International Journal of Dynamics and Control (2023) 11:1895–1913
https://doi.org/10.1007/s40435-022-01091-4

# Optimizing an adaptive fuzzy logic controller of a 3-DOF helicopter with a modified PSO algorithm

**Shokoufeh Naderi**[1] · **Maude J. Blondin**[1] · **Behrooz Rezaie**[2]

Received: 17 April 2022 / Revised: 20 October 2022 / Accepted: 28 November 2022 / Published online: 11 December 2022


**Abstract**
This paper investigates the controller optimization for a helicopter system with three degrees of freedom (3-DOF). The system is extensively nonlinear and highly sensitive to the controller's parameters, making it a real challenge to study these parameters' effects on the controller's performance. We combined fuzzy logic with adaptive control theory to control the system and used metaheuristic algorithms to determine these parameters. Then, we compare the results with the controller optimized through the standard PSO and PID controller. The results indicate the high ability of MPSO to perform the global search and to find a reasonable search space. The proposed method's effectiveness and robustness properties are shown through computer simulations, while the system is subject to uncertainties and disturbance. We also prove the efficiency of the MPSO algorithm by comparing it with the standard PSO and six other well-known metaheuristic algorithms and analyzing the results by statistical tests.

**Keywords** Modified particle swarm optimization (MPSO) · 3-DOF helicopter · Adaptive fuzzy logic controller

## 1 Introduction

During the last decades, unmanned aerial vehicles (UAVs) have widely been developed and used due to technological advancements [1]. They have applications in military and civil fields, such as traffic condition assessment and forest fire monitoring, to name a few [2]. They possess essential features like hovering and vertical take-off, which increase their applicability. However, they are highly nonlinear and subject to disturbances and uncertainties. The nonlinearity and susceptibility to disturbances demand a control structure that can reject external disturbances, such as wind [3].

Many classic, adaptive, and robust control strategies have been proposed to tackle this control problem. In [4], for example, a proportional derivative (PD) and a proportional integral derivative (PID) attitude and position controllers are designed to stabilize a rotorcraft in free flight. In [5], active vibration control is presented for a helicopter rotor blade that uses a linear quadratic regulator (LQR) to reduce vibrations. In [6], designing an adaptive model predictive control has been addressed for a 2-DOF helicopter in the presence of uncertainties and constraints.

These control strategies perform well in the presence of parametric uncertainties. However, they may underperform in real-life applications with uncertainties, such as external disturbances, noises, and unmodeled dynamics of the machine, mainly because the methods are developed and based on the exact mathematical model of the system [7,8].

On the other hand, intelligent control techniques can adapt themselves in the presence of uncertainties. These approaches have various structures, including neural networks, fuzzy systems, and machine learning models [9]. The feature of not being dependent on a precise mathematical model has led to many publications on their combination with conventional control strategies for UAVs. In [10], the design and experimental validation of an adaptive fuzzy PID controller are presented for a 3-DOF helicopter. The interval

✉ Shokoufeh Naderi
shokoufeh.naderi@usherbrooke.ca

Maude J. Blondin
maude.blondin2@usherbrooke.ca

Behrooz Rezaie
brezaie@nit.ac.ir

[1] Département Génie Electrique & Génie Informatique, Université de Sherbrooke, Sherbrooke, QC, Canada

[2] Faculty of Electrical and Computer Engineering, Babol Noshirvani University of Technology, Babol, Mazandaran, Iran

type 2 fuzzy logic is combined with adaptive control theory to control a 3-DOF helicopter in [7], which is robust to various types of uncertainties. However, using higher types of fuzzy systems increases the computational loads. Another example is the publication [11] that proposes designing data-driven attitude controllers for a 3-DOF helicopter under multiple constraints, in which the reinforcement learning technique updates the controller. An adaptive neural network backstepping controller is designed in [12] to compensate for unmodeled dynamics and external disturbances.

The problem with these intelligent controls is that they usually have many parameters to tune. Tuning is a complex task and almost impossible to do by trial and error. To overcome that, researchers usually implement metaheuristic algorithms [13]. These algorithms have recently attracted increased interest for this purpose thanks to their high convergence speed and high accuracy. For instance, in [14] the PSO algorithm is implemented to optimize the weighting matrices of the LQR controller to design an optimal flight control for a 2-DOF helicopter. The PSO is also used in [15] to tune the values of design parameters of an adaptive super-twisting sliding mode controller for a two-axis helicopter in the presence of model uncertainties. The publication [16] employs a genetic approach for real-time identification and control of a helicopter and [17] develops a chaotic artificial bee colony algorithm to identify a small-scale helicopter in hover conditions.

In some papers, the researchers have improved the performance of these algorithms by investigating the accuracy, speed, and convergence rate. For example, in [18], an improved genetic algorithm (GA) is used to optimize the initial fuzzy rules of an adaptive fuzzy PID controller for a micro-unmanned helicopter. However, in large-scale problems where the search space is not clearly specified, the algorithm may get trapped into local optimal solutions [9]. Therefore, there is a need for algorithms that offers fast convergence and high accuracy while simultaneously being capable of modifying the search space.

In this paper, we employ a modified particle swarm optimization (MPSO) algorithm to optimize the parameters of an adaptive fuzzy controller for a 3-DOF helicopter. The helicopter system is highly nonlinear, and the controller's values significantly affect its performance. Therefore, optimizing these parameters can lead to more effective results. We chose the MPSO algorithm because it shows a satisfactory performance when dealing with many parameters to optimize. Furthermore, it eliminates the need for specifying the exact boundaries of the search space, as the search space of the particles is modified based on the value of each particle. This results in searching in a reasonable space, which not only does it shorten the optimization time but also results in finding a better solution by avoiding getting trapped in local optimums. Additionally, the algorithm considers an elimination phase, meaning some poor particles are substituted by new particles in the new search space. These modifications have improved the convergence rate and accuracy of the algorithm. Indeed, simulation results in [13] prove the superiority of this algorithm over several metaheuristic algorithms, including improved GA, imperialist competitive algorithms, and artificial bee colony. This paper shows the effectiveness of using the MPSO algorithm to optimize the controller's parameters for the 3-DOF helicopter through simulations when the system is subject to uncertainties and disturbance. Also, we compare the performance of the MPSO algorithm with the standard PSO algorithm and some other metaheuristic algorithms.

