УДК 621.865:004.896 DOI: 10.17587/mau.19.714-724

I. N. Ibrahim, Ph. D. Student, ibrncfe@gmail.com, http://orcid.org/0000-0001-9544-3020,
Department of Mechatronics and Robotics, Kalashnikov Izhevsk State Technical University,
Izhevsk, 426069, Russian Federation

A Comparative Study for an Inverse Kinematics Solution of an Aerial Manipulator Based on the Differential Evolution Method and the Modified Shuffled Frog-Leaping Algorithm

Accepted on July 16, 2018

Abstract

This paper focuses on the real-time kinematics solution of an aerial manipulator mounted on an aerial vehicle, the vehicle's motion isn't considered in this study. Robot kinematics using Denavit-Hartenberg model was presented. The fundamental scope of this paper is to obtain a global online solution of design configurations with a weighted specific objective function and imposed constraints are fulfilled. Acknowledging the forward kinematics equations of the manipulator; the trajectory planning issue is consequently assigned to on an optimization issue. Several types of computing methods are documented in the literature and are well-known for solving complicated nonlinear functions. Accordingly, this study suggests two kinds of artificial intelligent techniques which are regarded as search methods; they are differential evolution (DE) method and modified shuffled frog-leaping algorithm (MSFLA). These algorithms are constrained metaheuristic and population-based approaches. moreover, they are able to solve the inverse kinematics problem taking into account the mobile platform additionally avoiding singularities since it doesn't demand the inversion of a Jacobian matrix. Simulation results are carried out for trajectory planning of 6 degree-of-freedom (DOF) kinematically aerial manipulator and confirmed the feasibility and effectiveness of the supposed methods.

Keywords: *Inverse Kinematics, Degree-of-Freedom (DOF), Human-like Aerial Manipulator, Optimization Algorithms, metaheuristic and Revolutionary Methods, differential evolution (DE), Shuffled Frog Leaping Algorithm*

1. Introduction

The inverse kinematics (IK) solver is a primary problem in robotic manipulation, particularly when demand real-time and precision in calculations. Mathematically, the numerical solution of kinematics is intricate because of the high degree of nonlinearity, furthermore, the Linear and dynamic programming techniques usually fail or reach local optimum in solving NP-hard problems with a large number of variables and non-linear objective functions, moreover, Traditionally Jacobian-based solutions are identified to scale inadequately with the high number of degrees of freedom (DOF) [1] in addition to singularities existence. In contrast, [2] presented a comparative study of several methods based on the Jacobian matrix, clarifying that the modified Levenberg—Marquardt method is much better for a quite large set of random configurations than others but may lose convergence compared to Jacobian transpose and Pseudocode inverse methods. Recently many researchers [3] proposed a new method for solving real-time IK without using the Jacobian matrix based on the position of end-effector (ee), using numerical and analytical mathematical tools but not mentioned exactly the performance as the time consuming to get the solution, in [4] also applied alike method for hyper-redundant manipulator arm. [5] combined two methods as a real-time IK solver for a human-like arm manipulator based on closed-form analytical equations for a given position while others [6] presented an on-line adaptive strategy based on the Lyapunov stability theory in addition to Radial Basis Function Network (RBFN) and quadratic programming which requires a complex hardware resources, the simulation was done for the position of ee in addition to avoid obstacles and was conducted on the 7-DOF PA-10 robot manipulator. In [7] a kinematic and time-optimal trajectory planning was considered for redundant robots, two approaches were presented, joint space decomposition and a numerical null-space method for a given pose. they were tested by 7-DOF industrial robots and demand high consuming time for resolving IK.

Metaheuristic optimization algorithms are an encouraging alternative approach to traditional IK techniques due to their strong performance on challenging and high-DOF problems in many various domains, the solution can be solved by minimizing an objective function allowing the end-effector to follow the desired path avoiding dynamics singularities and obstacles [8] was explained and proved that DE has emerged as one of the most powerful and versatile global numerical optimizers for non-differential and multimodal problems, they showed challenges of the variants of DE which may provide less time and more robustness in solving IK. In [9] presented a quadratic programming with branching idea with a weighted multi-objective function which gave a short-time response about seconds while [10] showed a comparative research of four different heuristic optimization algorithms GA, PSO, QPSO and GSA for 4-DOF manipulator in order to reach the target as a position. they proved that the Quantum PSO is the best with average execution time of 1.65 seconds. In [11] investigated the performance of many PSO variants to resolve two DOF IK problem for a given position, they proved that PSO-VG is the fast variant compared with others which took a less average convergence iteration about 740 for 15 particles. [12] derived and minimized a fitness function to resolve the pose IK problem based on PSO for multiple DOF up to 180, they concluded that the runtime and iteration are 4.22 seconds and 118 respectively for a 9-DoF. In [13] a hybrid method called DEMPSO based on differential evolution (DE) and Modified PSO algorithms was developed in order to minimize the solution time for the pose, moreover presented a comparative study for several swarm intelligent optimization algorithms as ABC and ACO algorithms, based on their results, the DEMPSO had a great advantages at execution time for reaching the position while similar performance with DE for the orientation aim, the simulation was conducted with population size 30 for 10-DoF serial-parallel robot, furthermore, [14] a comparison of three evolutionary algorithms as GA, PSO, and DE was discussed. In [15] presented a comparative study of IK solver for a mobile manipulator using DE algorithm, they concluded that hybrid DE and biogeography-based optimization called HBBO provides good results but a higher computational cost for weighted fitness function and pose target, in contrast, DE proved to be superior to PSO, CS, and TLBO, additionally the PSO algorithm verified that it does not solve the inverse kinematic problems correctly. In [16] a developed methodology was applied to synthesize of six-bar mechanism, it used DE with geometric centroid of precision positions technique (GCCP). [17] used DE to improve the design of a fuzzy controller for a wall-following hexapod robot. In addition [18] proposed a modified self-adaptive DE in order to improve the static force of humanoids robot, showing robust, safe, reliable performance compared with other metaheuristics. While [19] presented an approximation tool for the inverse model of the industrial robot based on an adaptive neural model optimized by advance DE. [20] proposed an optimal joint trajectory planning method using forward kinematics of 7-DoF free-floating space robot based on DE method, also they depict the general aspect of equality and inequality constraints which govern each joint in the manipulator. On another-side, [21] introduced an algorithm shuffled frog-leaping algorithm SFLA which is a population-based collaborative search metaphor inspired by natural memetics, it relies on a knowledge called 'meme' which causes someone to replicate it or to repeat it to someone else, spreading from brain to brain. All transmitted knowledge is memetic and spreading is much faster than a gene, the effectiveness, suitability, and global optimal resolving have been demonstrated in addition to short processing time. Additionally, [22] proposed an MSFLA for a high dimensional continuous function optimization. this method yields a strong robustness and best convergence also presented a comparative study for PSO, SFLA, MSFLA, and MSFLA-EO which designated that MSFLA is better than others. In [23] a modified SFLA was assumed for obtaining the optimum preventive maintenance scheduling of generating units in power system. While [24] presented a comparative study among five evolutionary-based optimization algorithms as GA, MA, PSW, ACO, and SFLA, they showed the processing time for solving the F8 function and concluded that the SFLA is the best.

