], Mbarek Chahboun [4], Mohcine Abouyaakoub [5], Ali Ait Ali [6], Aymane Amalaoui [7]
[1, 2, 3, 4, 5, 6] Laboratory of Engineering, Systems and Applications, National School of Applied Sciences, Sidi Mohamed Ben Abdellah University, Fez, Morocco
[7] Laboratory of innovative technologies, National School of Applied Sciences, Sidi Mohamed Ben Abdellah University, Fez, Morocco
Email: [1] aziz.elmrabet@usmba.ac.ma, [2] hicham.hihi@usmba.ac.ma, [3] Mohammedkhalil.laghraib@gmail.com, [4] mbarek.chahboun@usmba.ac.ma, [5] mohcine.abouyaakoub@usmba.ac.ma, [6] ali.aitali@usmba.ac.ma, [7] aymane.amalaoui@usmba.ac.ma
*Corresponding Author

***Abstract*—This study investigates the application of metaheuristic algorithms to solve the inverse kinematics (IK) problem in robotic manipulators, which is often challenging for high-degree-of-freedom systems. The research contribution is the comparative evaluation of four recent metaheuristic algorithms—Ant Colony Optimization, Mayfly Optimization Algorithm, Stochastic Paint Optimizer, and Ant Lion Optimizer—across different robot configurations. A kinematic analysis was conducted on three robotic arms: a 4-DOF SCARA, a 6-DOF ABB IRB 1600, and the dual-arm 15-DOF Motoman SDA20D/12L. For each manipulator, the end-effector pose was optimized by solving the IK problem using the selected algorithms. A total of 30 random target positions were tested within the operational space to ensure diversity in pose challenges; while not exhaustive, this sampling provides statistically informative trends. We evaluate each algorithm based on the number of optimal solutions obtained,** the **precision of the computed joint configurations, and execution time. The results indicate that the Mayfly Optimization Algorithm consistently delivered the highest precision with relatively low execution time across all robot types**. In contrast, the **Ant Lion Optimizer showed inconsistent performance in higher-DOF settings. Unlike traditional Jacobian-based or analytical IK methods, metaheuristics offer flexibility in handling complex, nonlinear systems without requiring gradient information. These findings provide practical insight for selecting suitable algorithms in real-world robotic applications.**

***Keywords*—Robot Arm; Inverse Kinematics; Metaheuristic Algorithm; Optimization Techniques; Computational Complexity.**

I. INTRODUCTION

The resolution of the inverse kinematics (IK) issue is essential for the attainment of precise control within robotic systems. Inverse kinematics pertains to the process of ascertaining the requisite joint angles necessary to position a manipulator's end-effector at a designated spatial location and orientation. As advancements in robotics lead to the development of increasingly intricate and redundant manipulators, particularly those exhibiting degrees of freedom (DOF) that surpass six, the resolution of this issue presents heightened challenges. Conventional analytical methodologies, including the algebraic method [1], geometric techniques [2], [3], and iterative solutions, frequently prove inadequate [4]. This inadequacy is particularly pronounced in robots characterized by high DOF, where multiple joint configurations may yield identical end-effector positions. Such redundant configurations introduce complexities, including singularities and nonlinearities [5], [6], necessitating alternative computational methodologies. Practical applications of robotics introduce further complexities, encompassing joint limitations, sensor inaccuracies, physical impediments, and model discrepancies [7], [8], [9]. These uncertainties undermine the reliability of idealized mathematical frameworks, thereby constraining the efficacy of traditional IK solvers. Although calibration may alleviate certain challenges, it remains incapable of entirely obviating the requirement for more adaptive and flexible solution strategies [10], [11].

In response to these constraints, metaheuristic algorithms have emerged as viable alternatives for the resolution of IK challenges within complex, nonlinear, and high-dimensional search spaces[12], [13], [14] . These algorithms—such as Genetic Algorithms, Particle Swarm Optimization [15], and Ant Colony Optimization [16]—draw inspiration from natural phenomena and possess the capability to transcend local optima through dynamic mechanisms of exploration and exploitation. Nonetheless, these algorithms are accompanied by trade-offs, including a heightened sensitivity to parameter calibration, stochastic behavior, and potential limitations in real-time applications.

A multitude of studies have highlighted the potential of these techniques across diverse robotic platforms. For example, as illustrated in [17], the integration of Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) with Artificial Neural Networks (ANNs) for a 4-DOF SCARA robot resulted in a mean squared error (MSE) of 0.12846 with the PSO-ANN hybrid—surpassing GA-ANN in terms of accuracy and underscoring the potential of hybrid methodologies for real-time control applications. In the case of the 6-DOF ABB IRB 1600 robot, [18] implemented NSGA-II and the Bacterial Chemotaxis Multi-Objective

Algorithm (BCMOA), achieving highly precise end-effector poses with orientation errors as minimal as 5.4×10^{-8} and mean positional errors of merely 0.8 mm. Additionally, the Boomerang Algorithm, introduced by [19], exhibited promising real-time capabilities by achieving a precise IK solution within 8 seconds for the ABB IRB120 robot, with a mean positional error of 0.04 mm and a success rate of 95%—outperforming several variants of PSO. For more sophisticated robots, such as the Motoman SDA20D/12L, [20] proposed a Multi-Objective Full-Parameter Optimization PSO (MOFOPSO) methodology that concurrently optimized position, posture, and joint constraints. This algorithm attained a positional error of only 0.52 mm, demonstrating both precision and stability in high-DOF manipulators.

Despite these advancements, comparative evaluations of newer metaheuristic algorithms across diverse manipulator architectures remain scarce. This study endeavors to bridge this gap by assessing the performance of four recent and promising metaheuristic algorithms—Ant Colony Optimization (ACO), Mayfly Optimization Algorithm (MOA), Stochastic Paint Optimizer Algorithm (SPOA), and Ant Lion Optimizer (ALO)—on a representative array of robotic arms, encompassing both redundant and non-redundant manipulators.

The research contribution is a systematic comparative analysis of these four metaheuristics across three manipulators: a 4-DOF SCARA, a 6-DOF ABB IRB 1600, and a 15-DOF Motoman SDA20D/12L. Using 30 randomly generated end-effector targets per manipulator, the study assesses each algorithm's performance based on positional accuracy, convergence behavior, computational time, and solution reliability. The findings aim to guide robotics researchers and engineers in selecting appropriate optimization strategies for various IK scenarios—particularly in contexts requiring scalable, precise, and real-time solutions.

As illustrated in Fig. 1, this paper is structured as follows: Section 1 presents the kinematic analysis of the manipulator robot examined in the study. Section 2 delves into an overview of metaheuristic techniques and kinematic analysis. Section 3 then describes the optimization problems addressed. Subsequently, Section 4 comprehensively details the performance of the algorithms and reports the obtained results. Finally, Section 5 encapsulates the conclusions drawn from this research, offering insights into the implications of the findings for future work in the field of robotic kinematics.

II. KINEMATIC ANALYSIS OF MANIPULATOR ROBOTS

In the context of robotic manipulators, two fundamental kinematic problems arise: forward kinematics and inverse kinematics. Forward kinematics involves determining the end-effector's position and orientation relative to the base, using the joint angles of the robot manipulator. Conversely, inverse kinematics focuses on identifying the necessary joint angles to achieve a desired end-effector position.

Establishing the forward kinematic model of the manipulator is a crucial step in addressing the inverse kinematics challenge. The Denavit-Hartenberg convention is the most widely adopted standard for elucidating the kinematic relationships between the links and joints of a serial manipulator. Firmly rooted in the mechanical configuration of the manipulator, this standard provides a robust framework for analysis [21].

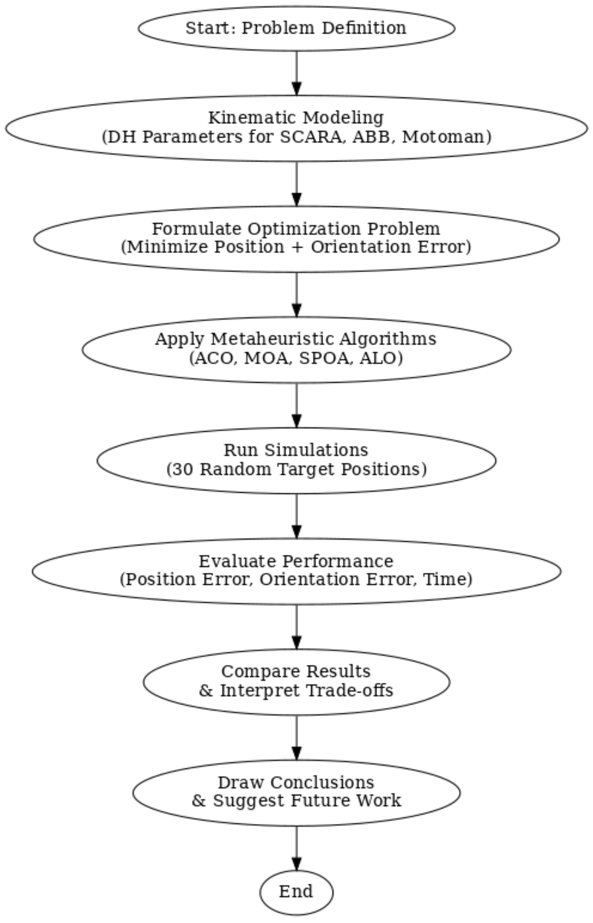

Start: Problem Definition
Kinematic Modeling (DH Parameters for SCARA, ABB, Motoman)
Formulate Optimization Problem (Minimize Position + Orientation Error)
Apply Metaheuristic Algorithms (ACO, MOA, SPOA, ALO)
Run Simulations (30 Random Target Positions)
Evaluate Performance (Position Error, Orientation Error, Time)
Compare Results & Interpret Trade-offs
Draw Conclusions & Suggest Future Work
End

Fig. 1. IK Problem-Solving Process for SCARA, ABB, and Motoman Manipulators

A. *Denavit-Hartenberg (DH)*

The Denavit-Hartenberg (DH) method is a widely adopted approach in robotics for modeling the kinematics of robots with i-degrees of freedom. This technique utilizes four key parameters: joint angle θ, joint distance d, link length a, and joint twist α, to characterize the geometric relationships between consecutive robot joints. These parameters are determined for each joint up to the i-th joint in the robot. The DH method provides a systematic framework for describing the kinematic structure of robotic systems.

- θ_i represents the joint angle, the angle between x_{i-1} and x_i.
- d_i denotes the translation along the z_{i-1} axis.
- a_i corresponds to the length of the i-th link.
- α_i signifies the joint twist, the angle between z_{i-1} and z_i.

The homogeneous transformation for each joint can be represented as the product of four basic transformations:

$$A_i = Rot_{z_i\theta_i} Trans_{z_i d_i} Trans_{x_i a_i} Rot_{x_i\alpha_i} \quad (1)$$

$$A_i = \begin{bmatrix} C_{\theta_i} & -S_{\theta_i}C_{\alpha_i} & S_{\theta_i}S_{\alpha_i} & a_iC_{\theta_i} \\ S_{\theta_i} & C_{\theta_i}C_{\alpha_i} & -C_{\theta_i}S_{\alpha_i} & a_iC_{\theta_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2)$$

B. 4-DOF SCARA manipulator

The 4-DOF SCARA manipulator is renowned for its simple yet highly effective design [22], making it a preferred choice for pick-and-place and assembly applications. The Denavit-Hartenberg parameters of the 4-DOF SCARA manipulator, accompanied by an illustrative GIF image, are referenced in Fig. 2 and Table I.

TABLE I. THE DH PARAMETERS FOR THE 4-DOF SCARA

Joint i	$\theta_i(deg)$	$d_i(mm)$	a_i	α_i(deg)
1	θ_1 *Between* ± 120	290	250	0
2	θ_2 *Between* ± 130	0	150	180
3	0	d_3 *Between* $0 - 150$	0	0
4	θ_4 *Between* ± 360	150	0	0

where $(\theta_1, \theta_2, d_3, \theta_4)$ are the joint variables. The key advantages of the 4-DOF SCARA manipulator include its inherent structural rigidity, high precision, and impressive performance capabilities, such as a repeatability of approximately ±0.01 mm and a maximum linear velocity. This SCARA manipulator was chosen for its suitability for applications requiring vertical pick-and-place operations and precise horizontal movement. Its kinematic construction facilitates swift and accurate placement, aligning with the objectives of the research endeavor.