In short, this paper contains the following contributions:

- Optimizing an adaptive type 1 fuzzy logic controller for a 3-DOF helicopter model through an MPSO algorithm.
- Comparing the performance of the proposed controller with the PID controller and with the controller optimized through the PSO algorithm.
- Analyzing the robustness of the proposed controller in the presence of uncertainty and disturbance.
- Comparing the performance of the MPSO algorithm and some other well-known metaheuristic algorithms for the task of optimizing the adaptive type 1 fuzzy logic controller of the helicopter model along with analyzing their results.

The adaptive type 2 fuzzy logic controller for the 3-DOF helicopter is presented in [7]. Since the membership functions are fixed in the conventional type 1 fuzzy systems, which may lead to weak performance in the presence of uncertainties, the authors of [7] proposed using type 2 fuzzy controllers for the highly nonlinear helicopter system. However, they did not present the results of the type 1 fuzzy logic controller. In our paper, we use an adaptive type 1 fuzzy controller. We show that, for the considered task, the type 1 fuzzy logic controller performs very well under uncertainties and disturbance, eliminating the need for employing higher types of fuzzy systems. As a result, the computational load is lowered.

The rest of this paper is organized as follows: Sect. 2 presents the dynamic model of the helicopter. Section 3 provides the adaptive fuzzy logic controller. Section 4 describes the basic concepts of the MPSO algorithm. Simulation results and MPSO algorithm efficiency analysis are presented in Sect. 5, and Sect. 6 provides the concluding remarks.

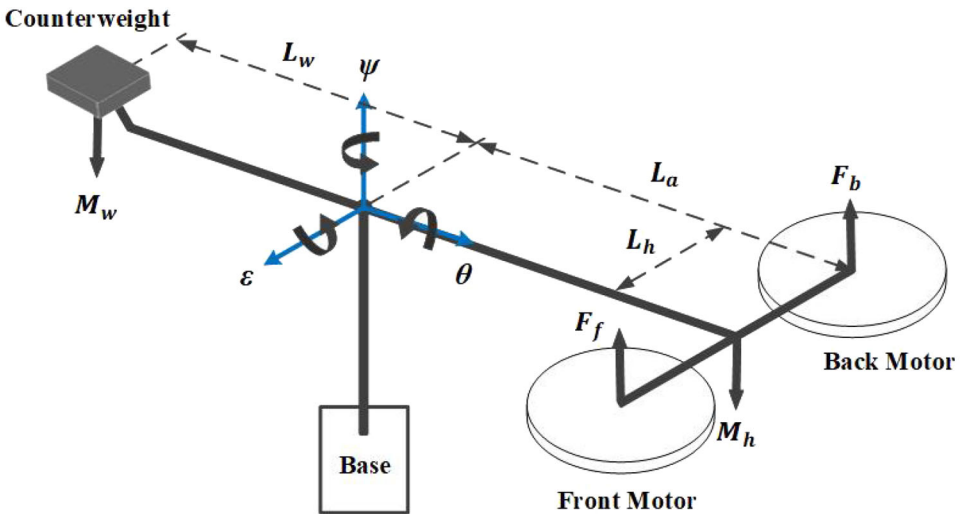



**Fig. 1** The 3-DOF helicopter

**Table 1** Values of the 3-DOF helicopter model [3]

| Model's parameter | Description | Value |
|---|---|---|
| $M_h$ | Mass of the helicopter | 1.426 kg |
| $M_\omega$ | Mass of the counterweight | 1.870 kg |
| $L_a$ | Distance between the roll axis and the center of mass | 0.660 m |
| $L_\omega$ | Distance between the lifting axis and the counterweight | 0.470 m |
| $L_h$ | Distance between the pitch axis and each motor | 0.178 m |
| $J_\epsilon$ | Moment of inertia about roll axis | 1.0348 kg.m$^2$ |
| $J_\theta$ | Moment of inertia about pitch axis | 0.0451 kg.m$^2$ |
| $J_\epsilon$ | Moment of inertia about yaw axis | 1.0348 kg.m$^2$ |
| $g$ | Gravitational constant | 9.81 m.s$^2$ |

## 2 Mathematical model of the 3-DOF helicopter system

The schematic of the 3-DOF helicopter system is shown in Fig. 1. It has two DC motors mounted at the two ends of a frame. The DC motors drive two propellers which generate the lift forces $F_f$ and $F_b$. These forces control the attitude of the helicopter. The system studied here is underactuated, i.e., it rotates freely about three axes—the pitch axis $\theta$, the roll axis $\epsilon$, and the yaw axis $\psi$—with only two control forces. The mathematical model of a 3-DOF helicopter is given by the following differential equations [3]:

$$\begin{aligned} J_\epsilon \ddot{\epsilon} &= g(M_h L_a - M_\omega L_\omega)\cos\epsilon + L_a \cos\theta\, u_1 \\ J_\theta \ddot{\theta} &= L_h u_2 \\ J_\psi \ddot{\psi} &= L_a \cos\epsilon\, sin\theta\, u_1 \end{aligned} \tag{1}$$

where $u_1 = F_f + F_b$, $u_2 = F_f - F_b$. The pitch $\theta$, roll $\epsilon$, and yaw $\psi$ angles give the position of the helicopter body. Values of the model parameters with their description are given in Table 1. As it is shown in Fig. 1, the roll ($\epsilon$) and yaw ($\psi$) angles are perpendiculars. The intersection of the three axes is considered the origin of the coordinate frame. The helicopter model is defined as follows:

- Pitch angle $\theta$ is defined as $-45° \leq \theta \leq +45°$.
- Roll angle $\epsilon$ is defined as $-27.5° \leq \epsilon \leq +30°$.