In this work, we applied two types of metaheuristic algorithms in order to solve the inverse kinematics of a mobile manipulator [25] as a constrained optimization problem the proposed algorithms are the differential evolution (DE) method and modified shuffled frog-leaping algorithm (MSFLA), which characterized as an accurate and fast convergence in discovering the solution based on the previous literature study. Initially, we define an objective function to minimize the error between the desired and the actual end-effector pose. The objective function takes into account the minimal movement between the previous and the actual joint

configurations. To overcome the constrained problems, we use a penalty function to penalize all those manipulator configurations that violate the allowed joint boundary. Hence, the proposed approach estimates the feasible manipulator configuration needed to reach the desired end-effector pose. The remainder of this paper is organized as follows: Section 2 and 3 present the architecture and kinematics of a robotic manipulator. Section 4 introduces metaheuristic optimization algorithms and the weighted objective function. Section 5 shows the simulation results of the proposed trajectory planning methods applied to a 6-DOF kinematically manipulator. The conclusive observations are listed in the last section.

2. The architecture of the aerial manipulator

The aerial manipulator was designated in [25] as a human-like arm and consists of 20 degrees of freedom. It was proposed that each joint has one DOF with revolute type while the links are a type of rigid-body which organized from the base-frame located in the center of aerial vehicle's gravity with a constant displacement based on a design consideration for the mass distribution. While the other main-frame called hand-frame located in the center of the hand part 7th frame (in the Fig. 1 has a number 7, see the 3rd side of cover) that was considered as a reference node for combining and moving the fingers in order to perform specific object-oriented tasks [25]. Furthermore, the realized prototype of the manipulator as shown in Fig. 1 was divided into two parts are arm-part and hand-part, the arm-part architecture consists of 6 joints originated from shoulder joints (J_1, J_2) to the forearm joint (J_3) and finally, wrist joints (J_4, J_5, J_6) along to center hand's frame. The specifications of the arm-part are given in the Table 1 which consists of six joints and four 3d-printed links in addition to two metal links as metal horns and brackets, besides, the scheme of this part is designated to be anti-resistance of the air friction in design as possible as shown in the figure. the last link is the hand-part which consists of five fingers driven by five micro-servo motors based on linkage mechanism out of scope in this paper. the total number of degrees-of-freedom for the intended arm-part is six.

Table 1

Specification of the Arm-Part

l_i	Between J_i and J_{i+1}	Length (cm)
l_0	link (Between J_0 and J_1)	3.5
l_1	link (Between J_1 and J_2)	6.5
l_2	link (Between J_2 and J_3)	28.2
l_3	link (Between J_3 and J_4)	21
l_4	link (Between J_4 and J_5)	2.5
l_5	link (Between J_5 and J_6)	7.5
l_6	link (Between J_6 and J_7)	5

The navigation process of the arm-part is performed by controlling the motion of hand-part located in the 7th frame in the workspace related to the base frame. Furthermore, this process is computed by analyzing the forward and inverse kinematics. the goal of this paper is to study the solutions of the inverse kinematics in order to realize this movement precisely within a short time. Hence, each of links was described by some of the properties as material type, stiffness, toughness, bearing gear, shape, weight, inertia, lubrication in addition to aerodynamic parameters. The values of motion range for the joints are manifested in the following Table 2 which readjusted to be more fitting for accomplishing more tasks compared to real joints of the humanlike arm [25].

3. Manipulator Kinematics

Now, in order to determine the relationship between the coordinate frames, that are assigned to the links and joints of the robot, homogeneous transformations are required. Three parameters are employed to describe the rotation while other three parameters are used to define the translation. accordingly, the Denavit Hartenberg (DH) convention was used to describe kinematically the rigid motion by assigning the values of four quantities for each link. Two describe the link itself, and two describe the link's connection to a neighboring link. Where θ, a, d and α are the joint angle, link length, link offset and link twist between joints. Considering T_i is the homogeneous transformation matrix between the frames that is a function of θ while the other three parameters are constant. The data in

Table 2

The motion range for the joints of the Arm-Part

Arm-part	angle	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
	range	$-90 \rightarrow +90$	$0 \rightarrow +180$	$-90 \rightarrow +90$	$-90 \rightarrow +90$	$-90 \rightarrow +90$	$-70 \rightarrow +90$

Table 3

Link parameters of the arm-part of manipulator

Modified Denavit Hartenberg				
α_{i-1}	α_{i-1} [cm]	d_i [cm]	θ_i	Initial Value of θ_i
$-\pi/2$	l_0	0	θ_1	$\pi/2$
$\pi/2$	l_1	0	θ_2	$-\pi/2$
0	l_2	0	θ_3	$-\pi/2$
$-\pi/2$	0	$l_3 + l_4$	θ_4	0
$\pi/2$	0	0	θ_5	$-\pi/2$
$-\pi/2$	l_5	0	θ_6	0

Standard Denavit Hartenberg					
α_i	a_i [cm]	d_i [cm]	θ_i	Initial Value of θ_i	Joint Offset
$-\pi/2$	6.4	0	θ_1	0	0
0	30.2	0	θ_2	$-\pi/2$	$-\pi/2$
$\pi/2$	0	0	θ_3	$\pi/2$	$\pi/2$
$\pi/2$	0	23.5	θ_4	0	0
$-\pi/2$	5.3	0	θ_5	$\pi/2$	$\pi/2$
0	5.6	-2	θ_6	0	0

the Table 3 present link parameters of the arm-part based on DH strategy in two formulas: standard and modified DH. Since the standard form was used in simulation by LabVIEW Robotics module in order to validate the design. The position of all links of an arm-part manipulator can be specified with a set of 6 joint variables from the shoulder's joints till wrist's joints. This set of variables is often referred to as a 6×1 joint vector [25].