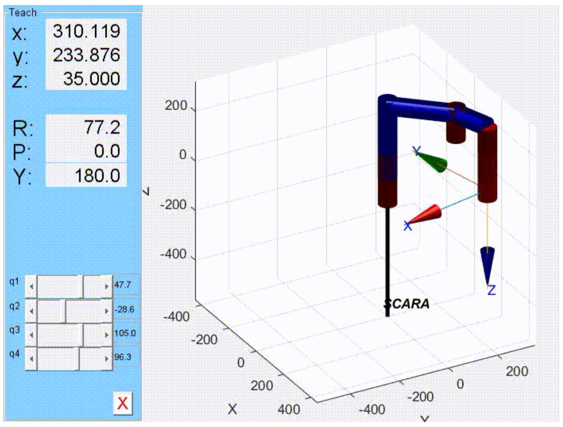

Teach
x: 310.119
y: 233.876
z: 35.000
R: 77.2
P: 0.0
Y: 180.0
q1 47.7
q2 -28.6
q3 105.0
q4 96.3

Fig. 2. The Simulation of 4-DOF SCARA manipulator in MATLAB

C. 6-DOF ABB IRB 1600 manipulator

The 6-degree-of-freedom (6-DOF) ABB IRB 1600 industrial robot, a prominent product from ABB Robotics, is distinguished by its exceptional accuracy and adaptability within industrial automation environments. Table II below outlines the Denavit-Hartenberg parameters, and a GIF image in Fig. 3 illustrates the configuration of this robotic manipulator.

In this configuration, the joint variables are $(\theta_1, \theta_2, d_3, \theta_4, \theta_5, \theta_6)$. The ABB IRB 1600 manipulator is a mainstay for activities requiring complex manipulation and multidirectional motions due to its increased degrees of freedom and robust construction. Its exceptional kinematic flexibility allows us to investigate and assess metaheuristic algorithms for addressing the intricate problems associated with robotic manipulation. This opens the door for a wide range of applications within the purview of our research, enabling us to explore innovative solutions and push the boundaries of what is possible in industrial automation and robotic systems.

TABLE II. DH PARAMETERS FOR THE 6-DOF ABB IRB 1600

Joint i	$\theta_i(deg)$	$d_i(mm)$	a_i	α_i(deg)
1	θ_1 Between ± 180	715	0	90
2	θ_2 Between $30 - 180$	0	350	0
3	0	d_3 Between $0 - 630$	0	-90
4	θ_4 Between ± 300	100	1200	90
5	θ_5 Between ± 120	0	0	-90
6	θ_6 Between ± 360	0	120	0

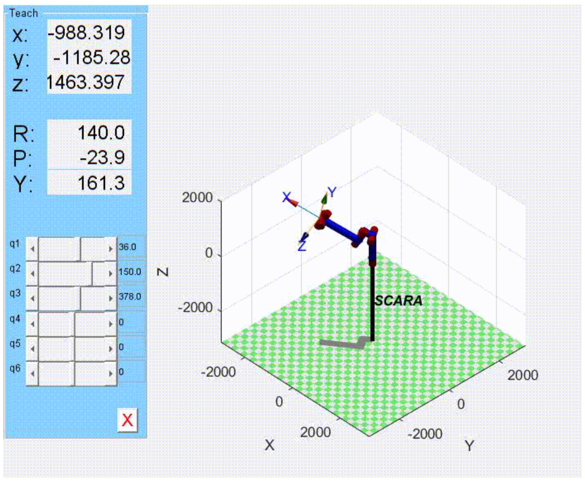

Teach
x: -988.319
y: -1185.28
z: 1463.397
R: 140.0
P: -23.9
Y: 161.3
q1 36.0
q2 150.0
q3 378.0
q4 0
q5 0
q6 0

Fig. 3. Simulation of 6-DOF ABB IRB 1600 in MATLAB

D. 6-DOF 12 link Motoman SDA20D

The Yaskawa Motoman SDA20D is a highly advanced and versatile robotic manipulator system that is renowned for its rapid operational speed and impressive payload capacity. Equipped with 6 degrees of freedom and 12 links, this manipulator offers exceptional versatility and performance capabilities, making it a popular choice for a wide range of applications. Table III shows the Denavit-Hartenberg settings for this 6-DOF, 12-link Motoman SDA20D manipulator, and Fig. 4 visual representation of the manipulator's configuration provided in the accompanying GIF image.

In this robot arm, the joint variables are $(d_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8, \theta_9, \theta_{10}, \theta_{11}, \theta_{12})$. The Motoman SDA20D robotic manipulator is distinguished by its exceptional reliability and operational capabilities, particularly when executing tasks requiring precise trajectory control and managing substantial payloads. Owing to its expansive range of motion, this manipulator enables the execution of complex movements critical to the research study, thereby ensuring optimal performance and accuracy throughout the experimental process.

TABLE III. DH PARAMETERS FOR THE 6-DOF 12 LINKS MOTOMAN SDA20D

Joint i	$\theta_i(deg)$	$d_i(mm)$	a_i	α_i(deg)
1	0	d_1 Between 0 − 360	0	0
2	θ_2 Between ±180	0	170	−90
3	θ_3 Between ±135	0	835	0
4	θ_4 Between ±180	1160	160	−90
5	θ_5 Between ±180	200	0	90
6	θ_6 Between ±360	1250	0	−90
7	θ_7 Between ±180	200	0	90
8	θ_8 Between ±360	1250	0	−90
9	θ_9 Between ±180	200	0	90
10	θ_{10} Between ±180	1160	160	−90
11	θ_{11} Between ±135	0	835	90
12	θ_{12} Between ±180	0	170	−90

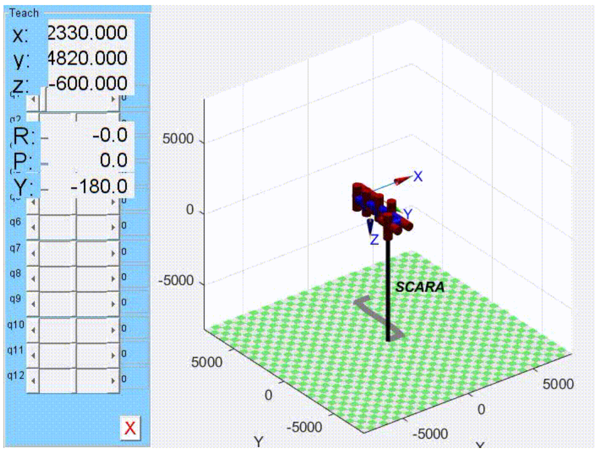


Fig. 4. Simulation of 6-DOF 12 links Motoman SDA20D in MATLAB

III. METAHEURISTIC OPTIMIZATION ALGORITHMS

In recent decades, metaheuristic optimization algorithms have emerged as highly effective tools for addressing optimization problems [23], [24], [25]. Drawing inspiration from natural phenomena, these algorithms explore the solution space in search of optimal outcomes. A diverse array of techniques, including genetic algorithms [26], artificial bee colony algorithms [27], particle swarm optimization [28], [29], stochastic paint optimization, and ant lion optimization, are among those inspired by nature [30], [31], [32].

In contrast to traditional analytical methods heavily reliant on mathematical deductions, metaheuristic algorithms offer a more adaptable and robust approach. Characterized by the dual attributes of diversification and intensification, these algorithms facilitate efficient exploration of the search space through the use of randomness, while also conducting local searches around the current best solution.

The resilience of metaheuristic algorithms in addressing a wide range of optimization challenges, their capacity to handle complex and lengthy problems, their ability to identify global solutions, and their rapid convergence rates have propelled them to the forefront of various domains. Consequently, they have become indispensable tools, offering significant advantages over conventional optimization methodologies.

A. *Particle Ant Colony Optimization Algorithm (ACO)*

Ant Colony Optimization (ACO) is a metaheuristic algorithm inspired by the foraging behavior of ant colonies. It is a combinatorial optimization method that assembles candidate solutions from a set of components that compete for attention, avoiding the need for extensive fine-tuning. ACO is colony-oriented, consisting of two distinct "colonies": the set of components that make up the candidate solution and the set of trials that constitute the candidate solution [33], [34]. The set of components is fixed, while the fitness of each component is updated over time. Each generation of the algorithm constructs one or more ant trails based on the pheromone selection components, and the fitness of each trail is then evaluated. The pheromone of each component is subsequently updated based on the fitness of the trail.

As we can see in Fig. 5, the ACO algorithm can be viewed as population-oriented, comprising two different "populations": the set of components that make up possible solutions and the set of trials that represent possible solutions [35]. The number of components is fixed, while the fitness of each component is updated over time. Each generation of the algorithm builds one or more ant trails based on the selected pheromones, and the fitness of each trail is evaluated. The pheromone of each component is then updated according to the fitness of the trail.

```
Output → Best

1:  begin
2:  $C \leftarrow \{C_1, \dots, C_n\}$ components
3:  popsize ← numbre of trails to build at once
4:  $\vec{p} \leftarrow < p_1, \dots, p_n >$ pheromones of the components, initially zero
5:  Best ← ■
6:  While Best is the ideal solution or we have run out of time
7:        P ← popsize trails built by iteratively selecting components basd
          on pheromones and costs or values
8:        For $P_i \in P$ do
9:              $P_i \leftarrow$ Optionally Hill − climb $P_i$
10:                   Adjust male positions by (3)
11:             if Best ← ■ or Fitness($P_i$) > Fitness(best) then
12:                   Best ← $P_i$
13:             end if
14:       Update $\vec{p}$ for components based on the fitness results for each $P_i \in P$
          in which they participated
15:       end for
16:  end while
17: end
```

Fig. 5. Pseudocode for the ACO algorithm

At each iteration t, an ant k builds a complete solution by selecting values for each joint variable. The probability of selecting a specific value j for joint i is:

$$P_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}(t)]^{\beta}}{\sum_{l \in \Omega_i} [\tau_{il}(t)]^{\alpha} \cdot [\eta_{il}(t)]^{\beta}} \tag{3}$$

Where, $\tau_{ij}(t)$is the pheromone value for variable j of joint i. $\eta_{ij}(t)$ is the heuristic value (e.g., inverse of the estimated error). α and β control the influence of pheromone vs. heuristic. Ω_i is the set of available discrete values for joint i.

After evaluating all ants, pheromone levels are updated to reinforce high-quality solutions:

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \sum_{k=1}^{m}\Delta\tau_{ij}^{k} \tag{4}$$

with:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{f(\Theta^k)}, & if\ \Theta^k\ uses\ \theta_{ij} \\ 0, & otherwise \end{cases} \tag{5}$$

Where, ρ is the evaporation rate. Q is a constant pheromone deposit factor. m is the number of ants.

This rule ensures that components of lower-error solutions gain higher pheromone reinforcement, improving their selection probability in the next iteration. Table IV shows the parameter settings used in this paper.

TABLE IV. TYPICAL PARAMETER SETTINGS USED FOR ACO

Parameter	Typical Range	Purpose
m	30–50	Number of ants per iteration
α	1–2	Influence of pheromone trails
β	2–5	Influence of heuristic information
ρ	0.1–0.5	Evaporation rate
Q	1	Pheromone deposit constant
T	300–500	Total number of iterations

In the context of inverse kinematics (IK), ACO is used to iteratively generate candidate joint angle configurations that minimize a composite error function comprising position and orientation discrepancies between the desired and actual end-effector poses. Each solution, corresponding to a complete set of joint variables for a robot manipulator, is treated as a "path" constructed by an artificial ant. These ants simulate the pheromone-guided movement found in nature, with high-performing solutions reinforcing the desirability of their components.

B. Mayfly Optimization Algorithm (MA)

The Mayfly Optimization Algorithm (MOA) is a metaheuristic approach that significantly improves upon the particle swarm optimization (PSO) [36], [37], genetic algorithm (GA) [38], and the firefly algorithm (FA) to create a powerful hybrid framework [39]. This innovative approach leverages the social behavior of mayflies and incorporates crossover techniques and local search mechanisms. Inspired by the life cycle of mayflies, where maturity triggers reproductive behavior, MOA aims to ensure the survival of the fittest and reflects the ability of the strongest mayflies to reproduce. Within the context of the algorithm, the position of each mayfly in the search space represents a potential solution to the optimization problem.