These constraints on the pitch and roll angles will appear as saturation functions in the program.

## 3 Adaptive fuzzy logic controller

### 3.1 Design of the controller

As shown in (1), the roll and yaw dynamics depend on the pitch value and are actuated by $u_1$, while the pitch dynamic is actuated by $u_2$. This means that only two input forces control the system, i.e., the system is underactuated. We must first define the desired roll and yaw trajectories to control the system with only two inputs. Next, the desired pitch trajectory is obtained based on an internal loop. Figure 2 shows the block diagram of the control structure of the helicopter system. As the figure shows, the controller combines backstepping and

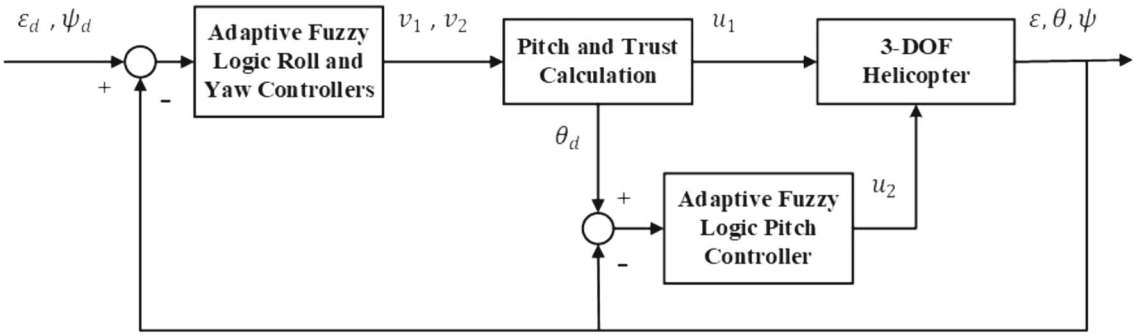



**Fig. 2** Block diagram of the control structure

fuzzy controllers. This structure has been previously used to control nonlinear systems [19]. The reason behind using this structure is to make the controller more robust to disturbances and uncertainties [20].

In the control structure, two virtual inputs $v_1$ and $v_2$ are defined to achieve decoupling [7]:

$$\begin{aligned} v_1 &= \cos\theta\, u_1 \\ v_2 &= \cos\epsilon\, \sin\theta\, u_1 \end{aligned} \tag{2}$$

Replacing the virtual inputs $v_1$ and $v_2$ in (1) transforms the nonlinear system (1) into the decoupled system (3), in which $\epsilon$, $\theta$, and $\psi$ are controlled by $v_1$, $u_2$, and $v_2$, respectively:

$$\begin{aligned} J_\epsilon \ddot{\epsilon} &= g(M_h L_a - M_\omega L_\omega) cos\epsilon + L_a v_1 \\ J_\theta \ddot{\theta} &= L_h u_2 \\ J_\psi \ddot{\psi} &= L_a v_2 \end{aligned} \tag{3}$$

Notice that $v_1$ and $v_2$ in (2) are not independent but are linked through $u_1$.
From (2), we have

$$u_1^2 = \frac{v_1^2}{\cos^2\epsilon} + v_2^2 \tag{4}$$

Then, it follows that

$$u_1 = S\sqrt{\frac{v_1^2}{\cos^2\epsilon} + v_2^2}\,, \quad S = \begin{cases} \text{sign}(v_2) & v_2 \neq 0 \\ 0 & v_2 = 0 \end{cases} \tag{5}$$

(2) also satisfies

$$\tan\theta = \frac{v_2}{\cos\epsilon\, v_1} \tag{6}$$

As we said, a way to control the system by only two inputs is to first define the desired roll and yaw trajectories and, then, to obtain the desired pitch trajectory. From (6), this desired pitch trajectory is obtained as follows:

$$\theta_d = \tan^{-1}\frac{v_2}{\cos\epsilon\, v_1} \tag{7}$$

So the control strategy is as follows:
First, the control inputs $v_1$ and $v_2$ are computed using the tracking error between $\epsilon$ and $\psi$ and their desired trajectories $\epsilon_d$ and $\psi_d$. Then, the pitch desired trajectory $\theta_d$ and $u_1$ are computed using (7) and (5). Once the $\theta_d$ is computed, the pitch controller allows to design $u_2$.

Given the desired trajectories, the errors are defined as follows:

$$\begin{aligned} e_\epsilon &= \epsilon - \epsilon_d \\ e_\theta &= \theta - \theta_d \\ e_\psi &= \psi - \psi_d \end{aligned} \tag{8}$$

Each adaptive fuzzy logic controller has to adjust its weight to track these errors to zero. The errors and their derivatives are considered as inputs of the controllers. So, each adaptive fuzzy controller has two inputs and an output. Using the centroid defuzzification method, the output of the controller is as follows [21]:

$$Y(x) = \frac{\sum_{i=1}^{n} f^i y^i}{\sum_{i=1}^{n} f^i}, \tag{9}$$

where $n$ is the number of fuzzy logic rules and $y^i$ is the output singleton number. $f^i$ is expressed by the following equation[1]:

$$f^i = \prod_{j=1}^{n} \mu_{A_j^i}(x_j), \tag{10}$$

where $\mu_{A_j^i}$ is the value of the membership function for the fuzzy variable $x_j$.