The space of all joint variables is referred to as joint-space $\Theta = [\theta_1, \theta_2, ..., \theta_6]^T$, here we have been concerned with computing the Cartesian space representation from knowledge of the joint-space information. hence the homogeneous transformations of links are as following. these transformations, ${}^{i-1}_{i}T$, will be a function of all joint variables. If the robot's joint-position sensors are estimated by servo mechanisms, the Cartesian position and orientation of the hand-part can be computed by ${}^{0}_{7}T$ [25].

$$
{}^{0}_{1}T = \begin{bmatrix} C_1 & -S_1 & 0 & l_0 \\ 0 & 0 & 1 & 0 \\ -S_1 & -C_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^{1}_{2}T = \begin{bmatrix} C_2 & -S_2 & 0 & l_1 \\ 0 & 0 & -1 & 0 \\ S_2 & C_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

$$
{}^{2}_{3}T = \begin{bmatrix} C_3 & -S_3 & 0 & l_2 \\ S_3 & C_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^{3}_{4}T = \begin{bmatrix} C_4 & -S_4 & 0 & 0 \\ 0 & 0 & 1 & l_3 + l_4 \\ -S_4 & -C_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

$$
{}^{4}_{5}T = \begin{bmatrix} C_5 & -S_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ S_5 & C_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, {}^{5}_{6}T = \begin{bmatrix} C_6 & -S_6 & 0 & l_5 \\ 0 & 0 & 1 & 0 \\ -S_6 & -C_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

$$
{}^{6}_{7}T = \begin{bmatrix} 1 & 0 & 0 & l_6 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.
$$

Where the computations of the transformation as follows:

$$
{}^{0}_{7}T = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{11}l_6 \\ b_{21} & b_{22} & b_{23} & b_{21}l_6 \\ b_{31} & b_{32} & b_{33} & b_{31}l_6 \\ 0 & 0 & 0 & 1 \end{bmatrix},
$$

$$
\begin{aligned}
b_{11} &= [(C\theta_1 C\theta_{23} C\theta_4 - S\theta_1 S\theta_4) C\theta_5 - C\theta_1 S\theta_{23} S\theta_5] C\theta_6 + \\
&+ (-C\theta_1 C\theta_{23} C\theta_4 S\theta_5 + S\theta_1 S\theta_4 S\theta_5 - S\theta_5 S\theta_1 S\theta_{23}) S\theta_6; \\
b_{12} &= -[(C\theta_1 C\theta_{23} C\theta_4 - S\theta_1 S\theta_4) C\theta_5 - C\theta_1 S\theta_{23} S\theta_5] S\theta_6 + \\
&+ (-C\theta_1 C\theta_{23} S\theta_4 + S\theta_1 S\theta_4) C\theta_6; \\
b_{13} &= -(C\theta_1 C\theta_{23} C\theta_4 - S\theta_1 S\theta_4) S\theta_5 - C\theta_1 S\theta_{23} S\theta_5; \\
b_{21} &= (C\theta_1 S\theta_{23} C\theta_4 + C\theta_{23} S\theta_5) C\theta_6 - S\theta_{23} S\theta_4 S\theta_6; \\
b_{22} &= -(S\theta_{23} C\theta_4 C\theta_5 + C\theta_{23} S\theta_5) S\theta_6 - S\theta_{23} S\theta_4 C\theta_6; \\
b_{23} &= -C\theta_4 S\theta_{23} S\theta_5 + C\theta_{23} C\theta_5; \\
b_{31} &= [(S\theta_1 C\theta_{23} - C\theta_1 S\theta_4) C\theta_5 + S\theta_1 S\theta_{23} S\theta_5] C\theta_6 + \\
&+ (S\theta_1 C\theta_{23} S\theta_4 + C\theta_1 C\theta_4) S\theta_6; \\
b_{32} &= -[(S\theta_1 C\theta_{23} - C\theta_1 S\theta_4) C\theta_5 + S\theta_1 S\theta_{23} S\theta_5] S\theta_6 - \\
&- (S\theta_1 C\theta_{23} S\theta_4 + C\theta_1 C\theta_4) C\theta_6; \\
b_{33} &= (S\theta_1 C\theta_{23} - C\theta_1 S\theta_4) S\theta_5 + S\theta_1 S\theta_{23} C\theta_5.
\end{aligned}
$$

4. Proposed Optimization Techniques or solving kinematics

The evolutionary optimization algorithms can solve the complicated nonlinear equations completely and efficiently. The solution of the inverse kinematics for the manipulator results is a very difficult problem to solve by traditional approaches. Besides, the suggested strategies do not require the inversion of any Jacobian matrix, and then it avoids singularities configurations. In this paper, two algorithms used to optimize this problem are differential evolution and modified shuffled frog-leaping algorithm. In general, this solution is based on the forward kinematics

equations which always produces a solution in addition to the objective function. Hence, the general aspect of the problem may express as minimizing $J(\Theta)$, constrained by $\Theta_{\min} \leqslant \Theta \leqslant \Theta_{\max}$, furthermore, the objective function could be defined as the weighted sum of the errors as follows

$$J(\Theta) = \sigma P_{error}(\Theta) + \beta O_{error}(\Theta) = \\ = \sigma \left\| P_G - P_E(\Theta) \right\| + \varepsilon \left\| O_G - O_E(\Theta) \right\|, \quad (1)$$

where $P_{error}(\Theta)$ and $O_{error}(\Theta)$ represent the position and orientation errors respectively and could be computed as a difference in distance between the target and current position, in this work we used Euclidean formula as a representation of distance. While the parameters σ and ε are the weights of the position and the orientation, respectively. Let $G = (P_G,\ O_G)$ be a given target end-effector pose while $E(\Theta) = (P_E(\Theta),\ O_E(\Theta))$ is the current end-effector pose in the workspace corresponding to configuration $\Theta = [\theta_1,\ \theta_2 ... \theta_6]^T$ which can be calculated using forward kinematics — where P refers to the 3D position vector of pose while O refers to vector of Roll-Pitch-Yaw Euler Angles of pose (in radians), respectively, that optimization algorithms are exploring directly in the configuration space of the manipulator. Hence, each individual $\Theta_i = [\theta_{i,1}, \theta_{i,2} ... \theta_{i,j} ... \theta_{i,6}]^T$ represents an i-th candidate set of joint angles. henceforward, at each iteration, we evaluate each candidate configuration Θ_i by passing it through the forward kinematics model and measuring the position and orientation error between where the end-effector would be at configuration Θ_i and the target end-effector pose. In order to enforce joint limits, each dimension j of element Θ_i should be limited to searching in the range of valid joint angles $\Theta_i \in [\Theta_{\min},\ \Theta_{\max}]$. This can be realized by clamping each dimension j within these bounds at each iteration immediately after it is updated.