As shown in Fig. 6, the MOA algorithm operates as follows: two distinct groups of mayflies, representing males and females, are randomly initialized within the problem space. These mayflies embody possible solutions, which are encoded as d-dimensional vectors x and evaluated according to a predefined objective function f. The velocity vector v reflects the adjustments in the mayflies' positions. The trajectory of each mayfly encapsulates the dynamic interplay between individual experience and social influences. Specifically, at each iteration, the mayfly adjusts its path to reach its personal best position based on the collective experience of the group, as described in the research by [40].

```
Output → Optimal Solution (gbest)

1:  begin
2:  Evaluate all solutions according to the objective function $f(x)$
3:  Find the best value from all solutions (gbest)
4:  While $t < T$
5:      find pbest by (2), the best solution of each male mayfly
6:      For each $i = 1$ to $M$ do
7:          For each $j = 1$ to $d$ do
8:              Adjust male velocity by (2)
9:              Adjust male positions by (3)
10:         end for
11:     end For
12:     For each $i = 1$ to $M$ do
13:         For each $j = 1$ to $d$ do
14:             Adjust female velocity by (4)
15:             Adjust female positions by (3)
16:         end for
17:         Update pbest
18:     end For
19:     Rank male mayflies
20:     Rank female mayflies
21:     Mate the mayflies
22:     Evaluate the offspring
23:     Separate offspring to male and female randomly
24:     Replace the worst solution with the best new one Update pbest and gbest
25: end while
26: end
```

Fig. 6. Pseudocode for the MOA algorithm

In the field of entomology, as shown in Fig. 2, mayflies exhibit a distinctive life cycle: they spend several years as aquatic larvae before emerging as short-lived adult forms that fly in the air for a period of only one to seven days. The need to study the mating behaviors of male and female mayflies inspired the development of the Mayfly Optimization (MO) algorithm. Building upon the Particle Swarm Optimization (PSO) framework, the MO algorithm simulates the position updates of mayflies:

$$p_i(t+1) = p_i(t) + v_i(t) \tag{6}$$

The position p_i represents the position for the i −th individual in the next iteration. Accounting for the distinct behaviors of male and female mayflies, their velocities are updated using different approaches.

For male mayflies, which exhibit remarkable speeds as they soar above water surfaces, the velocity calculations are distinctive:

$$\begin{aligned} v_i(t+1) = v_i(t) + a_1 e^{-\beta r_p}\left(pbest_i - x_i(t)\right) \\ + \alpha_2 e^{-\beta r_g^2}\left(gbest_i - x_i(t)\right) \end{aligned} \tag{7}$$

$$x_i(t+1) = x_i(t) + v_i(t) \tag{8}$$

In this context, v_i represents the velocity of the i-th mayfly at the current time step, while x_i denotes the position of that mayfly. The parameter β is a fixed visibility coefficient that limits the range over which a mayfly can perceive its environment. Additionally, $pbest_i$ indicates the optimal position previously visited by the i −th mayfly, and $gbest_i$ refers to the position of the best male mayfly component. The positive attraction constants α_1 and α_2 capture the cognitive and social influences, respectively, that guide the mayflies' movements.

Regarding the female mayflies, they are attracted to their male counterparts and follow deterministic attraction models that dictate the changes to their location and velocity.

$$v_i(t+1) = \begin{cases} g.v(t) + a_f e^{-\beta r_{mf}^2}\left(x_i(t) - y_i(t)\right) \;\; ; \;\; f\left(y_i(t)\right) > f\left(x_i(t)\right) \\ g.v(t) + fl.r_1 \;\; ; \;\; f\left(y_i(t)\right) \le f\left(x_i(t)\right) \end{cases} \tag{9}$$

Where α_f and β are fixed parameters, g, and fl represent weights that decrease from their maximum to lowest values and $f(x_i(t))$ and $f(y_i(t))$ denote the fitness values. A random integer sampled from the uniform distribution from -1 to 1 is represented by r_1.

When mayflies engage in mating, a selection process akin to the male-female attraction dynamics occurs, whereby the superior male mayflies mate with the better female mayflies to produce offspring:

$$\begin{cases} off_1 = \alpha_3 male + (1-\alpha_3) female \\ off_2 = (1-\alpha_3) male + \alpha_3 female \end{cases} \tag{10}$$

Here the male is the parent male mayfly, the female is the female parent, and α_3 represents a random number adhering to a Gaussian distribution.

In this study, MOA is applied to solve the inverse kinematics (IK) problem by minimizing a composite objective function that accounts for both position and orientation errors of the end-effector. The algorithm’s ability to navigate high-dimensional, nonlinear, and multimodal search spaces makes it particularly effective for robotic systems with redundancy or singularities. The specific parameter settings used for MOA in this work are summarized in Table V.

TABLE V. TYPICAL PARAMETER SETTINGS USED FOR MOA

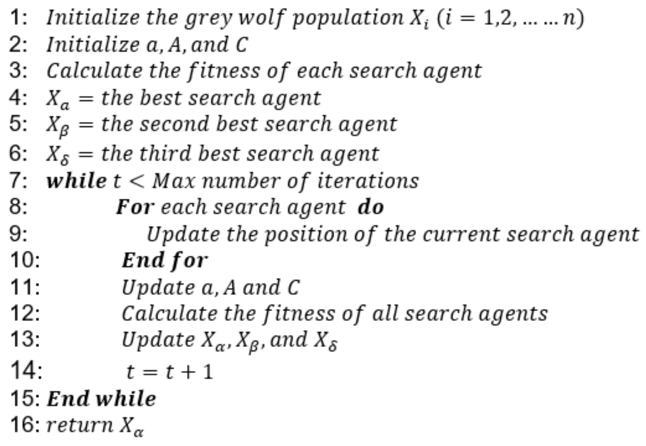
Parameter	Range	Description
Population size	30–50	Number of male + female mayflies
a_1, a_2	1.0–2.0	Cognitive and social coefficients
β	1.0–2.0	Visibility coefficient
g , fl	0.9–0.1	Female damping and random weights
α_3	0.2–0.8	Crossover factor
T	300–500	Max number of iterations

C. Stochastic Paint Optimizer Algorithm (SPO)

In this section, a new meta-heuristic technique called the Stochastic Paint Optimizer (SPO) is presented, which utilizes the rich semantic associations of colors in painting. The SPO algorithm is inspired by the complex processes involved in selecting and combining colors, akin to an artist's approach to creating an artwork.

Through a series of meticulously designed procedural steps, the SPO algorithm optimizes color compositions on a virtual canvas. These procedures encompass the creation of initial paint configurations, paint combinations, paint clustering, and termination criteria. The canvas serves as the search space in the SPO algorithm, where paints are represented as solutions with their constituent colors acting as design variables. Evaluating the visual appeal of paint is part of the assessment process, facilitated by a "beauty index" derived from objective function values. The distinct contributions of each color on the canvas to the overall perception of the artwork necessitate a sophisticated grading system based on the primary, secondary, and tertiary color categories on the color wheel. Leveraging established color combination techniques, the SPO algorithm orchestrates the synthesis of novel colors to craft optimal paint configurations [41], [42].

The algorithmic implementation of the SPO technique involves iterative phases of color mixing, color clustering, and fitness evaluation [43], [44]. In each iteration, the algorithm explores the paint search space by introducing controlled stochastic perturbations to the current paint configuration, akin to an artist's experimental approach, as represented in Fig. 7.



```
1: Initialize the grey wolf population $X_i$ $(i = 1,2, \ldots \ldots n)$
2: Initialize $a, A,$ and $C$
3: Calculate the fitness of each search agent
4: $X_a$ = the best search agent
5: $X_\beta$ = the second best search agent
6: $X_\delta$ = the third best search agent
7: while $t <$ Max number of iterations
8:         For each search agent do
9:             Update the position of the current search agent
10:        End for
11:        Update $a, A$ and $C$
12:        Calculate the fitness of all search agents
13:        Update $X_\alpha, X_\beta,$ and $X_\delta$
14:        $t = t + 1$
15: End while
16: return $X_\alpha$
```

Fig. 7. Pseudocode for the SPO algorithm

The effectiveness of the SPO algorithm stems from its seamless integration of color theory principles with optimization methodologies, leading to the generation of visually appealing and harmonious artworks. By leveraging the inherent stochasticity in color selection and combination processes, the SPO algorithm transcends conventional optimization paradigms, providing a novel perspective on artistic expression and creativity in the digital realm.

In this context, each candidate solution (analogous to a "paint") is represented as a vector of real-valued decision variables:

$$X_i = [x_{i1}, x_{i2}, \ldots, x_{id}], X_i \in \mathbb{R}^d \tag{11}$$

where d is the number of joint parameters for the manipulator.

The algorithm proceeds through iterative refinement of a solution population $P = \{X_1, X_2, \ldots, X_N\}$, using a stochastic update rule designed to perturb and explore new regions of the solution space. In each iteration, solutions are modified using controlled noise and clustering heuristics as follows:

$$X_i^{t+1} = X_i^t + \epsilon \cdot N(0, \sigma^2) \tag{12}$$

Where, ϵ is a learning or perturbation rate. $N(0, \sigma^2)$ is Gaussian noise with variance σ^2.

A clustering-based selection mechanism is applied after each generation to retain the top k individuals with minimum error, simulating the refinement of "visual harmony" in the solution population. The population is then replenished by generating new candidates via interpolation of top solutions and perturbation:

$$X_{new} = \lambda_1 X_\alpha + \lambda_2 X_\beta + \lambda_3 X_\delta + \delta \tag{13}$$

Where, $X_\alpha, X_\beta, X_\delta$ are three high-fitness solutions. $\lambda_1 + \lambda_2 + \lambda_3 = 1$, with $\lambda_i \in [0,1]$ being the **blending coefficients**. δ is a small random vector (mutation).

The algorithm terminates after a fixed number of iterations t or when the best solution achieves an error below a threshold ϵ_{min}.Table VI summarizes the specific parameter settings used for SPO in this work.

TABLE VI. TYPICAL PARAMETER SETTINGS USED FOR SPO

Parameter	Range	Description
Population size	30–50	Number of candidate solutions in each generation
Top-k selection	10–30% of N	Percentage of elite individuals retained for interpolation
$\lambda_1, \lambda_2, \lambda_3$	Random or fixed (sum = 1)	Weights used in generating new candidates via interpolation
T	300–500	Max number of iterations

D. Ant Lion Optimizer Algorithm (ALO)

Ant Lion Optimizer Algorithm is a powerful population-based Metaheuristic Algorithm that meticulously emulates the sophisticated hunting strategies of antlions in their natural habitats [45]. Named after their remarkable hunting prowess and favored prey, antlions employ a distinctive tactic of excavating cone-shaped pits in sand using circular motions executed by their formidable jaws. These patient predators then position themselves at the pit's base, awaiting unsuspecting prey to stumble into their snare. Upon detecting an ensnared insect unable to escape the steep pit walls, antlions swiftly dispatch their quarry. In nature, as insects instinctively strive to evade traps, they often engage in frantic movements to flee from predators. In response, astute antlions deftly fling sand just ahead of the prey's path, inducing them to slide inexorably into the pit's depths. It meets its demise when the prey's evasive maneuvers fail to outpace the encroaching jaws. Interestingly, antlions also adjust the dimensions of their pits based on factors such as their degree of hunger and the lunar phase [46].

As shown in Fig. 8, the algorithm employs two distinct types of search agents: ants and antlions. Antlions, the superior agents, maintain fixed positions unless replacing a specific ant. In contrast, ants perform random walks within the search space and risk being captured by antlions if they fall into the antlions' traps [47].