[1] The mathematical expression of $\mu_{A_j^i}(x_j)$ for the particular case of optimizing the adaptive fuzzy logic controller is given in Sect. 5

Considering $W \in \mathbb{R}^n$ as the adjustable parameter vector composed of the fuzzy logic consequent part, the output of the adaptive fuzzy logic controller is expressed by

$$Y = \Phi^T W + \sigma = \hat{\Phi}^T \hat{W}, \tag{11}$$

where $\sigma$ is the fuzzy logic output error and $\hat{\Phi} \in \mathbb{R}^n$ is an $n$-dimensional vector representing known functions of the fuzzy logic antecedent part [7]. $\hat{\Phi}$ is defined as follows:

$$\hat{\Phi} = \frac{f^i}{\sum_{i=1}^{n} f^i}. \tag{12}$$

More details about the choice of fuzzy rules can be found in [8].

The control law is given as [7]:

$$\begin{aligned} v_1 &= \hat{\Phi}_\epsilon^T \hat{W}_\epsilon \\ v_2 &= \hat{\Phi}_\psi^T \hat{W}_\psi \\ u_2 &= \hat{\Phi}_\theta^T \hat{W}_\theta \end{aligned} \tag{13}$$

and the error dynamic equation is as follows: [7]:

$$\ddot{e} + K_d \dot{e} + K_p e = \hat{\eta}\sigma, \tag{14}$$

where $\hat{\eta}$ is the estimation of $\eta$, defined as

$$\eta_\epsilon = \frac{L_a}{J_\epsilon}, \quad \eta_\psi = \frac{L_a}{J_\psi}, \quad \eta_\theta = \frac{L_h}{J_\theta}, \tag{15}$$

and $\sigma = \hat{\Phi}^T \hat{W} - \Phi^T W$ from (11).

The controllers have to drive the tracking errors $e_\epsilon$, $e_\psi$, and $e_\theta$ to zero. To guarantee that the outputs of the system track the desired trajectories, we must have a tuning method for the parameters $\hat{W}$. The following theorem provides this tuning method [7,22].

**Theorem 1** *Considering a nonlinear system in the form of (1), (2), and (3) with the control law (13), the stability of the closed-loop system is achieved with the following adaptation law:*

$$\dot{\hat{W}} = -\Gamma \hat{\Phi} B^T P E, \tag{16}$$

*where* $\Gamma = diag(\gamma_1, \gamma_2, \ldots, \gamma_j)$ *and* $\gamma_l$ *is a positive constant,* $l = 1, \ldots, j$. $p$ *is a chosen symmetric positive definite matrix that satisfies the following Lyapunov equation:*

$$A^T P + PA = -Q, \tag{17}$$

*with* $Q > 0$, *and:*

$$E = \begin{bmatrix} e \\ \dot{e} \end{bmatrix}, \; A = \begin{bmatrix} 0 & 1 \\ -K_p & -K_d \end{bmatrix}, \; B = \begin{bmatrix} 0 \\ \hat{\eta} \end{bmatrix},$$

*where* $K_p > 0$ *and* $K_d > 0$.

***Proof*** See [7]. □

### 3.2 Optimization of the controller

During the adaptive fuzzy controller design, one issue arises, i.e., selecting the design parameter values of the controller, including $K_p$, $K_d$, and the parameters of the fuzzy system. For each fuzzy controller, Gaussian membership functions have been considered. Each membership function has two parameters, i.e., center $c$ and sigma $\sigma$. Choosing the right values for these parameters can greatly affect the performance of the controllers. Moreover, because the 3-DOF helicopter model is highly complex and nonlinear, any small change in the $K_p$ and $K_d$ values can lead to unstable behavior. In this paper, the MPSO algorithm is implemented to find the optimal values of these parameters. A fitness function computes the cost of each particle while the algorithm searches for the best value. We choose the root-mean-square error (RMSE) as the cost function because it intensifies the impact of large errors. So, the optimization problem can be defined as follows:

$$\begin{aligned} \min \quad & f(\mathbf{K}, \mathbf{\Gamma}, \mathbf{c}) = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(e_\epsilon^2 + e_\psi^2 + e_\theta^2\right)}, \\ \text{s.t.} \quad & -45^\circ \leq \theta \leq +45^\circ, \\ & -27.5^\circ \leq \epsilon \leq +30^\circ, \end{aligned} \tag{18}$$

where $\mathbf{K} = [K_\epsilon, K_\psi, K_\theta, K_p, K_d]$, $\mathbf{0} = [\Gamma_\epsilon, \Gamma_\psi, \Gamma_\theta]$, $\mathbf{c}$ is a vector of all centers of the membership functions (they can be found in Table 2) and the errors ($e_\epsilon$, $e_\psi$, and $e_\theta$) are defined in (8).

## 4 MPSO algorithm

The PSO algorithm is an optimization algorithm based on the behavior of bird flocking and was first introduced in 1995 [23]. For the first iteration of the algorithm, the velocity and position of all particles are initialized randomly. They share their information with their neighbors as they move toward the solution. Therefore, apart from the experience that each particle gains, they also use other particles' experience. This combination updates their positions and velocities based on their knowledge and the most successful particle's experience, i.e., the one closer to the target. And a cost function measures this success. The longer the distance between a particle and the target is, the higher the cost function value for that particle is.