4.1. Differential Evolution Algorithm

The first study on DE algorithm was introduced by Storn and Price [8], [18], [26]. It is one of the most powerful stochastic population-based optimization algorithms. It was invented to optimize functions in an n-dimensional continuous domain. moreover, it occupies several benefits such as simple implementation, good performance, global optimization, robust, low space complexity, converges fast, and has a good balance between exploration and exploitation. In particular, DE is relevant to standard evolutionary algorithms in which a population of candidate solutions, initialized to a uniform sampling of the instance space, are continuously enhanced by periodically adding a scaled variant of the difference vector to a third individual to generate a new candidate solution and then producing the succeeding generation. DE consists of four stages: initialization, mutation, crossover, and selection. The last three of these are iterated until a termination condition such as the maximum number of generations is reached. Nevertheless, unlike other evolutionary algorithms before-mentioned as evolution strategies, mutation is performed by applying the scaled difference between members of the population. This has the impact of adjusting the step size to the fitness aspect over time. The implementation of this method is illustrated in Algorithm 1.

Algorithm 1: The pseudo-code of the differential evolution algorithm

Initialization:
$Population^{(1)} \leftarrow \{\Theta_1^{(1)}, \Theta_2^{(1)}, \ldots, \Theta_i^{(1)}, \ldots, \Theta_{NP}^{(1)}\}, g \leftarrow 1, g_{\max}$
Evolution Process:
  While *Termination criteria not met* do
  for $i \leftarrow 1, NP$ *do*
    Mutation Process: $v_i^{(g)} \leftarrow mutate(\Theta_i^{(g)})$
    Crossover Process: $u_i^{(g)} \leftarrow crossover(\Theta_i^{(g)}, v_i^{(g)})$
    Selection Process:
    if $f(u_i^{(g)}) \leqslant f(\Theta_i^{(g)})$ *then*
      insert $u_i^{(g)}$ *into* $population^{(g+1)}$
    else
      insert $\Theta_i^{(g)}$ *into* $population^{(g+1)}$
    end if
    end for
  $g \leftarrow g + 1$
end while

The trajectory planning strategy can be transformed into an optimization issue with multiple constraints. Firstly, it demands to determine the dimension of the population NP, the generation number g with maximum $g_{\max}$, the dimension real-valued of the individual is equal to the configuration space of the manipulator, the scale factor F, and the crossover factor C_r. Then Individuals in the population are expressed by: $\Theta_i^{(g)} = (\theta_{i,1}^{(g)}, \theta_{i,2}^{(g)}, \ldots, \theta_{i,6}^{(g)})$; $i = 1, 2, \ldots, NP$, represents the design variable of the i-th individual in generation g DE begins by initializing a population of NP to cover as much as possible of the exploration space constrained by the minimum and maximum bounds $\Theta_{\min} = [\theta_{\min,1}, \theta_{\min,2}, \ldots, \theta_{\min,i}, \ldots, \theta_{\min,6}]^T$ and $\Theta_{\max} = [\theta_{\max,1}, \theta_{\max,2}, \ldots, \theta_{\max,i}, \ldots, \theta_{\max,6}]^T$. Hence, the i-th individual may then be initialized as: $\theta_{i,j}^{(1)} = \theta_{\min,j} + H(0, 1)[\theta_{\max,j} - \theta_{\min,j}]$, with $H = rand(0, 1)$

being a uniformly random value between 0 and 1. Henceforward, the mutant strategy is adopted after initialization to generate a donor vector $\nu_i^{(g)} = (\upsilon_{i,1}^{(g)}, \upsilon_{i,2}^{(g)}, \ldots, \upsilon_{i,6}^{(g)})$ by its corresponding target vector $\Theta_i^{(g)}$ [8, 20] have been proposed:

DE/*rand*/1:

$$\nu_i^{(g)} = \Theta_{r_1}^{(g)} + F_i(\Theta_{r_2}^{(g)} - \Theta_{r_3}^{(g)})$$

DE/best/2:

$$\nu_i^{(g)} = \Theta_{best}^{(g)} + F_i(\Theta_{r_1}^{(g)} - \Theta_{r_2}^{(g)}) + F_i(\Theta_{r_3}^{(g)} - \Theta_{r_4}^{(g)})$$

DE/current-to-best/1:

$$\nu_i^{(g)} = \Theta_i^{(g)} + F_i(\Theta_{best}^{(g)} - \Theta_i^{(g)}) + F_i(\Theta_{r_1}^{(g)} - \Theta_{r_2}^{(g)})$$

either-or: this strategy merges two methods to generate the donor vector [26].

$p_f \leftarrow mutation\ \ probability \in [0, 1]$,
$a \leftarrow random\ \ number \in [0, 1]$
if $a < p_f$ *then*
use $DE/rand/1$: $\nu_i^{(g)} = \Theta_{r_1}^{(g)} + F_i(\Theta_{r_2}^{(g)} - \Theta_{r_3}^{(g)})$
else
use $DE/rand/2$:
$\nu_i^{(g)} = \Theta_{r_1}^{(g)} + \mathrm{K}(\Theta_{r_2}^{(g)} - \Theta_{r_1}^{(g)} + \Theta_{r_3}^{(g)} - \Theta_{r_1}^{(g)})$
end if

Where, F_i is the scaling factor within 0 and 1, indices r_1, r_2, r_3 and r_4 are randomly selected integers from the range [1, *NP*], such that $r_1 \neq r_2 \neq r_3 \neq i$. $\Theta_{best}^{(g)}$ is the best individual in the current population, also p_f and a are the mutation probability and random number, respectively. At that point, a crossover between $\nu_i^{(g)}$ and $\Theta_i^{(g)}$ is performed to generate a trial vector $u_i^{(g)} = (\mu_{i,1}^{(g)}, \mu_{i,2}^{(g)}, \ldots, \mu_{i,6}^{(g)})$, two methods were used in this paper, A binomial and Exponential crossover procedures [8]. The binomial crossover provides a trial vector by selecting an element from the donor vector whenever a randomly produced value formed from a uniform distribution is below the crossover rate C_r. Additionally, an element h is randomly taken per iteration to always come from a donor vector as follows:

$$\mu_{i,j}^{(g)} = \begin{cases} \upsilon_{i,j}^{(g)} \ \ if\ i = h\ or\ rand(0, 1) \leqslant C_r, \\ \theta_{i,j}^{(g)} \ \ otherwise. \end{cases}$$

Exponential crossover tries to exploit relationships between adjacent elements. It works by choosing a random starting element and selecting the next L consecutive elements in a circular manner from the donor vector. The number of elements L is calculated as follows:

Algorithm 2: Exponential crossover

$L \leftarrow 0$
repeat
$L \leftarrow 0$
until $rand(0,1) > C_r$ *or* $L > D$

After crossover, the objective function as explained in Eq. 1 is evaluated for the trial vector $u_i^{(g)}$. According to the greedy selection only, as shown in algorithm 1. Afterward, the better of $u_i^{(g)}$ and $\Theta_i^{(g)}$ will be picked to remain into the next generation.