The algorithmic formulation initializes a population of solutions as ants navigating the search landscape. Each ant's position is represented as a vector:

$$Ant_i = \left[A_i^1, A_i^2, A_i^3, \dots, A_i^d\right] \tag{14}$$

where Ant_i represents the $i-$th ant's, and indicates its coordinates in the d-th dimension. The locations of the ants are determined by the following rule (15):

$$Ant_i^t = \frac{R_A^t + R_E^t}{2} \tag{15}$$

Where, R_A^t denotes a random walk performed by an ant near the antlion selected through the roulette wheel mechanism at the $t-$th iteration. R_E^t represents the location of a randomly walking ant, denoted as Ant_i, near the best-performing antlion, known as the elite antlion, within the ant swarm.

```
Output → The elitist antlion and the corresponding fitness value

1:  begin
2:  Initialize the random positions of all agents $x = (x_1, x_2, x_3, \dots, x_n)$
      inside ub and lb bounds.
3:  Calculate the fitness of population
4:  Select the Elite antlion E
5:  While end condition is not met
6:            For each Anti do
7:             Select an antlion A based on roulette wheel
8:            Run random walk of Anti nearby Antlion A; $R_A^t$
9:            Run random walk of Anti nearby Antlion E; $R_E^t$
10:           Update the location of Anti
11:       end for
12:      Update the fitness values of all agents
13:      Merge all ants and sort them based on the fitness
              metric and select the fittest n agents
14:      As the new antlions and the worst n agents as the ants
15:      Update the elite ant if an antlion is better than the elite agent
16: end while
17: end
```

Fig. 8. Pseudocode for the ALO algorithm

The roulette wheel mechanism utilizes the fitness values of the antlions to select an antlion A for an ant to perform a random walk nearby, while the elite antlion, identified as the best-performing antlion, is designated as E.

The random walking behavior of an ant Ant_i^t in proximity to a presumed antlion $Antlion_j^t$ is expressed as follows:

$$R_j^t = \frac{(X_i - a_i) \times (d_i - c_i^t)}{b_i^t - a_i} + c_i \tag{16}$$

Where, R_j^t represents the ant's position after performing a random walk near the selected antlion j during iteration t. a_i denotes the minimum step size of the random walk X_i^t in the i-th dimension. b_i represents the maximum step size of the random walk X_i^t in the i-th dimension. c and d define the lower and upper bounds of the random walk, respectively.

The position of each ant in each dimension is updated using a random walk process as follows:

$$x(t) = \left[0, cum\left(2r(t_1)\right) - 1, cum\left(2r(t_2)\right) - 1, \dots, cum\left(2r(t_T)\right) - 1\right] \tag{17}$$

Where, T represents the total number of iterations. t_i denotes the index of the current iteration. cum indicates the cumulative summation operation. r is a random function that is calculated as follows:

$$r(t) = \begin{cases} 1\ ; r > 0.5 \\ 0\ ; r \le 0.5 \end{cases} \tag{18}$$

ALO is effective for multi-DOF manipulators in high-dimensional, non-convex spaces such as IK. Its dual-random-walk approach enhances global exploration while maintaining elite-guided convergence. However, due to its stochastic nature and high iteration count, ALO can be computationally expensive and less suitable for real-time IK control. The specific parameter settings used for ALO in this work are summarized in Table VII.

TABLE VII. TYPICAL PARAMETER SETTINGS USED FOR ALO

Parameter	Range	Description
Population size	30–50	Number of ants and antlions
Top-k selection	10–30% of N	Percentage of elite individuals retained for interpolation
Scaling factor	Dynamic	Bounds shrink with each iteration
T	300–500	Max number of iterations

IV. OPTIMIZATION PROBLEM

This section presents a generalized objective function for solving inverse kinematics (IK) problems using metaheuristic optimization techniques. The proposed formulation provides a unified framework that accommodates the multimodal and nonlinear characteristics inherent in robotic manipulators, thus enabling the application of diverse algorithmic strategies [48].

Utilizing optimization methodologies to address the inverse kinematics challenge, we can quantify the error E as a measure of the discrepancy between the current position of the end effector and its intended position relative to the specified objective. The orientation error, E_O, delineates the variance in the desired orientation of the end effector, facilitating the attainment of precise accuracy. Meanwhile, the position error, E_p, highlights disparities in the coordinates of the end effector relative to its designated location.

Consequently, the following equation encapsulates the composite error E, which is contingent upon both orientation and position errors. This composite error is calculated as a weighted sum of the position error, E_p, and the orientation error, E_O, where the weighting coefficients α and β determine the relative importance of each error component.

$$E = \alpha E_p + \beta E_O \tag{19}$$

The prioritization of position and orientation errors can be achieved by employing constant weighting coefficients, denoted as α and β, respectively. The resulting formulation of the objective function aimed at resolving the inverse kinematics problem is represented by the equation below:

$$\min_{\theta\in[\theta_{min},\theta_{max}]} E \tag{20}$$

A. Position Based Error

The Euclidean distance metric, which represents the distance between the end-effector's current position and desired position, is used to quantify the positional discrepancy of the end-effector. This can be expressed mathematically as follows [49]:

$$E_p = ||P_d - P_c|| \tag{21}$$

B. Orientation Based Error

To measure the orientation error between the required and current frames, a comparative analysis of their orientations can offer valuable insights. Although the XYZ and UVW frames have distinct orientations, they share a common origin. As a result, aligning the current XYZ frame with the target UVW frame necessitates a rotational adjustment, represented by R_E. This rotational alignment of the current frame towards the target frame serves as a quantifiable metric for orientation discrepancies.

$$R_E = R_d R_c^{-1} = R_d R_c^T = \begin{pmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{pmatrix} \tag{22}$$

The orientation error can be mathematically quantified by extracting the variables n_x, o_y, and a_z from the rotation matrix R_E using the following equation, as described in the work by Kumar et al. [50].

$$E_o = 2cos^{-1}\left(\frac{1}{2}\sqrt{n_x + o_y + a_z + 1}\right) \tag{23}$$

V. RESULTS AND DISCUSSION

In this study, we utilized three distinct robotic manipulators to conduct our experiments: a 4-degree-of-freedom SCARA manipulator, a 6-DOF ABB IRB 1600 manipulator, and a 6-DOF 12-link Motoman SDA20D manipulator. These manipulators were evaluated at thirty randomly selected locations within their respective workspaces.

To assess the performance of the Ant Lion Optimizer algorithm, Stochastic Paint Optimizer Algorithm, Mayfly Optimization Algorithm, and Ant Colony Optimization, we employed various metrics, including position error. This error was quantified by calculating the Euclidean distance between the algorithm's solution location and the actual position. This metric provided valuable insights into the accuracy and effectiveness of each algorithm in solving the inverse kinematics problem for the robotic manipulators under investigation:

$$E_{rr} = ||P_s - P_r|| \tag{24}$$

Where E_{rr} is the position error.

Additionally, the assessment of orientation errors involved comparing the pitch (rotation by θ_s about the fixed y-axis), yaw (rotation by ψ_s about the fixed x-axis), and roll (rotation by ϕ_s about the fixed z-axis) angles of the algorithm's solution with the corresponding actual angles (θ_r, ψ_r,ϕ_r):

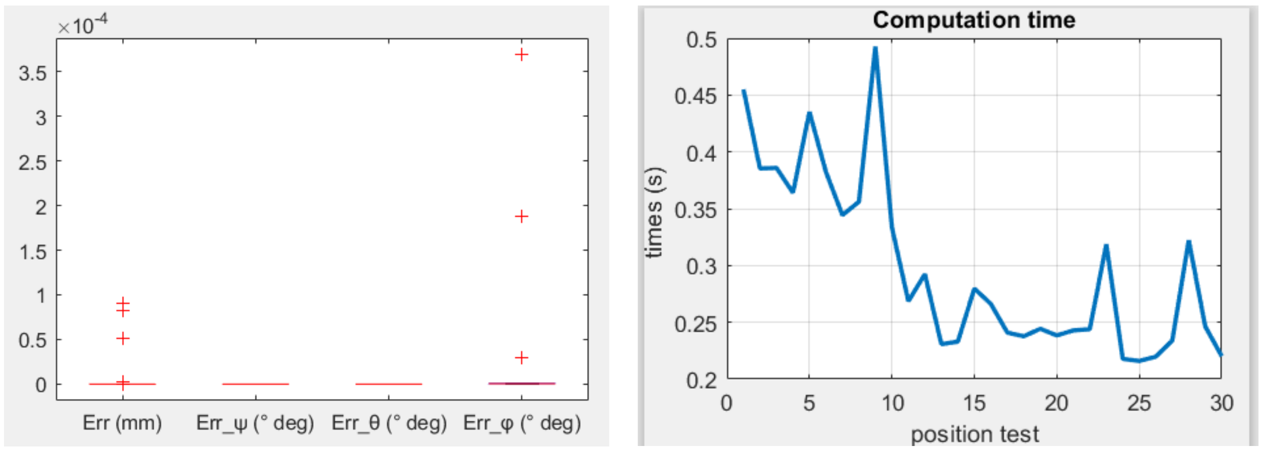
$$\begin{aligned} E_{rr_\psi} &= |\psi_r - \psi_s|; \\ E_{rr_\theta} &= |\theta_r - \theta_s|; \\ E_{rr_\phi} &= |\phi_r - \phi_s| \end{aligned} \tag{25}$$

Where, E_{rr_ψ} is the orientation error of the yaw. E_{rr_θ} is the orientation error of the pitch. $E_{rr_\phi}$is the orientation error of the roll.

A. 4-DOF SCARA Manipulator

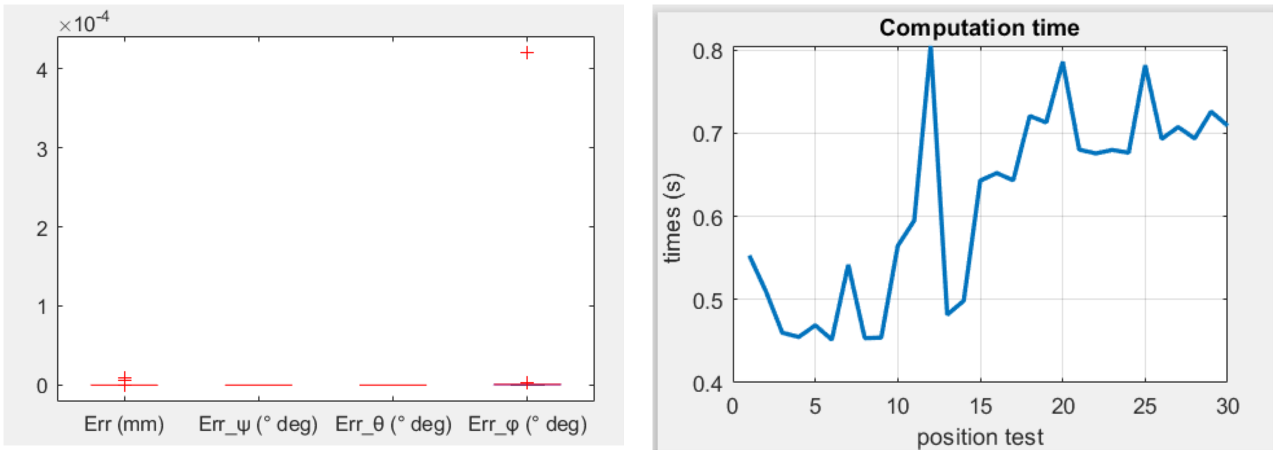
This section thoroughly assesses the performance of diverse metaheuristic algorithms when applied to the 4-DOF SCARA manipulator. The evaluation examines position and orientation errors and the time required for the algorithms to converge to the optimal solution. The findings are presented graphically in Fig. 9 through Fig. 12.

- Ant Colony Optimization: As shown in Fig. 9, this algorithm achieved outstanding results with minimal errors. The maximum position error was 8.99e-05 mm, the minimum was 0 mm, and the mean error was 7.55e-06 mm. The orientation errors were consistently zero across all axes. Additionally, the algorithm demonstrated efficient computation times, with a maximum of 2.4634 seconds, a minimum of 1.0797 seconds, and a mean of 1.4913 seconds.
- Mayfly Optimization Algorithm: Similar to ACO, as depicted in Fig. 10, the Mayfly Optimization Algorithm exhibited excellent error reduction. The position error ranged from 9.17e-06 mm to 0 mm, with a mean error of 4.8115e-07 mm. Orientation errors were also consistently zero. The computation times were competitive, with a maximum of 4.0249 seconds, a minimum of 2.2581 seconds, and a mean of 3.0788 seconds.
- Stochastic Paint Optimizer Algorithm: SPO demonstrated in Fig. 11 competitive performance in minimizing position errors, with a mean error of 3.1412e-05 mm, comparable to other algorithms. Its orientation errors were also negligible. The moderate computational time of 1.6283 seconds suggests efficient optimization for precise robotic arm configurations.
- Ant Lion Optimizer: As shown in Fig. 12, ALO demonstrated good performance in minimizing position errors, with a mean error of 2.12e-05 mm, comparable to other algorithms. However, it exhibited higher orientation errors, especially in the ϕ angle. The computational time of 3.8612 seconds suggests moderate efficiency in solving the inverse kinematic problem for this robotic arm.