**Table 2** Parameter values of the optimized controllers

| Parameters | MPSO | PSO |
|---|---|---|
| $K_\epsilon$ | 83.21 | 68.46 |
| $K_\psi$ | 168.53 | 157.95 |
| $K_\theta$ | 10.15 | 125.58 |
| $K_p$ | 1.78 | 5.57 |
| $K_d$ | 48.46 | 33.19 |
| $\Gamma_\epsilon = \Gamma_\psi$ | diag [79 68 33 0 79 42 76] | diag [70 102 77 125 55 13 112] |
| $\Gamma_\theta$ | diag [53 48 11 49 3 88 19] | diag [172 100 119 185 70 155 113] |
| $c_{NLe}$ | −2.80 | −120.92 |
| $c_{NMe}$ | −0.00 | −48.60 |
| $c_{NSe}$ | −0.00 | −0.04 |
| $c_{Ze}$ | 0.00 | 0.00 |
| $c_{PSe}$ | $-c_{NSe}$ | $-c_{NSe}$ |
| $c_{PMe}$ | $-c_{NMe}$ | $-c_{NMe}$ |
| $c_{PLe}$ | $-c_{NMe}$ | $-c_{NMe}$ |
| $c_{NL\dot{e}}$ | −9.51 | −129.19 |
| $c_{NM\dot{e}}$ | −6.32 | −125.02 |
| $c_{NS\dot{e}}$ | −0.97 | −0.75 |
| $c_{Z\dot{e}}$ | 0.00 | 0.00 |
| $c_{PS\dot{e}}$ | $-c_{NS\dot{e}}$ | $-c_{NS\dot{e}}$ |
| $c_{PM\dot{e}}$ | $-c_{NM\dot{e}}$ | $-c_{NM\dot{e}}$ |
| $c_{PL\dot{e}}$ | $-c_{NM\dot{e}}$ | $-c_{NM\dot{e}}$ |
| RMSE | 0.187 | 0.236 |

The following equations update the position and velocity of particle $i$ [13]:

$$v_i(t+1) = wv_i(t) + c_1 r_1(p_{best,i}(t) - x_i(t)) + c_2 r_2(g_{best}(t) - x_i(t)), \quad (19)$$

$$x_i(t+1) = x_i(t) + v_i(t+1), \quad (20)$$

where $v_i(t)$ and $x_i(t)$ are, respectively, the velocity and the position of the ith particle at instance $t$. $c_1$ and $c_2$ are cognitive and social acceleration factors, respectively. $r_1$ and $r_2$ are uniform random numbers distributed within the interval [0, 1], and $w$ is the inertia weight. $p_{\text{best},i}$ and $g_{\text{best}}$ are, respectively, the best solution that the ith particle and all particles have obtained so far. In every iteration, the best solution of each particle and the best solution of all particles are stored. In the following iteration, the algorithm uses the stored information to modify the position and velocity of particles. Hence, the particles gradually move toward $g_{\text{best}}$ until reaching the target. The parameters of (19) are selected as follows:

- The inertia weight ($w$) balances the local and global search. A large value intensifies the global search and a small value the local search. In the beginning, its value should be large to search the entire space extensively. Then, it should gradually be reduced to reach the optimal solution.
- The cognitive and social acceleration factors ($c_1$ and $c_2$) also help balance the local and global search; they are often set to the same value. They represent the effects of $p_{\text{best},i}$ and $g_{\text{best}}$ on the velocity of each particle.

The iterative procedure of the algorithm continues until the stopping condition is met, such as achieving an acceptable cost function value or reaching a maximum number of iterations.

The PSO algorithm is easy to implement with few parameters to modify. Nevertheless, there are some drawbacks. In problems with many local minimums, the algorithm is susceptible to falling into one. Moreover, specifying the exact area of the search space is not always possible. To overcome these drawbacks, a modified version of this algorithm is presented in [13].

In the modified version, the search space of each particle is modified based on the parameters of that particle. So after some iterations, the search space of each particle is decreased or increased independent of other particles. An adaptive search space leads to searching in a more reasonable bounded space. Furthermore, the algorithm considers an elimination phase; the new particles substitute for the poor