4.2. Modified Shuffled Frog-Leaping Algorithm

The shuffled frog-leaping algorithm (SFLA) was developed by Eusuff and Lansey in 2003 [21]. It is a member of the Memetic algorithm family, a particular kind of a meta-heuristic optimization approach and a type of evolutionary algorithms which based on population. It is inspired by the memetic evolution of frogs exploring food in a lake, which consolidates the benefits of the genetic-based MA and the social behavior-based particle swarm optimization [21]. In general, the SFLA incorporates two alternating processes: a local exploration in the sub-memeplex and global information exchange among all memeplexes.

The optimization achievement of the SFLA basically relies on two facts, the first one is the evolution process on each memeplex which embraces different cultures of frogs, this culture stimulates fitness values and the evolution process serves as a local search within memeplex analogous to PSO algorithm which imitates the social behavior of the leaping action of frogs searching for food. Additionally, the extra fact is an idea held within each frog which can be influenced by the ideas of other frogs from other memeplexes throughout the shuffling rule, this animates the cooperation process which it implies an adaptation idea and improves the success rate of discovering the solution in optimization puzzle. In this process, a modification was applied to the frog-leaping action that it enhances the exploration manner in the space [22], [23]. Moreover, the randomization strategy in the evolution process proffers the algorithm the ability to discover the local best solution within search space stochastically in addition to the communication process that possibly finds a global optimum solution in shorter

time. The local search and the shuffling processes continue until defined convergence criteria are satisfied. The pseudocode of the algorithm was presented in Algorithm 3.

Algorithm 3: The pseudo-code of the Shuffled Frog-Leaping Algorithm

Initialization:
Population ← {Θ_1, Θ_2, ..., Θ_i, ..., Θ_{NP}};
m ← *number of memeplexes;*
n ← *quantity of frogs in each memeplex;*
l ← 1, *iN*
while (convergence criteria is satisfied Or ubtil met iN) do
  Rank Step: Evaluate each frog Θ_i using a fitness function;
  Partition Step:
  Construct an array *U* of frogs and their fitness's values;
  Sort the array *U* in descending order based on the fitness column;
  Construct (Y^k; k = 1, ..., m) memeplexes each including n frogs;
  Evaluation Step:
  for l ← 1, *iM do*
    for k ← 1, *m do*
      Determine the worst and best frogs based on their fitness's values;
      Improve the worst frog position using a leaping distance;
      end for
  end for
  Shuffle Memeplexes Step: combine the evolved memeplexes;
  Check Convergence: Update the population best frog's position Θ_g;
  $l \leftarrow l + 1$;
end while

The MSFLA meta-heuristic strategy is summarized in the following steps:

a. Initialization step, construct the population *NP* of frogs randomly similar to the first step in DE algorithm, then Select *m*, and *n*, where *m* is the number of memeplexes and *n* is the number of frogs in each memeplex. Therefore, the total amount of frogs *NP* can be calculated as $NP = mn$, additionally, the *i*-th frog is expressed as a vector with the dimension is equal to the configuration space as follows $\Theta_i = (\theta_{i,1}, \theta_{i,2}, \ldots, \theta_{i,6})$; $i = 1, 2, \ldots, NP$.

b. Rank step, compute the performance value f_i for each frog Θ_i. Sort the *NP* frogs in a descending order according to their fitness. Save them in an array $U = \{f_i, \Theta_i;\ i = 1, 2, ..., NP\}$, so that $i = 1$ denotes the frog with the best performance value and could save it as a Θ_g in each iteration while algorithm running.

c. Partition Step, partition array *U* into *m* memeplexes Y_1, Y_2, ..., Y_m, each including *n* frogs, such that $Y^k = [\Theta_i^k, f_i^k | \Theta_i^k = \Theta_{(k+m(i-1))}, f_i^k = f_{(k+m(i-1))}, i = 1, \ldots, n]; k = 1, \ldots, m$. In this process, the first frog goes to the first memeplex, the second frog goes to the second memeplex, frog *m* goes to the *m*-th memeplex, and frog $m + 1$ goes back to the first memeplex, etc.

d. Memetic Evaluation step, evolve each memeplex Y^k; k = 1, ..., m according to the frog-leaping algorithm as follow. Within each memeplex, the frogs with the best and the worst fitness values are defined as Θ_b and Θ_w, respectively. Furthermore, the frog with the global best fitness is defined as Θ_g. Next, an improvement process is applied to only the frog with the worst fitness in each cycle. Hence, the position of the frog with the worst fitness is modified which emulates the leaping process as follows: leaping distance $D = C_L rand(0, 1)[\Theta_b - \Theta_w]$, then new position $\Theta_w = \Theta_w + D$; $D \in]-D_{\max}, D_{\max}[$. Where, *rand*(0, 1) is a random number between 0 and 1, $D_{\max}$ is the maximum allowed change in a frog's position and C_L is the modification of the algorithm which it is a constant indicates the amount of frog-leaping in each memeplex. The evaluation process for all memeplexes is repeated by an adaptable number of iterations called *iM* until no improvement becomes possible.

e. Shuffle Memeplexes Step, shuffle frogs and replace all memeplexes Y^k; k = 1, ..., m into *U*, such that $U = \{f_i, \Theta_i;\ i = 1, 2, ..., NP\}$ similar to initialization phase, afterward sort *U* in order of decreasing performance value, Update the population best frog's position Θ_g.

f. Check the convergence criteria if satisfied then stop otherwise return to the partition step and continue for a specific quantity of iterations which called *iN*, finally after each iteration the first frog in the sorted list represents a global solution. The number of iteration *iM* specifies the depth of search within memeplexes while the *iN* governs the solution producing process.

5. Simulation Results and Discussions

In this work, we solve inverse kinematics of the redundant manipulator with six joints to follow a destination pose. The manipulator's joints correspond to the variable θ_j, $J = 1, 2, ..., 6$ had been constrained by a Table 2. The DH table is presented in Table 3. In the inverse kinematics experiments, the desired end-effector pose for the arm-part of the manipulator was determined as a variable $G = (P_G, O_G) = (x, y, z, roll, pitch, yaw) = (-20, 3, 40, 0, 10, 15)$. Moreover, the

Table 4

Setting of the DE Algorithm

Mutation Method	Random
Scale Factor	0.9
Crossover Method	Uniform
Crossover Probability	0.95

parameters of the objective function were adjusted as follows $\varepsilon = 1 - \beta = 0.7$ so there is a balance between position and orientation to be optimized. In case of DE algorithm, Table 4 shows DE settings while Table 5 presents the results of utilizing DE for some scenarios.