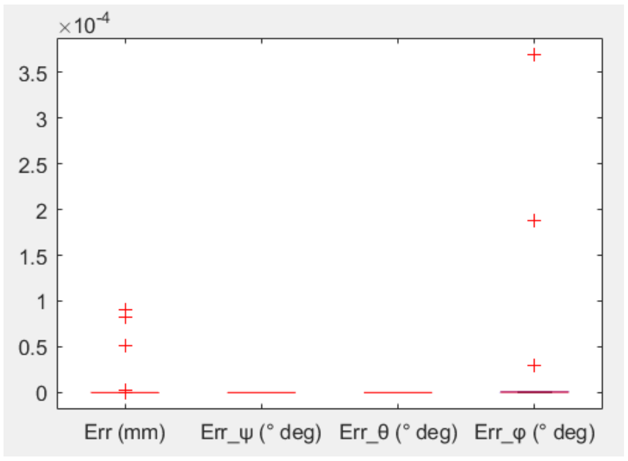
(a) (b)

Fig. 9. Results of Ant Colony Optimization, (a) position and orientation errors, and (b) Computation time

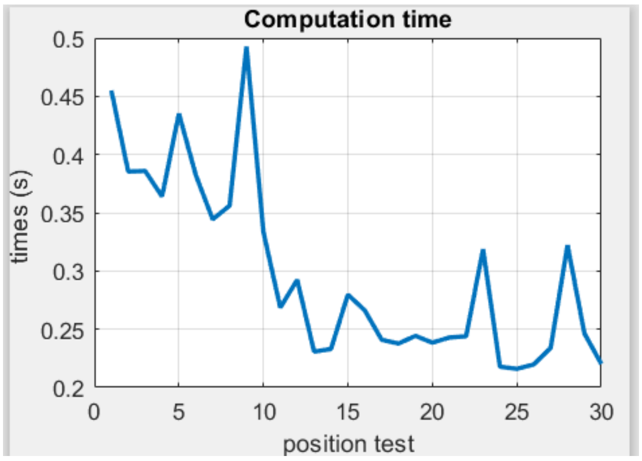


(a) (b)

Fig. 10. Results of Mayfly Optimization Algorithm, (a) position and orientation errors and (b) Computation time

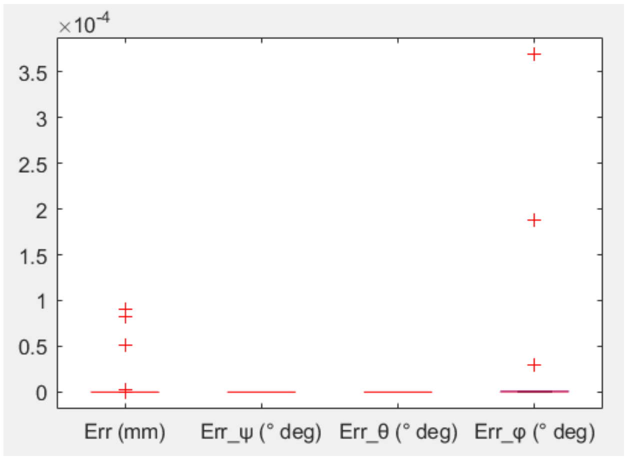


(a)

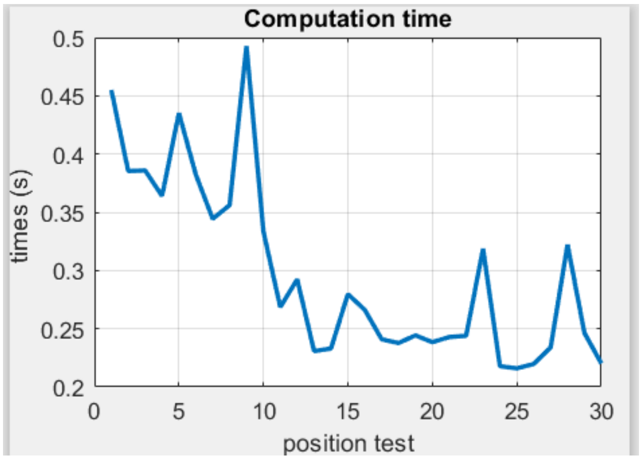


(b)

Fig. 11. Results of Stochastic Paint Optimizer Algorithm, (a) position and orientation errors, and (b) Computation time



(a)



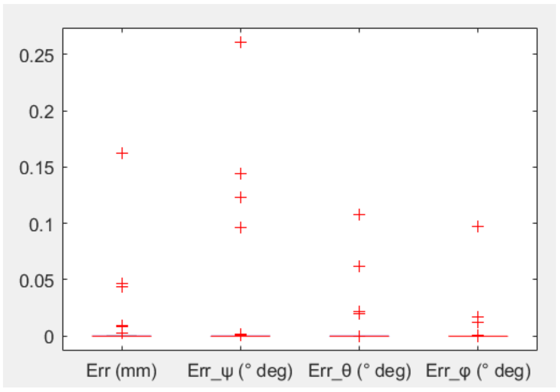
(b)

Fig. 12. Results of Ant Lion Optimizer algorithm, (a) position and orientation errors, and (b) Computation time

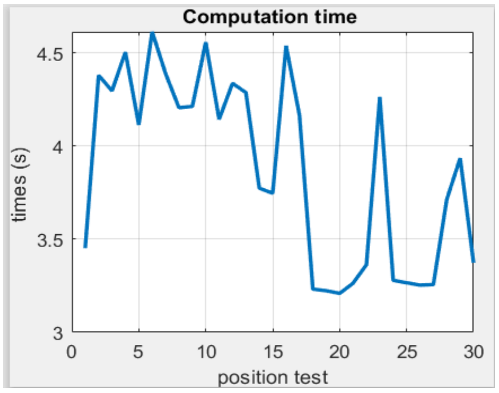
B. *6-DOF ABB IRB 1600 Manipulator*

This section examines the effectiveness of various metaheuristic algorithms when applied to the 6-degree-of-freedom ABB IRB 1600 manipulator. Fig. 13 to Fig. 16 provide graphical representations of the position and orientation errors, as well as the convergence times, for each algorithm tested on this robotic platform.

- Ant Colony Optimization: As depicted in Fig. 13, ACO demonstrated good performance in minimizing position errors, with a mean error of 0.0091 mm, which is acceptable for most applications. However, as shown in Fig. 13, it exhibited slightly higher orientation errors compared to other algorithms, particularly in the ϕ angle. Additionally, the computational time of 3.8788 seconds indicates moderate efficiency in solving the inverse kinematic problem for this robotic arm.
- Mayfly Optimization Algorithm: As observed in Fig. 14, MO exhibited exceptional performance in minimizing errors across all metrics for the ABB IRB 1600 manipulator. Its mean position error of 0.0091 mm and orientation errors were significantly lower compared to other algorithms. However, as shown in Fig. 14, its computational time of 3.8788 seconds suggests the need for optimization to improve efficiency.
- Stochastic Paint Optimizer Algorithm: As illustrated in Fig. 15, SPO showed good performance in minimizing errors for the ABB IRB 1600 manipulator, achieving mean errors similar to other algorithms. Its computational time of 3.8788 seconds indicates moderate efficiency in solving the inverse kinematic problem.
- Ant Lion Optimizer: ALO showcased in Fig. 16 a competitive performance in minimizing position errors, with a mean error of 0.2393 mm, comparable to other algorithms. However, its orientation errors were slightly higher, indicating room for improvement. The computational time of 3.8788 seconds suggests reasonable efficiency in achieving accurate robotic arm configurations.

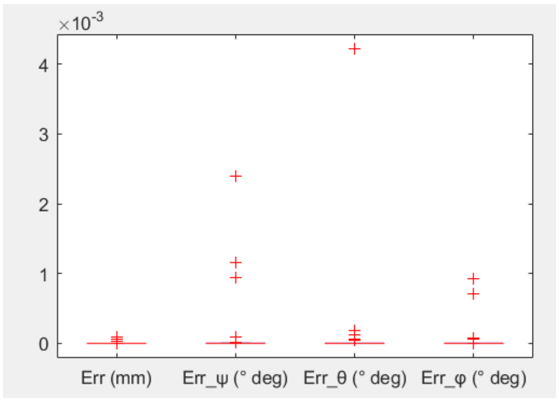


(a)

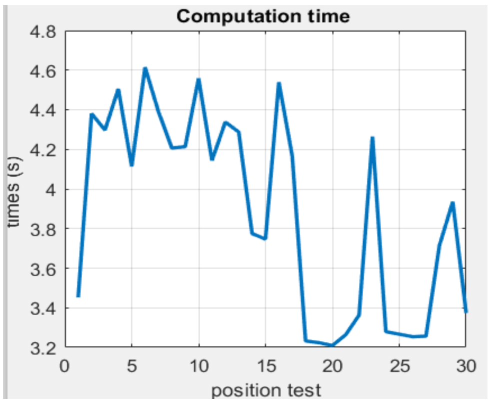


(b)

Fig. 13. Results of Ant Colony Optimization, (a) position and orientation errors, and (b) Computation time

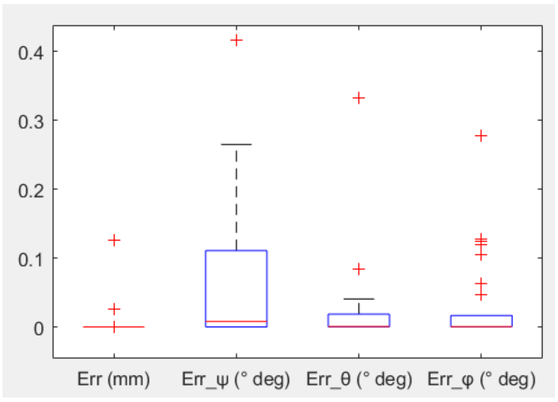


(a)

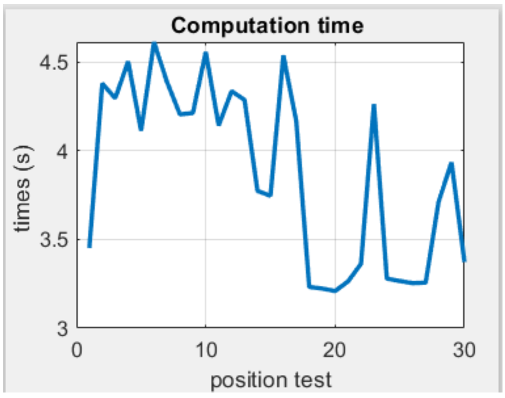


(b)

Fig. 14. Results of Mayfly Optimization Algorithm, (a) position and orientation errors, and (b) Computation time

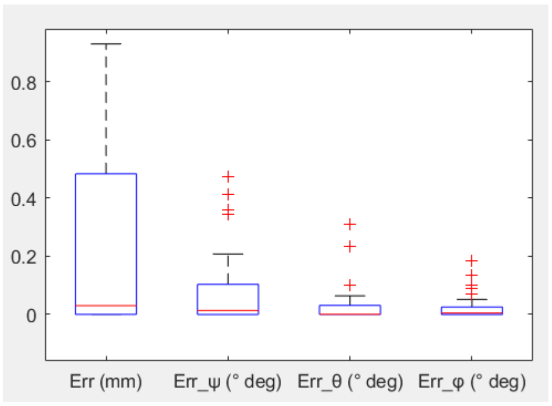


(a)

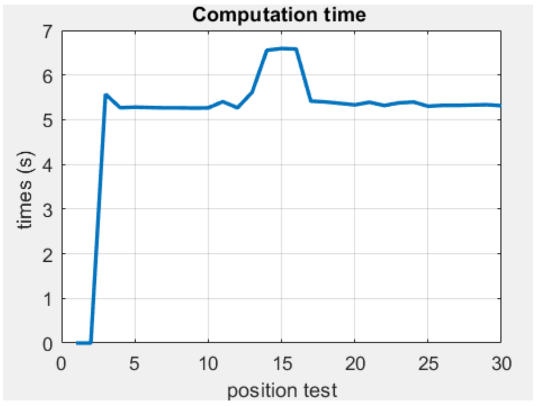


(b)