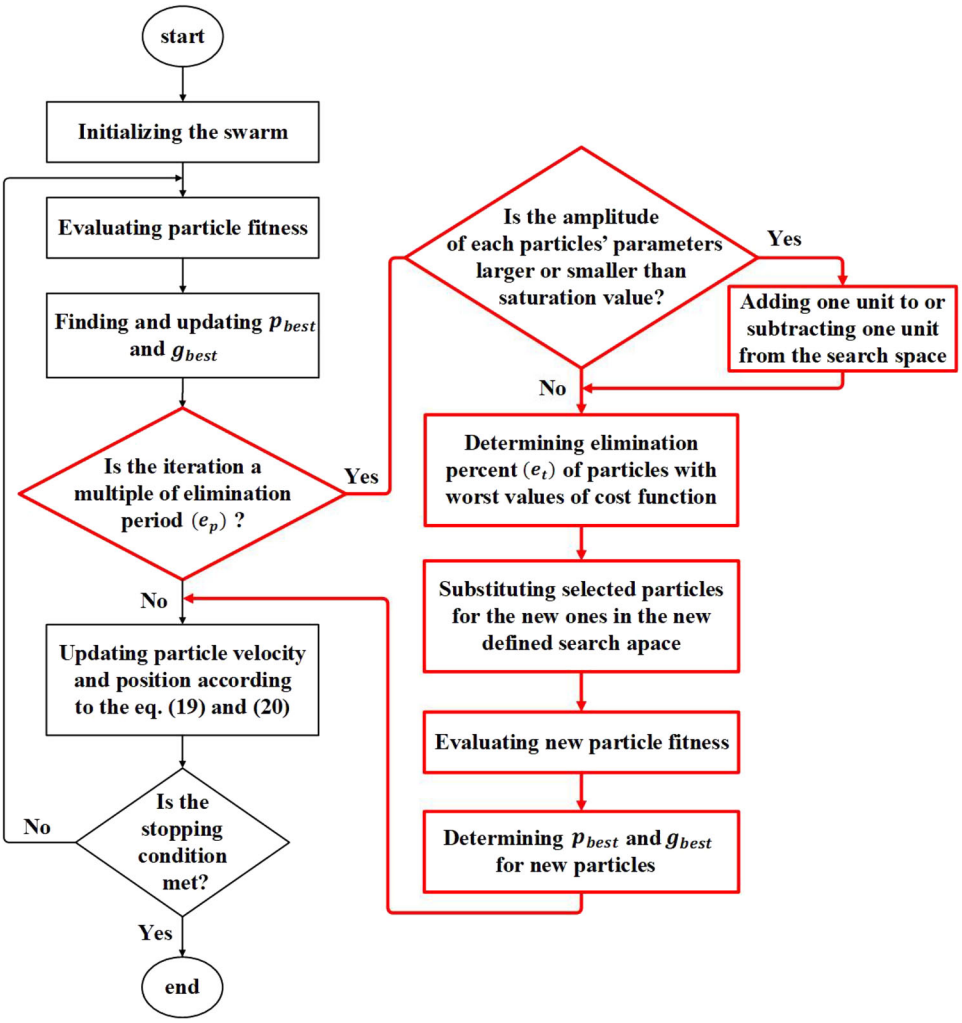



**Fig. 3** Flowchart of the MPSO algorithm (red blocks show the parts added to the standard PSO algorithm)

particles, i.e., particles with the largest cost functions. This elimination phase, $e_p$, depends on the number of iterations. For example, $e_p$ set to 30 means that the elimination phase is performed every 30 iterations. Figure 3 presents the flowchart of the MPSO algorithm. The red blocks show the parts added to the standard PSO algorithm to clarify the differences. The $e_t$ in the flowchart is determined based on the percentage of the initial population and shows the number of particles to be deleted in every elimination phase. For example, if $e_t = 20$, 20% of all particles with the highest cost function values are substituted by new particles in the new search space at every elimination step. The saturation percentage is an arbitrary value of the upper or lower bound of the search space [9]. For instance, if the initial bounds of the search space are considered to be $-1$ and 1, and the saturation value is set to 90%, the values of each particle's parameters are checked in each elimination phase. Then, for the upper bound, 1, if these values are less than or greater than 1, the boundary is decreased to 0.9 or increased to 1.1. For the lower bound, $-1$, if the values are lower than or greater than $-1$, the boundary is decreased to $-1.1$ or increased to $-0.9$. So, when the value is lower than the bound, one unit is subtracted from the search space, and when it is greater than the bound, one unit is added. This unit is determined based on the saturation value: 100% $-90\%$ (saturation value) $= 0.1$. The general framework of the algorithm is described in Algorithm 1. $[x_{min}\ \ x_{max}]$ in Algorithm 1 defines the search space.

## 5 Simulation result

This section presents the effectiveness of the proposed adaptive fuzzy controller. We compare the performance of the MPSO and the standard PSO algorithms to optimize the considered controller for the 3-DOF helicopter. A comparison has also been made between the adaptive fuzzy controller and a classical PID controller.

The dynamical system is simulated in a MATLAB Script file (m-file) with the numerical integration method of trapezoidal and the sampling time of 0.01. Table 1 shows the values

**Algorithm 1** MPSO algorithm

```
Initialization
for i=1 to the swarm size do
  Initialize the velocity and position randomly
end for
Main loop
for i=1 to the maximum iteration number do
  Evaluate each particle
  Find and update g_best and p_best for each particle
  Elimination phase
  if the reminder of i/e_p = 0 then
    for i=1 to the swarm size do
      if g_best > (saturation value) x_max then
        x_max ← x_max + (1 − threshold) x_max
        x_min ← x_min + (1 − threshold) x_min
      end if
      if g_best < (saturation value) x_min then
        x_min ← x_min − (1 − threshold) x_min
        x_max ← x_max − (1 − threshold) x_max
      end if
    end for
    Identify e_t% of the swarm with the highest fitness values and delete them
    Create new particles in the newly defined search space
    Set velocity of new particles to zero
    Evaluate new particles and find p_best for them
    Update g_best for the swarm
  end if
  for i=1 to the swarm size do
    Update velocity and position of particles according to (19) and (20)
  end for
end for
```

of the 3-DOF helicopter model. The initial values of the pitch, roll, and yaw angles are set to zero. The following equation is considered to be the desired trajectory for the roll and yaw angles [7]:

$$\epsilon_d, \psi_d = \frac{1}{1 + e^{-2.5(t+2)}}. \tag{21}$$

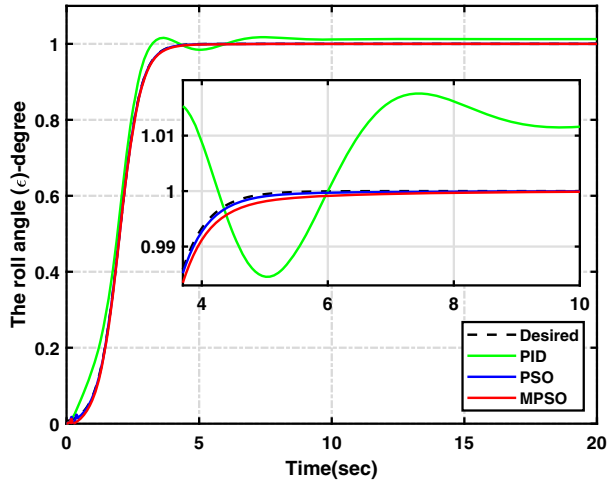
As stated in Sect. 3, the value of the inertia weight $w$ should be large at the beginning to help the global search and then be reduced to find the optimal solution. Therefore, $w = 1$ at the beginning and linearly is reduced to 0.98 of its value at each iteration. The acceleration factors, $c_1$ and $c_2$, are both set to 2, and the population size is chosen to be 30. These settings are the same for both PSO and MPSO algorithms. In the MPSO algorithm, the elimination percent $e_t$ and the elimination period $e_p$ are set to 75 and 40, respectively. Both algorithms have been run 25 times, and the best results for each algorithm have been reported. The constraint handling method used in PSO is death penalty approach. In this method, the infeasible regions of search space are not explored, resulting in losing valuable information in these regions [24]. But this issue has been addressed in MPSO algorithm since in MPSO, the search space of each parameter is dynamic and can be modified based on the value of that parameter.