As presented in Table 6, it is obvious that the execution time depends on the size of the population and the iterations, respectively. Further, the population size achieves the diversity feature which let the algorithm explores more solutions in the workspace while the high iteration gives a solution much closer to the target. Hereafter, a Fig. 2 presents the values of the objective function, while the Fig. 4 and Fig. 5 illustrate the position and orientation of end-effector for the manipulator after applying the solutions to validate IK solver. Forthwith, the Fig. 3 (see the 3rd side of cover) demonstrates the configuration joints for reaching the setpoint G, which shows multiple solutions after each new iteration be-

Table 5

Inverse Kinematics Results of Differential Evolution Algorithm

Test No.	Population	Iterations	$J(\Theta)$	Total Error	Execution Time [*ms*]	Reaching Target (*x*, *y*, *z*, *roll*, *pitch*, *yaw*)
1	6	250	4.60464	−8.22166	247	(−20.0619, 3.00659, 40.0358, 3.74, −4.49369, 17.5544)
2	6	500	1.53162	1.78975	594	(−19.56, 2.64638, 39.8767, 0.465805, 13.2258, 13.135)
3	8	600	1.19735e−5	8.26589e−6	861	(−20, 3, 40, −8.25797e−6, 10, 15)
4	8	800	3.60207e−7	6.13326e−7	1125	(−20, 3, 40, 2.59552e−8, 10, 15)
5	10	500	0.000230295	0.000677389	986	(−19.99, 3.0001, 40.00001, 4.21087e−5, 10.004, 15)
6	10	750	1.55363e−7	1.13202e−7	1301	(−20, 3, 40, 1.13202e−7, 10, 15)
7	10	1000	4.53651e−9	−3.13365e−9	1917	(−20, 3, 40, −1.40416e−9, 10, 15)
8	12	250	0.22408	−0.001729	567	(−19.8094, 3.06099, 40.1352, 0.0528143, 10.1526, 15.087)
9	12	400	0.0004035	0.000107144	835	(−20.0001, 3.00003, 40, −0.0006738, 9.999932, 14.9998)
10	12	500	3.42549e−5	0.000107144	1146	(−20, 3, 4, 3.25415e−5, 10, 15)
11	20	500	0.000828201	0.00156291	1736	(−19.9994, 3.00002, 40.0002, 0.000503176, 10.0009, 14.999)
12	20	750	1.28825e−6	−4.33271e−6	2628	(−20, 3, 40, −3.29831e−6, 10, 15)
13	30	500	0.00107932	−0.00085163	2614	(−20.0004, 2.99975, 39.999, 0.000577463, 9.999, 15.0007)
14	30	1000	9.20132e−7	3.09732e−8	5255	(−20, 3, 40, 3.978432e−8, 10, 15)

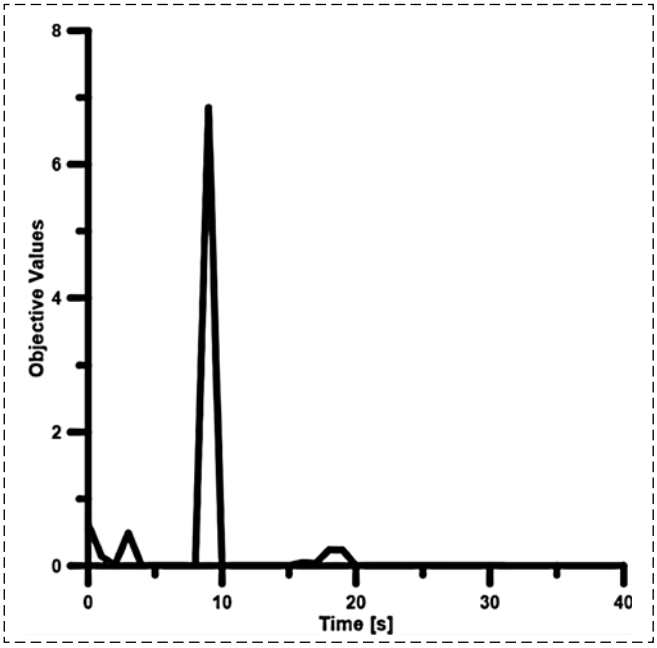

Fig. 2. Displays the values of the objective function after applying IK-DE solver

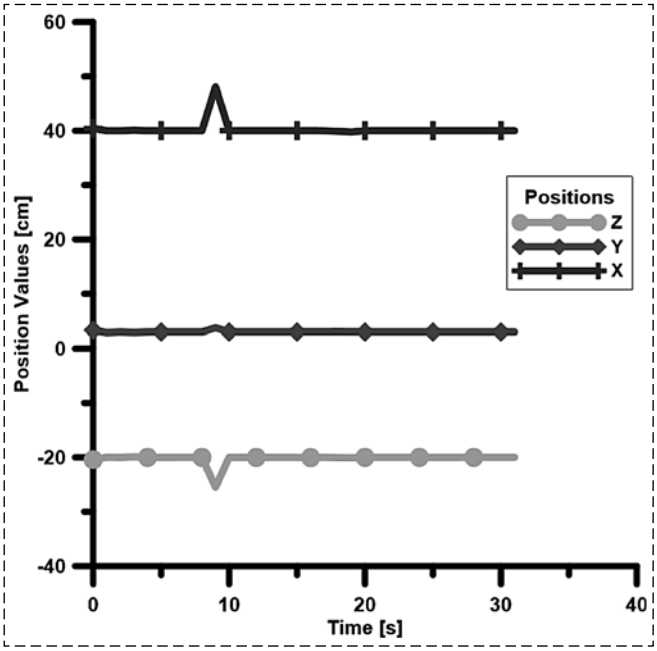

Fig. 4. Illustrates the position of end-effector for the manipulator after applying the solutions to validate IK-DE solver

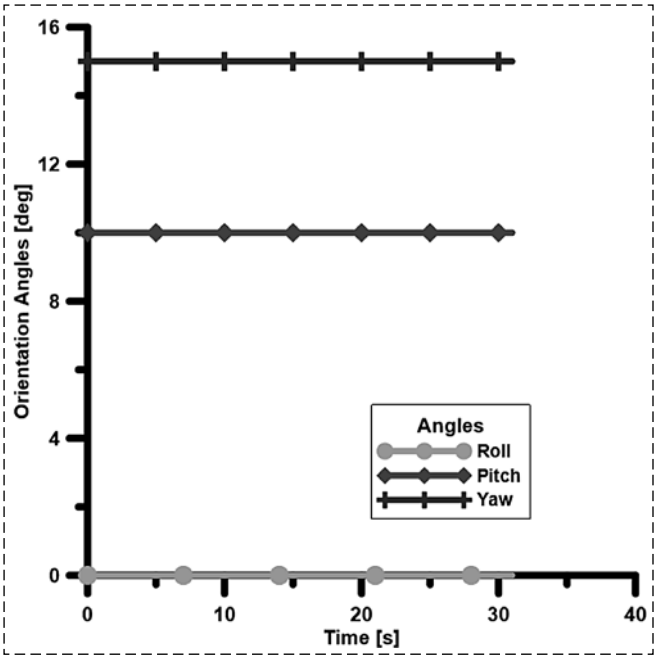

Fig. 5. Illustrates the orientation of end-effector for the manipulator after applying the solutions to validate IK-DE solver

cause of the redundant nature of our manipulator, in addition, each new solution is considered as a global solution within its iteration, new high iteration grants algorithm an ability to explore a new global solution. therefore, it's important to alter the settings of DE to conclude one solution based on the objective function in shorter time.