Fig. 15. Results of Stochastic Paint Optimizer Algorithm, (a) position and orientation errors, and (b) Computation time



(a)



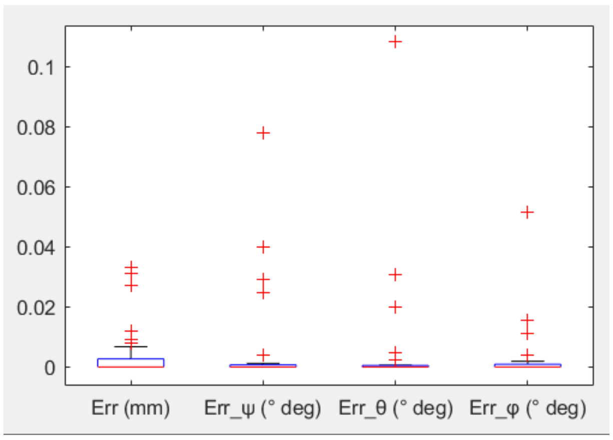
(b)

Fig. 16. Results of Ant Lion Optimizer algorithm, (a) position and orientation errors, and (b) Computation time

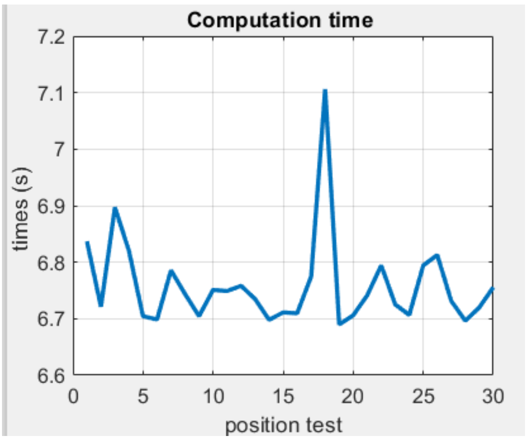
C. 6-DOF 12 Links Motoman SDA20D

We now focus on evaluating the effectiveness of individual metaheuristic algorithms on the 12-link, 6-degree-of-freedom Motoman SDA20D manipulator. Fig. 17 to Fig. 20 provide comprehensive graphical depictions of position and orientation errors, together with convergence durations for every algorithm used on this same robotic platform.

- Ant Colony Optimization: As shown in Fig. 17, ACO demonstrated a competitive performance in minimizing position errors for the Motoman SDA20D robotic arm, with a mean error of 0.0044 mm, indicating its effectiveness in achieving accurate robotic arm configurations. However, it exhibited relatively higher orientation errors compared to other algorithms, particularly in the ϕ angle. The longer computational time of 6.7593 seconds suggests a trade-off between accuracy and computational efficiency.
- Mayfly Optimization Algorithm: MO showcased remarkable accuracy in minimizing errors for the Motoman SDA20D robotic arm, achieving the lowest mean errors among the algorithms considered. Despite its longer computational time of 8.3173 seconds, its superior accuracy makes it a preferred choice for high-precision applications, as illustrated in Fig. 18.
- Stochastic Paint Optimizer Algorithm: As depicted in Fig. 19, SPO exhibited good performance in minimizing errors for the ABB IRB 1600 manipulator, achieving mean errors similar to other algorithms. Its computational time of 3.8788 seconds indicates moderate efficiency in solving the inverse kinematic problem.
- Ant Lion Optimizer: ALO demonstrated good performance in minimizing position errors for the Motoman SDA20D robotic arm, with a mean error of 0.0022 mm, as shown in Fig. 20, comparable to other algorithms. However, it exhibited relatively higher orientation errors, especially in the ϕ angle. The longer computational time of 8.0490 seconds suggests a trade-off between accuracy and computational efficiency, as seen in Fig. 19.

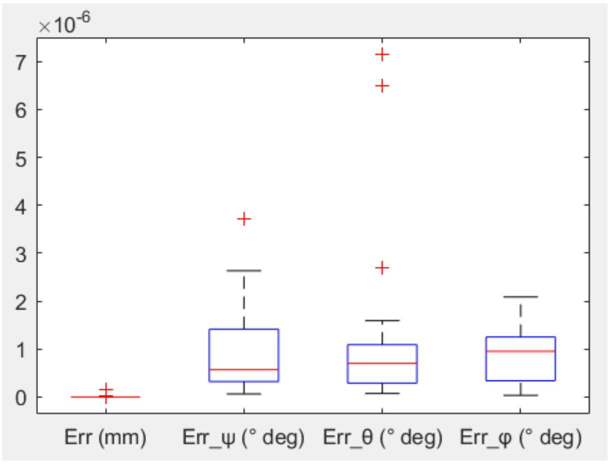


(a)

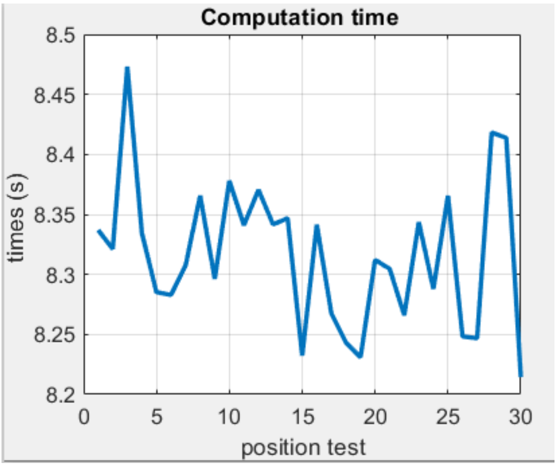


(b)

Fig. 17. Results of Ant Colony Optimization, (a) position and orientation errors, and (b) Computation time

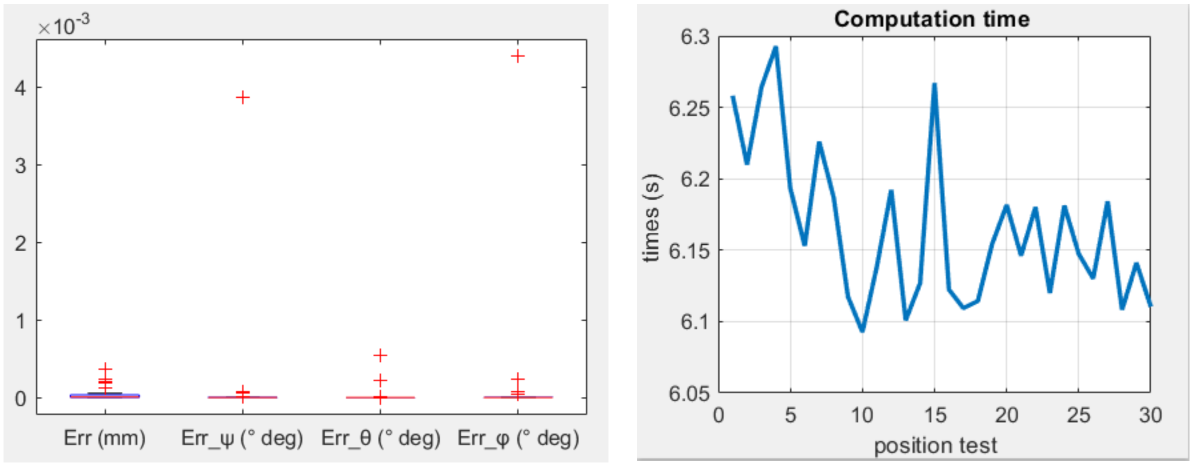


(a)



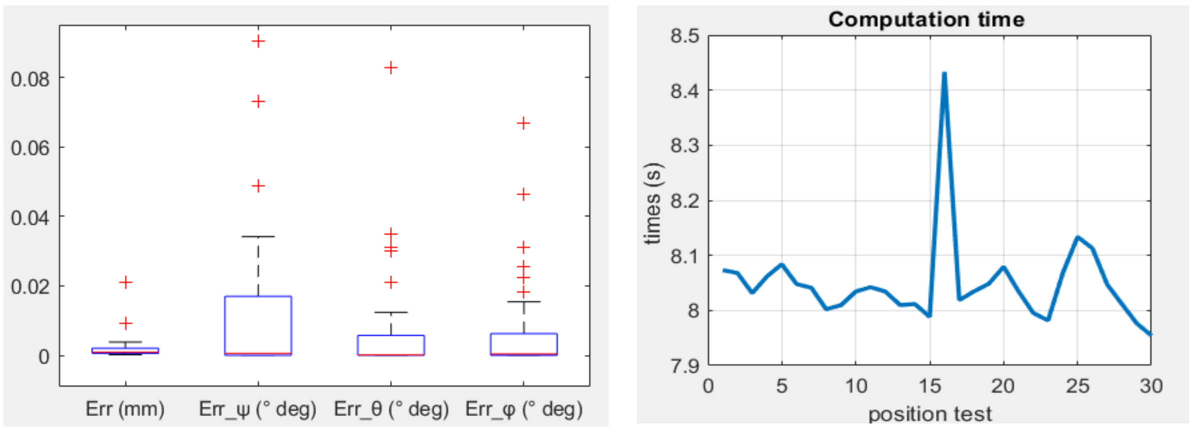
(b)

Fig. 18. Results of Mayfly Optimization Algorithm, (a) position and orientation errors, and (b) Computation time



(a) (b)

Fig. 19. Results of Stochastic Paint Optimizer Algorithm, (a) position and orientation errors, and (b) Computation time



(a) (b)

Fig. 20. Results of Ant Lion Optimizer algorithm, (a) position and orientation errors, and (b) Computation time

TABLE VIII. TABLE COMPARING THE RESULTS OF EVERY ROBOT ARM

Robot Arm	The algorithm	Results	MAX	MIN	MEAN
4-dof SCARA manipulator	**Ant Colony Optimization**	Position error Err (mm)	8.99e-05	0	7.55e-06
		Orientation error Err_{ψ} (° deg)	0	0	0
		Orientation error Err_{θ} (° deg)	0	0	0
		Orientation error Err_{φ} (° deg)	3.69e-04	3.8250e-08	1.9988e-05
		Times (s)	2.4634	1.0797	1.4913
	Mayfly Optimization Algorithm	Position error Err (mm)	9.17e-06	0	4.8115e-07
		Orientation error Err_{ψ} (° deg)	0	0	0
		Orientation error Err_{θ} (° deg)	0	0	0
		Orientation error Err_{φ} (° deg)	4.20e-04	2.9490e-08	1.4663e-05
		Times (s)	4.0249	2.2581	3.0788
	Stochastic Paint Optimizer Algorithm	Position error Err (mm)	6.99e-04	2.3611e-12	3.1412e-05
		Orientation error Err_{ψ} (° deg)	0	0	0
		Orientation error Err_{θ} (° deg)	0	0	0
		Orientation error Err_{φ} (° deg)	3.17e-04	9.3697e-09	2.4197e-05
		Times (s)	2.4797	1.3113	1.6283
	Ant Lion Optimizer algorithm	Position error Err (mm)	4.98e-05	3.8003e-06	2.12e-05
		Orientation error Err_{ψ} (° deg)	0	0	0
		Orientation error Err_{θ} (° deg)	0	0	0
		Orientation error Err_{φ} (° deg)	0.0981	4.3288e-06	0.0164
		Times (s)	5.2077	3.5073	3.8612
6-dof ABB IRB 1600 manipulator	**Ant Colony Optimization**	Position error Err (mm)	0.1625	0	0.0091
		Orientation error Err_{ψ} (° deg)	0.2606	5.0763e-08	0.0209
		Orientation error Err_{θ} (° deg)	0.1080	3.3750e-08	0.0071
		Orientation error Err_{φ} (° deg)	0.0974	1.4513e-07	0.0049
		Times (s)	4.6144	3.2092	3.8788
	Mayfly Optimization Algorithm	Position error Err (mm)	0.1625	0	0.0091
		Orientation error Err_{ψ} (° deg)	0.2606	5.0763e-08	0.0209