Each controller has two inputs $e$ and $\dot{e}$, and there are seven Gaussian membership functions for each input, namely negative large (NL), negative medium (NM), negative small (NS), zero (Z), positive small (PS), positive medium (PM), and positive large (PL) [22]. The membership functions are selected as follows:

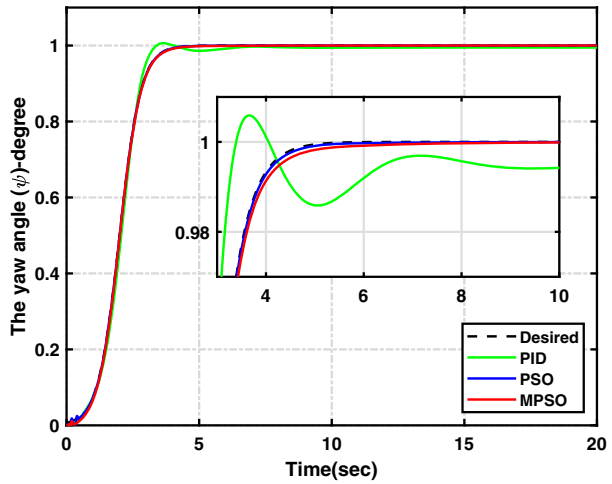
$$\mu_{A_j^{NL}} = 1/(1 + e^{5(x_j + c_{NL})})$$
$$\mu_{A_j^{NM}} = e^{-(x_j + c_{NM})}$$
$$\mu_{A_j^{NS}} = e^{-(x_j + c_{NS})}$$
$$\mu_{A_j^{Z}} = e^{-x_j}$$
$$\mu_{A_j^{PS}} = e^{-(x_j - c_{PS})}$$
$$\mu_{A_j^{PM}} = e^{-(x_j - c_{PM})}$$
$$\mu_{A_j^{PL}} = 1/(1 + e^{-5(x_j - c_{PL})})$$

Table 2 represents the final optimized values of controllers' parameters with the cost function values. The centers $c$ of membership functions for all three controllers are the same, and sigmas are considered to be 1. As the table shows, the centers of membership functions have substantially high values when optimized with the PSO algorithm. This happens when we already know the approximate boundary of some parameters, but we cannot define it in the algorithm. Indeed, defining the upper and lower bound of the search space for each parameter is possible in the MPSO algorithm, allowing it to avoid falling into local minimums.

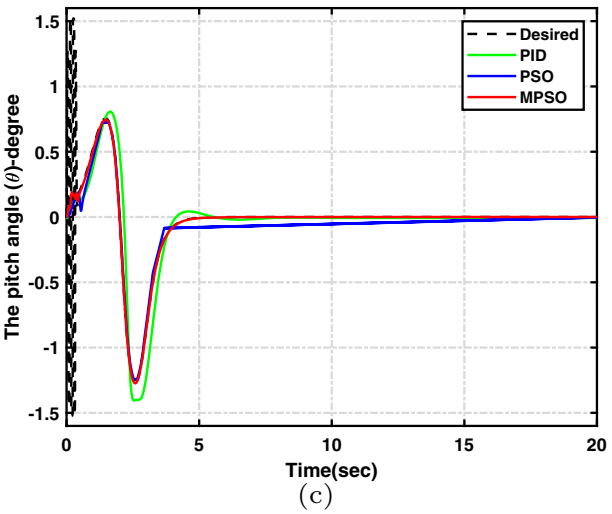
Figures 4, 5, and 6 present the yaw, pitch, and roll angles, the tracking errors, and the control signals for the nominal case. In these figures, the adaptive fuzzy logic controller's performance has been compared with a classical PID controller's performance when there are three PID controllers to control roll, yaw, and pitch angles. The control structure for the PID controller is like the one in [25], and the gains are optimized through the MPSO algorithms. The figures reveal a steady-state error of less than 0.02 for the roll and yaw angles with the PID controller. The system's settling time (for all three angles) is higher for the PID controller. The adaptive fuzzy logic controller improves tracking when optimized with PSO and MPSO algorithms. The PID controller's only priority over the adaptive fuzzy controller optimized by the PSO algorithm is its lower control effort. Overall, these figures prove the adaptive fuzzy logic controller's superiority over the PID controller.




(a)

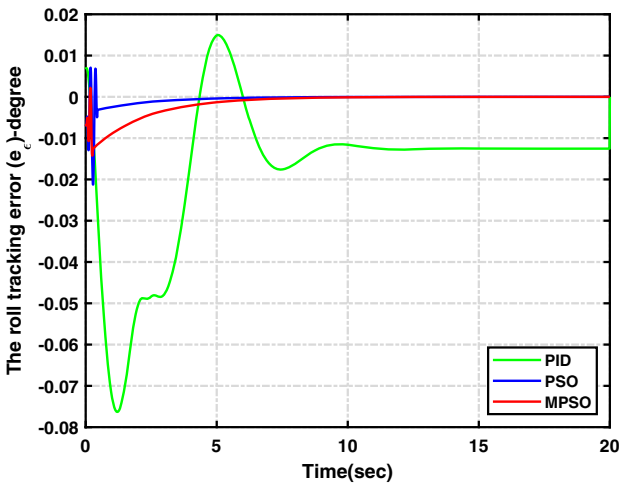



(b)