By analyzing the results in the Table 5 and Fig. 3, it was explicit that the test number 9 is the proper solution with a convergence time equal to 835 ms and a total error equal to −0.001729 taking into consideration that the adaptation of DE parameters nearby setting of this result may improve the solution to be more fitting but longer convergence time.

In case of MSFLA, the parameters of the algorithm were introduced in the Table 6 additionally, the summary of the results of utilizing the algorithm for multiple scenarios was represented in the Table 7.

As presented in the results, the amounts of execution time are longer than those in the experimental results of DE algorithm. The population of the DE algorithm is created randomly and the size of the population should be selected based on the problem and dimension of the workspace in order to cover all the space and consequently obtain the global solution quickly. Besides, the maximum number of iteration depends on the accuracy required on the problem.

Table 6

Setting of the MSFL Algorithm

m	Number of memeplexes	3
n	Number of frogs within memeplexes	NP/m
C_L	Amount of Leaping	1.3

Table 7

Inverse Kinematics Results of MSFL Algorithm

Tests	Population	Iterations		$J(\Theta)$	Total Error	Execution Time [*ms*]	Reaching Target (x, y, z, *roll*, *pitch*, *yaw*)
		iN	*iM*				
1	20	30	10	11.618	29.7156	729	(−15.7365, 5.43, 52.57, 6.63, 12.66, 16.164)
2	30	30	10	7.6614	12.0878	1045	(−21.1832, 2.91528, 50.77, −0.201742.10.0158, 14.8583)
3	40	30	15	10.5382	19.2097	1685	(−25.08, 8.56, 46.818, −6.2, 9.6, 5.4251)
4	40	40	30	18.4625	18.4625	4526	(−25.2365, 8.34134, 47.591, −2.537.8.6258.14.1301)
5	60	40	30	8.2925	8.2925	6645	(−24.4625, 0.04208, 44.589, 1.655, 11.1161, 14.054)
6	80	50	40	11.0236	11.0236	13 540	(−26.9977, 3.59399, 42.871, −0.0686, 9.816, 15.6762)
7	100	60	60	29.7744	29.7744	24 191	(−20.0331, 30.03867, 39.9706, −7.709, 3.6786, −0.7192)
8	130	70	60	0.151062	0.648529	46 282	(−20.09, 2.988, 40.004, 0.20727, 9.642, 14.894)
9	170	60	50	0.616846	2.16791	40 459	(−20.1505, 2.8357, 40.0911, 0.8872, 10.3554, 16.149)
10	200	90	40	0.113833	0.297961	57 362	(−19.9268, 3.00718, 39.98, −0.1344, 10.1049, 14.894)
11	200	100	60	0.0728863	0.378871	92 779	(−20.0018, 2.99737, 39.9906, −0.13745, 10.1506,14.9178)
12	200	120	80	2.76717	5.81648	150 246	(−20.4796, 4.15287, 40.4894, 0.80744, 4.08227, 13.7867)
13	200	200	100	2.67129	1.93396	318 481	(−19.2649, 2.23482, 41.9365, −0.978994, 10.8776, 11.4859)
14	250	90	40	0.003134	0.0165418	69 818	(−19.9995, 3.00023, 39.9998, 0.00495, 10.0061, 14.995)
15	250	140	80	1.26635	6.55357	215 027	(−20, 3, 10, −7.01976e−10, 10, 15)
16	250	140	100	4.64762e-9	1.05135e-8	260 325	(−20.0982, 2.9671, 40.0008, −1.68742, 6.68997, 13.5741)
17	300	140	80	1.01476e-9	3.3374e-9	255 989	(−20, 3, 40, 1.16487e−9, 10, 15)
18	500	90	40	5.4912e-10	9.9601e-10	136 888	(−20, 3, 40, −1.66261e−11, 10, 15)
19	500	200	100	3.02894e-15	1.5664e-14	68 1646	(−20, 3, 40, 3.22962e−15, 10, 15)
20	1000	30	45	0.0968137	0.0255	95197	(−20.0305, 3.0454, 40.0451, −0.0037, 10.957, 14.8752)

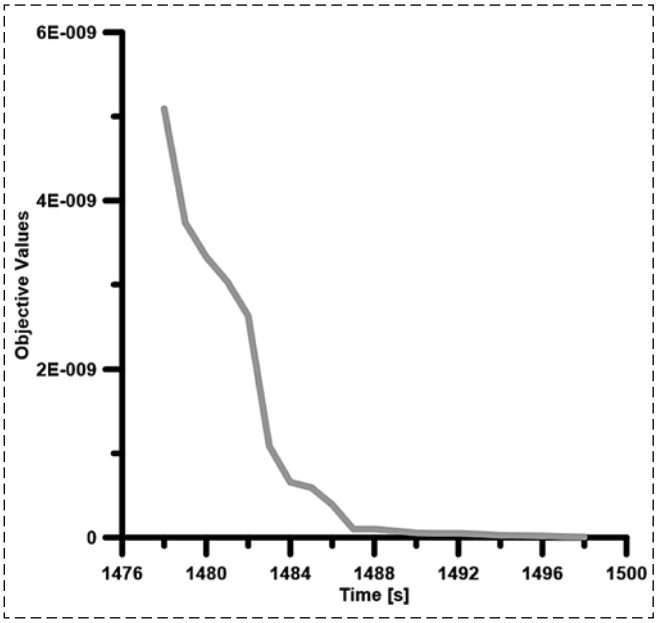

Fig. 6. Displays the values of the objective function after applying IK-MSFLA solver

Hereafter, the Fig. 6 displays the values of the objective function, while the Fig. 7 and Fig. 8 (see the 3rd side of cover) represent the position and orientation of end-effector for the manipulator after applying the solutions to validate IK solver. Forthwith, the Fig. 9 (see the 3rd side of cover) shows the configuration joints for reaching the setpoint G.