		Orientation error Err_θ (° deg)	0.1080	3.3750e-08	0.0071
		Orientation error Err_φ (° deg)	0.0974	1.4513e-07	0.0049
		Times (s)	4.6144	3.2092	3.8788
	Stochastic Paint Optimizer Algorithm	Position error Err (mm)	0.1625	0	0.0091
		Orientation error Err_ψ (° deg)	0.2606	5.0763e-08	0.0209
		Orientation error Err_θ (° deg)	0.1080	3.3750e-08	0.0071
		Orientation error Err_φ (° deg)	0.0974	1.4513e-07	0.0049
		Times (s)	4.6144	3.2092	3.8788
	Ant Lion Optimizer algorithm	Position error Err (mm)	0.9293	0	0.2393
		Orientation error Err_ψ (° deg)	0.4736	0	0.0850
		Orientation error Err_θ (° deg)	0.3105	0	0.0312
		Orientation error Err_φ (° deg)	0.1826	0	0.0269
		Times (s)	6.5926	0	5.1123
6-dof 12-link Motoman SDA20D	**Ant Colony Optimization**	Position error Err (mm)	0.0332	3.9399e-12	0.0044
		Orientation error Err_ψ (° deg)	0.0782	3.1056e-08	0.0060
		Orientation error Err_θ (° deg)	0.1083	5.7238e-08	0.0056
		Orientation error Err_φ (° deg)	0.0518	1.2966e-07	0.0030
		Times (s)	7.1060	6.6899	6.7593
	Mayfly Optimization Algorithm	Position error Err (mm)	1.49e-07	0	5.7463e-09
		Orientation error Err_ψ (° deg)	3.71e-06	6.3506e-08	8.7512e-07
		Orientation error Err_θ (° deg)	7.14e-06	7.0878e-08	1.1281e-06
		Orientation error Err_φ (° deg)	2.08e-06	3.2610e-08	8.6681e-07
		Times (s)	8.4731	8.2144	8.3173
	Stochastic Paint Optimizer Algorithm	Position error Err (mm)	3.73e-04	8.2627e-11	4.7558e-05
		Orientation error Err_ψ (° deg)	0.0039	1.2975e-08	1.3772e-04
		Orientation error Err_θ (° deg)	5.50e-04	8.5515e-09	2.7164e-05
		Orientation error Err_φ (° deg)	0.0044	3.0660e-08	1.5937e-04
		Times (s)	6.2927	6.0927	6.1651
	Ant Lion Optimizer algorithm	Position error Err (mm)	0.0209	2.2022e-04	0.0022
		Orientation error Err_ψ (° deg)	0.0904	1.0403e-05	0.0133
		Orientation error Err_θ (° deg)	0.0827	5.9961e-07	0.0081
		Orientation error Err_φ (° deg)	0.0668	5.3355e-06	0.0086
		Times (s)	8.4333	7.9543	8.0490

All simulations were performed on a standard desktop computer with an Intel Core i7-3600T 2.0 GHz processor and 8 GB DDR4 RAM. The algorithms were implemented in Python without GPU acceleration or parallel computing enhancements. As such, the reported computation times reflect single-threaded CPU performance. It is expected that substantial speed improvements could be achieved on more advanced systems or through optimized implementations using C++, multithreading, or GPU-based frameworks. Future studies should explore these performance gains, particularly for applications requiring real-time inverse kinematics solutions.

Table VIII summarizes each metric's mean, maximum, and minimum values. Across all robots, the Mayfly Optimization Algorithm (MOA) consistently achieved the lowest position and orientation errors, validating its strong global search capability and fine-grained convergence. However, MOA also had the highest average computation times, particularly for the more complex Motoman SDA20D (8.3173 s). ACO and SPOA demonstrated competitive accuracy with faster execution times, while ALO maintained acceptable position accuracy but was less robust in minimizing orientation error, especially for higher-DOF configurations.

As shown in Fig. 9 to Fig. 12, all four algorithms produced negligible orientation errors (almost zero) due to the limited complexity of the SCARA configuration. MOA achieved the **lowest mean position error** (4.81e-07 mm), followed by ACO (7.55e-06 mm) and SPOA (3.14e-05 mm). ALO's performance, while still acceptable, had a higher orientation error in the ϕ angle (mean 0.0164°), suggesting possible instability in rotational refinement. ACO and SPOA offered superior time efficiency (mean ≈ 1.5 s), making them suitable for time-sensitive SCARA-based tasks. MOA, despite its precision, required over 3 seconds on average, reflecting its intensive search dynamics.

Fig. 13 to Fig. 16 detail the results for the ABB IRB 1600. Here, MOA, ACO, and SPOA yielded **identical mean position errors** (0.0091 mm), yet ALO showed significant deviation (0.2393 mm). Orientation error analysis revealed that MOA maintained the most stable and accurate pose solutions, while ALO again demonstrated higher angular error (mean $E_\psi = 0.085°$). All algorithms showed similar computation times (≈ 3.8 s), although ALO's performance drop in precision indicates suboptimal adaptation to this 6-DOF workspace.

Fig. 17 to Fig. 20 reveal that this highly redundant system posed the greatest challenge. MOA significantly outperformed others, achieving a **mean position error of just 5.75e-09 mm** and **orientation error components below 1e-06°**. ACO and SPOA delivered reasonable results (mean position errors: 0.0044 mm and 4.75e-05 mm, respectively), though ACO's orientation error remained slightly elevated. ALO again struggled with orientation refinement (mean $E_\phi = 0.0086°$). However, its runtime (8.0490 s) was slightly lower than MOA's (8.3173 s), suggesting marginal speed gains at the cost of precision.

Compared to recent inverse kinematics research using nature-inspired or hybrid techniques, the results of this study show substantial improvements in accuracy and competitive computational efficiency.

For the 4-DOF SCARA robot, [17] reported a mean squared error (MSE) of 0.12846 mm using a hybrid PSO-ANN approach. In contrast, our MOA implementation achieved a mean position error of 4.81e-07 mm—an improvement of over two orders of magnitude—without requiring neural network training overhead. This underscores the strength of pure metaheuristic search when effectively tuned.

For the 6-DOF ABB IRB 1600, [18] applied NSGA-II and BCMOA, achieving a mean position error of ~0.8 mm and orientation error of 0.056°. In our work, MOA reduced these metrics to 0.0091 mm and 0.0209°, respectively, confirming significant accuracy gains. Additionally, [19] applied on a similar ABB robot, reported a mean position error of 0.0408 mm and orientation error of 0.0761°, with a very low runtime (~0.78 s). While Boomerang was faster, our results achieved higher precision, which is favorable in tasks prioritizing accuracy over speed.

For the Motoman SDA20D, [20] reported a position error of 0.52 mm using MOFOPSO. In contrast, our MOA approach yielded a mean position error of 5.75e-09 mm, which is dramatically lower and demonstrates the MOA's strong capability in handling high-DOF redundant kinematic structures. Even SPOA and ACO achieved superior precision compared to MOFOPSO.

These comparisons highlight that metaheuristic algorithms, when properly applied and tuned, can outperform hybrid or multi-objective methods from the literature—especially in achieving sub-millimeter or sub-degree accuracy essential in high-precision robotic applications.

VI. Conclusion

This study presents a comprehensive comparative analysis of four metaheuristic algorithms—Ant Colony Optimization (ACO), Mayfly Optimization Algorithm (MOA), Stochastic Paint Optimizer Algorithm (SPOA), and Ant Lion Optimizer (ALO)—for solving the inverse kinematics (IK) problem across three robotic manipulators: the 4-DOF SCARA, the 6-DOF ABB IRB 1600, and the 6-DOF 12-link Motoman SDA20D. A unified objective function combining position and orientation errors was optimized for each manipulator at 30 randomly sampled target positions.

The key findings show that the MOA consistently achieved the highest precision in both position and orientation tracking, particularly for redundant systems like the Motoman SDA20D. However, this came at the cost of significantly longer computation times—e.g., 8.3173 seconds on average for the most complex robot—making it less suitable for real-time applications without further optimization. In contrast, ACO and SPOA offered faster convergence with only minor accuracy trade-offs, making them viable for tasks with tighter execution constraints. ALO, while competitive in minimizing position errors, exhibited instability in orientation accuracy, especially in higher-DOF systems.

The primary theoretical contribution of this study is the development of a consistent benchmarking framework to evaluate metaheuristic-based IK solvers across manipulators of varying complexity. The analysis demonstrates the sensitivity of algorithm performance to kinematic structure, highlighting that no single algorithm is universally optimal. Moreover, this work reinforces the importance of balancing exploration-exploitation strategies, parameter tuning, and task-specific error tolerances when deploying metaheuristic methods in robotics.

Despite its contributions, the study has several limitations. First, all algorithms were tested under offline simulation conditions using 30 random target configurations, which may not fully capture singularities, boundary effects, or dynamically changing environments. Second, the evaluation did not include comparisons to classical deterministic IK methods (e.g., Jacobian pseudoinverse or CCD), leaving open questions about whether the increased computational overhead of metaheuristics is justified in scenarios demanding real-time precision. Third, parameter tuning was static, and the results may not generalize to structurally dissimilar robots without reconfiguration.

To extend the impact of this research, future work should include:

- Integration of adaptive parameter tuning or **hybrid metaheuristic–local optimization methods** to accelerate convergence,
- Application of **machine learning (ML)** and **deep learning (DL)** techniques, including **reinforcement learning (RL)**, to dynamically guide or initialize the IK solution space,
- Exploration of **hybrid metaheuristic–neural network frameworks** that leverage learned models to reduce search time while retaining solution diversity,
- Benchmarking against analytical and numerical IK solvers under noise and perturbation,
- Deployment of these algorithms on physical robots to assess real-world robustness,
- Expansion to more complex or high-DOF manipulators to evaluate algorithmic scalability.

In summary, this study provides valuable guidance for selecting and adapting metaheuristic algorithms to solve inverse kinematics challenges across various robotic platforms. While highly effective in accuracy, trade-offs in computational cost must be addressed before real-time deployment. The results lay a strong foundation for future exploration into scalable, robust, and adaptive IK optimization frameworks in modern robotics.

References

[1] D. Manocha and Y. Zhu, "A fast algorithm and system for the inverse kinematics of general serial manipulators," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 3348–3353, 1994, doi: 10.1109/ROBOT.1994.351055.

[2] R. R. Kumar and P. Chand, "Inverse kinematics solution for trajectory tracking using artificial neural networks for SCORBOT ER-4u," in *2015 6th International Conference on Automation, Robotics and*

Applications (ICARA), pp. 364–369, 2015, doi: 10.1109/ICARA.2015.7081175.

[3] A. Zamanzadeh and H. Ahmadi, "Inverse kinematic of European Robotic Arm based on a new geometrical approach," *AUT J. Mech. Eng.*, vol. 5, no. 1, Mar. 2021, doi: 10.22060/ajme.2020.17642.5866.

[4] C.-K. Ho and C.-T. King, "Automating the Learning of Inverse Kinematics for Robotic Arms with Redundant DoFs," *arXiv preprint arXiv:2202.07869*, 2022, doi: 10.48550/ARXIV.2202.07869.

[5] Y. Nakamura and H. Hanafusa, "Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control," *J. Dyn. Syst. Meas. Control*, vol. 108, no. 3, pp. 163–171, Sep. 1986, doi: 10.1115/1.3143764.

[6] H. N. Ghafil and K. Jármai, "Optimization Algorithms for Inverse Kinematics of Robots with MATLAB Source Code," in *Vehicle and Automotive Engineering 3*, pp. 468–477, 2021, doi: 10.1007/978-981-15-9529-5_40.

[7] K. K. Dash, B. B. Chaudhury, and S. K. Senapati, "Inverse Kinematics Analysis of an Industrial Robot Using Soft Computing," in *Applications of Robotics in Industry Using Advanced Mechanisms*, vol. 5, pp. 270–279, 2020, doi: 10.1007/978-3-030-30271-9_25.

[8] G. Singh and V. K. Banga, "Kinematics and Trajectory Planning Analysis Based On Hybrid Optimization Algorithms for an Industrial Robotic Manipulators," *Soft computing*, vol. 26, no. 21, pp. 11339-11372, 2022. doi: 10.21203/rs.3.rs-1313895/v1.

[9] S. Yaseen and J. Prakash, "Analysis of Numerical Method on Inverse Kinematics of Robotic Arm Welding with Artificial Intelligence," *J. Phys. Conf. Ser.*, vol. 1964, no. 6, p. 062104, Jul. 2021, doi: 10.1088/1742-6596/1964/6/062104.

[10] T. Aravinthkumar, M. Suresh, and B. Vinod, "Kinematic Analysis of 6 DOF Articulated Robotic Arm," *Int. Res. J. Multidiscip. Technovation*, pp. 1–5, Jan. 2021, doi: 10.34256/irjmt2111.

[11] M. Haohao *et al.*, "Inverse kinematics of six degrees of freedom robot manipulator based on improved dung beetle optimizer algorithm," *IAES Int. J. Robot. Autom. IJRA*, vol. 13, no. 3, p. 272, Sep. 2024, doi: 10.11591/ijra.v13i3.pp272-282.

[12] F. Jin and J. Zhai, "SCAPSO-based Inverse Kinematics Method and Its Application to Industrial Robotic Manipulator," in *2020 39th Chinese Control Conference (CCC)*, pp. 5933–5938, 2020, doi: 10.23919/CCC50068.2020.9188613.

[13] V. Vazquez-Castillo, J. Torres-Figueroa, E. A. Merchan-Cruz, E. Vega-Alvarado, P. A. Nino-Suarez, and R. G. Rodriguez-Canizo, "Inverse Kinematics Solution of Articulated Robots Using a Heuristic Approach for Optimizing Joint Displacement," *IEEE Access*, vol. 10, pp. 63132–63151, 2022, doi: 10.1109/ACCESS.2022.3182496.

[14] H. Deng and C. Xie, "An improved particle swarm optimization algorithm for inverse kinematics solution of multi-DOF serial robotic manipulators," *Soft Comput.*, vol. 25, no. 21, pp. 13695–13708, Nov. 2021, doi: 10.1007/s00500-021-06007-6.

[15] S. Zhang, A. Li, J. Ren, and R. Ren, "Kinematics inverse solution of assembly robot based on improved particle swarm optimization," *Robotica*, vol. 42, no. 3, pp. 833–845, Mar. 2024, doi: 10.1017/S0263574723001789.

[16] L. A. Nguyen, H. Danaci, and T. L. Harman, "Inverse Kinematics For Serial Robot Manipulator End Effector Position And Orientation By Particle Swarm Optimization," in *2022 26th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pp. 288–293, 2022, doi: 10.1109/MMAR55195.2022.9874317.

[17] R. Bouzid, J. Narayan, and H. Gritli, "Hybrid Metaheuristic and Artificial Neural Network Approach for Solving Inverse Kinematics of a SCARA Manipulator Robot," in *2024 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pp. 385–392, 2024, doi: 10.1109/3ict64318.2024.10824671.

[18] C. A. Pena, M. A. Guzman, and P. F. Cardenas, "Inverse kinematics of a 6 DOF industrial robot manipulator based on bio-inspired multi-objective optimization techniques," in *2016 IEEE Colombian Conference on Robotics and Automation (CCRA)*, pp. 1–6, 2016, doi: 10.1109/CCRA.2016.7811428.

[19] O. Duymazlar and D. EngiN, "Boomerang Algorithm based on Swarm Optimization for Inverse Kinematics of 6 DOF Open Chain Manipulators," *Turk. J. Electr. Eng. Comput. Sci.*, vol. 31, no. 2, pp. 342–359, Mar. 2023, doi: 10.55730/1300-0632.3988.

[20] S. Luo, D. Chu, Q. Li, and Y. He, "Inverse Kinematics Solution of 6-DOF Manipulator Based on Multi-Objective Full-Parameter Optimization PSO Algorithm," *Front. Neurorobotics*, vol. 16, p. 791796, Mar. 2022, doi: 10.3389/fnbot.2022.791796.

[21] L. C. Antonio-Gopar, C. Lopez-Franco, N. Arana-Daniel, E. Gonzalez-Vallejo, and A. Y. Alanis, "Inverse Kinematics for a Manipulator Robot based on Differential Evolution Algorithm," in *2018 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*, pp. 1–5, 2018, doi: 10.1109/LA-CCI.2018.8625233.

[22] H. K. Dave, M. D. Chanpura, S. J. Kathrotiya, D. D. Patolia, D. D. Dodiya, and P. S. Kharva, "Design, Development and Control of SCARA for Manufacturing Processes," in *Recent Advances in Manufacturing Modelling and Optimization*, pp. 551–567, 2022, doi: 10.1007/978-981-16-9952-8_47.

[23] X.-S. Yang, "Metaheuristic Optimization: Algorithm Analysis and Open Problems," in *Experimental Algorithms*, vol. 6630, pp. 21–32, 2011, doi: 10.1007/978-3-642-20662-7_2.

[24] A. H. Halim, I. Ismail, and S. Das, "Performance assessment of the metaheuristic optimization algorithms: an exhaustive review," *Artif. Intell. Rev.*, vol. 54, no. 3, pp. 2323–2409, Mar. 2021, doi: 10.1007/s10462-020-09906-6.

[25] X.-S. Yang, A. H. Gandomi, S. Talatahari, and A. H. Alavi, *Metaheuristics in water, geotechnical and transport engineering*, 1st. edition. in Elsevier insights. Amsterdam: Elsevier, 2012.

[26] A. G. Bakirtzis, P. N. Biskas, C. E. Zoumas, and V. Petridis, "Optimal power flow by enhanced genetic algorithm," *IEEE Trans. Power Syst.*, vol. 17, no. 2, pp. 229–236, May 2002, doi: 10.1109/TPWRS.2002.1007886.

[27] H. S. Maharana and S. K. Dash, "Quantum behaved artificial bee colony based conventional controller for optimum dispatch," *Int. J. Electr. Comput. Eng. IJECE*, vol. 13, no. 2, p. 1260, Apr. 2023, doi: 10.11591/ijece.v13i2.pp1260-1271.

[28] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi, "A particle swarm optimization for reactive power and voltage control considering voltage security assessment," *IEEE Trans. Power Syst.*, vol. 15, no. 4, pp. 1232–1239, Nov. 2000, doi: 10.1109/59.898095.

[29] G. Zhao *et al.*, "A Tandem Robotic Arm Inverse Kinematic Solution Based on an Improved Particle Swarm Algorithm," *Front. Bioeng. Biotechnol.*, vol. 10, p. 832829, May 2022, doi: 10.3389/fbioe.2022.832829.

[30] A. Kaveh, S. Talatahari, and N. Khodadadi, "Stochastic paint optimizer: theory and application in civil engineering," *Eng. Comput.*, vol. 38, no. 3, pp. 1921–1952, Jun. 2022, doi: 10.1007/s00366-020-01179-5.

[31] L. Abualigah, M. Shehab, M. Alshinwan, S. Mirjalili, and M. A. Elaziz, "Ant Lion Optimizer: A Comprehensive Survey of Its Variants and Applications," *Arch. Comput. Methods Eng.*, vol. 28, no. 3, pp. 1397–1416, May 2021, doi: 10.1007/s11831-020-09420-6.

[32] P. K. Sahu, G. Balamurali, G. B. Mahanta, and B. B. Biswal, "A Heuristic Comparison of Optimization Algorithms for the Trajectory Planning of a 4-axis SCARA Robot Manipulator," in *Computational Intelligence in Data Mining*, vol. 711, pp. 569–582, 2019, doi: 10.1007/978-981-10-8055-5_51.

[33] M. Dorigo and T. Stützle, "Ant Colony Optimization: Overview and Recent Advances," in *Handbook of Metaheuristics*, vol. 272, pp. 311–351, 2019, doi: 10.1007/978-3-319-91086-4_10.

[34] S. Manakkadu and S. Dutta, "ACO based Adaptive RBFN Control for Robot Manipulators," *arXiv preprint arXiv:2208.09165*, 2022, doi: 10.48550/ARXIV.2208.09165.

[35] V. Maniezzo and A. Carbonaro, "Ant Colony Optimization: An Overview," in *Essays and Surveys in Metaheuristics*, vol. 15, pp. 469–492, 2002, doi: 10.1007/978-1-4615-1507-4_21.

[36] C. Wang, B. Song, H. Zhang, and C. Sun, "Analysis of passive location communication system based on intelligent optimization algorithm," *J. Phys. Conf. Ser.*, vol. 1325, no. 1, p. 012147, Oct. 2019, doi: 10.1088/1742-6596/1325/1/012147.

[37] M. Couceiro and P. Ghamisi, "Particle Swarm Optimization," in *Fractional Order Darwinian Particle Swarm Optimization*, pp. 1–10, 2016, doi: 10.1007/978-3-319-19635-0_1.

[38] M. Fernandez, J. Caballero, L. Fernandez, and A. Sarai, "Genetic algorithm optimization in drug design QSAR: Bayesian-regularized genetic neural networks (BRGNN) and genetic algorithm-optimized

support vectors machines (GA-SVM)," *Mol. Divers.*, vol. 15, no. 1, pp. 269–289, Feb. 2011, doi: 10.1007/s11030-010-9234-9.

[39] X. S. Yang and X. He, "Firefly algorithm: recent advances and applications," *Int. J. Swarm Intell.*, vol. 1, no. 1, p. 36, 2013, doi: 10.1504/IJSI.2013.055801.

[40] K. Zervoudakis and S. Tsafarakis, "A mayfly optimization algorithm," *Comput. Ind. Eng.*, vol. 145, p. 106559, Jul. 2020, doi: 10.1016/j.cie.2020.106559.

[41] A. Sundaram and N. S. Alkhaldi, "Multi-Objective Stochastic Paint Optimizer for Solving Dynamic Economic Emission Dispatch with Transmission Loss Prediction Using Random Forest Machine Learning Model," *Energies*, vol. 17, no. 4, p. 860, Feb. 2024, doi: 10.3390/en17040860.

[42] A. Kumar, V. Kumar, and V. Modgil, "Performance modeling and optimization for complex repairable system of paint manufacturing unit using a hybrid BFO-PSO algorithm," *Int. J. Qual. Reliab. Manag.*, vol. 36, no. 7, pp. 1212–1228, Aug. 2019, doi: 10.1108/IJQRM-02-2018-0041.

[43] N. Sharma, Varun, and Siddhartha, "Stochastic techniques used for optimization in solar systems: A review," *Renew. Sustain. Energy Rev.*, vol. 16, no. 3, pp. 1399–1411, Apr. 2012, doi: 10.1016/j.rser.2011.11.019.

[44] M. Azizi and S. Talatahari, "Enhanced Stochastic Paint Optimizer for Nonlinear Design of Fuzzy Logic Controllers in Steel Building Structures for the Near-Fault Earthquakes," in *Advances in Civil and Industrial Engineering*, pp. 306–335, 2023, doi: 10.4018/978-1-6684-5643-9.ch012.

[45] S. Mirjalili, "The Ant Lion Optimizer," *Adv. Eng. Softw.*, vol. 83, pp. 80–98, May 2015, doi: 10.1016/j.advengsoft.2015.01.010.

[46] S. Mirjalili, P. Jangir, and S. Saremi, "Multi-objective ant lion optimizer: a multi-objective optimization algorithm for solving engineering problems," *Appl. Intell.*, vol. 46, no. 1, pp. 79–95, Jan. 2017, doi: 10.1007/s10489-016-0825-8.

[47] D. Oliva, S. Hinojosa, M. A. Elaziz, and N. Ortega-Sánchez, "Context based image segmentation using antlion optimization and sine cosine algorithm," *Multimed. Tools Appl.*, vol. 77, no. 19, pp. 25761–25797, Oct. 2018, doi: 10.1007/s11042-018-5815-x.

[48] S. Luke, *Essentials of metaheuristics: a set of undergraduate lecture notes*. Washington, DC: The author, 2010.

[49] S. Dereli and R. Köker, "Simulation based calculation of the inverse kinematics solution of 7-DOF robot manipulator using artificial bee colony algorithm," *SN Appl. Sci.*, vol. 2, no. 1, 2020, doi: 10.1007/s42452-019-1791-7.

[50] A. Kumar, V. K. Banga, D. Kumar, and T. Yingthawornsuk, "Kinematics Solution using Metaheuristic Algorithms," in *2019 15th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*, pp. 505–510, 2019, doi: 10.1109/SITIS.2019.00086.