6. Conclusion

The design and kinematics of a human-size aerial manipulation robot have been introduced. this design has been separated into two parts: arm-part and hand part in order to be easier in the navigation process and control system. the purpose of this study is to observe a perception of actuating the manipulator so fast; to reach a target point in the 3D workspace, robotically investigate a kinematics solution. therefore, in this paper, we have presented a methodology to solve the inverse kinematics of a 6-DOF lightweight manipulator based on evolutionary algorithms. Furthermore, the metaheuristic algorithms included in the simulations were the differential evolution (DE) and modified shuffled frog-leaping (MSFL). These algorithms were designated as multi-objective, multi-dimensional, stochastic and diversity methods. The simulation and experiments were conducted to prove the effectiveness of each approach. Consequently, the DE had the best performance over SFLA with the high success rate and the fast convergence as well, the simulation was carried out through PC with following specifications (Core i7, Q740, 1.73GHz). Subsequently, all the optimization techniques of population-based algorithms depend on the concept of generating a population randomly then relocate or migrate individuals in the space to obtain the local/global solution. besides, it is not arduous to promote those two algorithms and proffer them an adaptive feature as we fulfilled with MSFLA in this research to deliver more stabilization plus little run-time. Finally, the aim of this study was acknowledged as one from multiple objectives of our project, which are controlling the aerial manipulator integrated into an aerial vehicle, in the space additionally, studying of the disturbances effects on the vehicle and stabilizing the motion of the manipulator.

References

1. **Buss S. R.** (2004). Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, *17* (1—19), 16.

2. **Dulęba I., & Opałka M.** (2013). A comparison of Jacobian-based methods of inverse kinematics for serial robot manipulators, *International Journal of Applied Mathematics and Computer Science*, 23(2), 373—382.

3. **Wang X., Zhang D., Zhao C.** (2017). The inverse kinematics of a 7R 6-degree-of-freedom robot with non-spherical wrist, *Advances in Mechanical Engineering*, *9*(8), 1687814017714985.

4. **Ananthanarayanan H., & Ordóñez R.** (2015). Real-time Inverse Kinematics of (2n + 1) DOF hyper-redundant manipulator arm via a combined numerical and analytical approach, *Mechanism and Machine Theory*, *91*, 209—226.

5. **Tolani D., Badler N. I.** (1996). Real-time inverse kinematics of the human arm, *Presence: Teleoperators & Virtual Environments*, 5(4), 393—401.

6. **Toshani H., & Farrokhi M.** (2014). Real-time inverse kinematics of redundant manipulators using neural networks and quadratic programming: a Lyapunov-based approach, *Robotics and Autonomous Systems*, *62*(6), 766—781.

7. **Reiter A., Müller A., Gattringer H.** (2016, October). Inverse kinematics in minimum-time trajectory planning for kinematically redundant manipulators, In *Industrial Electronics Society, IECON 2016 — 42nd Annual Conference of the IEEE* (pp. 6873—6878). IEEE.

8. **Geitle M.** (2017). *Improving differential evolution using inductive programming* (Master's thesis).

9. **Bodily D. M., Allen T. F., Killpack M. D.** (2017, May). Motion planning for mobile robots using inverse kinematics branching, *In Robotics and Automation (ICRA), 2017 IEEE International Conference on* (pp. 5043—5050). IEEE.

10. **Ayyıldız M., Çetinkaya K.** (2016). Comparison of four different heuristic optimization algorithms for the inverse kinematics solution of a real 4-DOF serial robot manipulator, *Neural Computing and Applications*, *27*(4), 825—836.

11. **Rokbani N., Alimi A. M.** (2013). Inverse kinematics using particle swarm optimization, a statistical analysis, *Procedia Engineering*, *64*, 1602—1611.

12. **Collinsm T. J., Shen W. M.** (2017, April). Particle swarm optimization for high-DOF inverse kinematics, *In Control, Automation and Robotics (ICCAR), 2017 3rd International Conference on* (pp. 1—6). IEEE.

13. **Mao B., Xie Z., Wang Y., Handroos H., Wu H., Shi S.** (2017). A hybrid differential evolution and particle swarm optimization algorithm for numerical kinematics solution of remote maintenance manipulators, *Fusion Engineering and Design*, *124*, 587—590.

14. **Kachitvichyanukul V.** (2012). Comparison of three evolutionary algorithms: GA, PSO, and DE, *Industrial Engineering and Management Systems*, *11*(3), 215—223.

15. **López-Franco C., Hernández-Barragán J., Alanis A. Y., Arana-Daniel N., López-Franco M.** (2018). Inverse kinematics of mobile manipulators based on differential evolution, *International Journal of Advanced Robotic Systems*, *15*(1), 1729881417752738.

16. **Shiakolas P. S., Koladiya D., Kebrle J.** (2005). On the optimum synthesis of six-bar linkages using differential evolution and the geometric centroid of precision positions technique, *Mechanism and Machine Theory*, *40*(3), 319—335.

17. **Juang C. F., Chen Y. H., Jhan Y. H.** (2015). Wall-following control of a hexapod robot using a data-driven fuzzy controller learned through differential evolution, *IEEE Transactions on Industrial electronics*, *62*(1), 611—619.

18. **Pierezan J., Freire R. Z., Weihmann L., Reynoso-Meza G., dos Santos Coelho L.** (2017). Static force capability optimization of humanoids robots based on modified self-adaptive differential evolution, *Computers & Operations Research*, *84*, 205—215.

19. **Ngoc Son N., Anh H. P. H., Thanh Nam N.** (2016). Robot manipulator identification based on adaptive multiple-input and multiple-output neural model optimized by advanced differential evolution algorithm, *International Journal of Advanced Robotic Systems*, *14*(1), 1729881416677695.

20. **Wang M., Luo J., Fang J., Yuan J.** (2018). Optimal Trajectory Planning of Free-Floating Space Manipulator Using Differential Evolution Algorithm, *Advances in Space Research*.

21. **Eusuff M., Lansey K., Pasha F.** (2006). Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization, *Engineering optimization*, *38*(2), 129—154.

22. **Li X., Luo J., Chen M. R., Wang N.** (2012). An improved shuffled frog-leaping algorithm with extremal optimisation for continuous optimization, *Information Sciences*, *192*, 143—151.

23. **Samuel G. G., Rajan C. C. A.** (2014). A modified shuffled frog leaping algorithm for long-term generation maintenance scheduling, *In Proceedings of the Third International Conference on Soft Computing for Problem Solving* (pp. 11—24). Springer, New Delhi.

24. **Afzalan E., Taghikhani M. A., Sedighizadeh M.** (2012). Optimal placement and sizing of DG in radial distribution networks using SFLA, *International Journal of Energy Engineering*, 2(3), 73—77.

25. **Ibrahim I. N.** (2018). Ultra Light-Weight Robotic Manipulator. *Bulletin of Kalashnikov ISTU*, 2018, vol. 21, no. 1, pp. 12—18, DOI: 10.22213/2413-1172-2018-1-12-18 (in Russian).

26. **Simon D.** (2013). *Evolutionary optimization algorithms*. John Wiley & Sons.