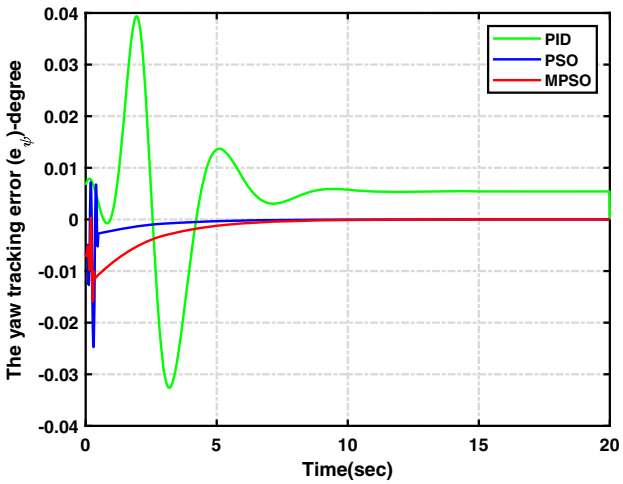



(c)

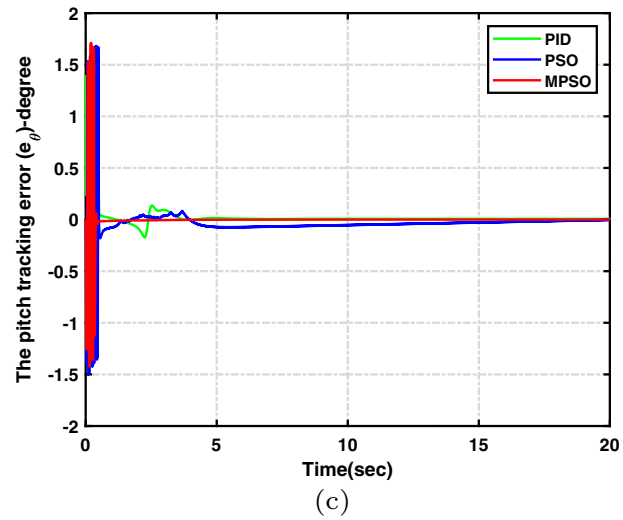
**Fig. 4** Simulation results in the nominal case: outputs of the system




(a)

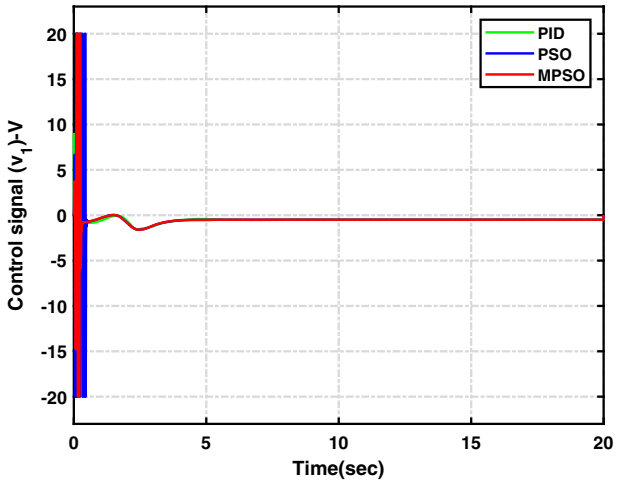



(b)

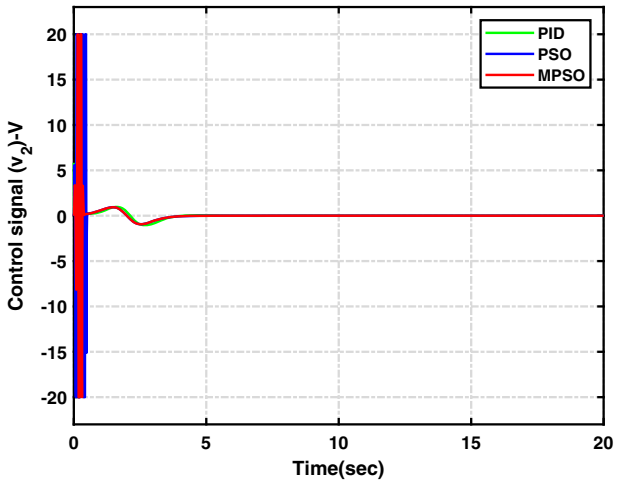



(c)

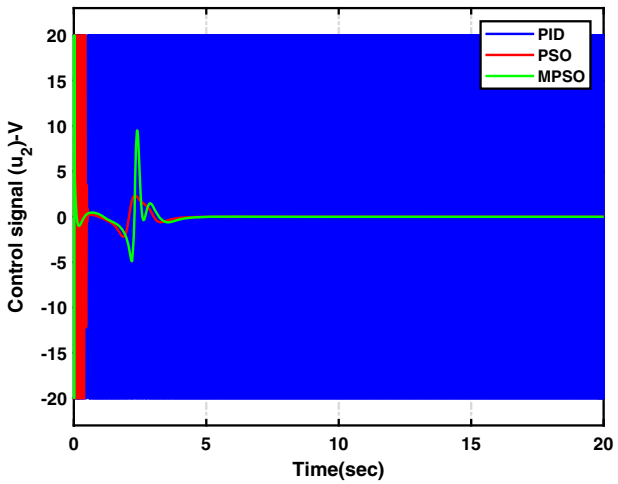
**Fig. 5** Simulation results in the nominal case: motion tracking errors




(a)




(b)




(c)

**Fig. 6** Simulation results in the nominal case: control signals

**Table 3** Comparison between the performance of PSO and MPSO algorithms

| | IACS | RMSE | ITAE |
|---|---|---|---|
| PSO | 213.429 | 0.236 | 6.9818 |
| MPSO | 15.177 | 0.187 | 0.3077 |

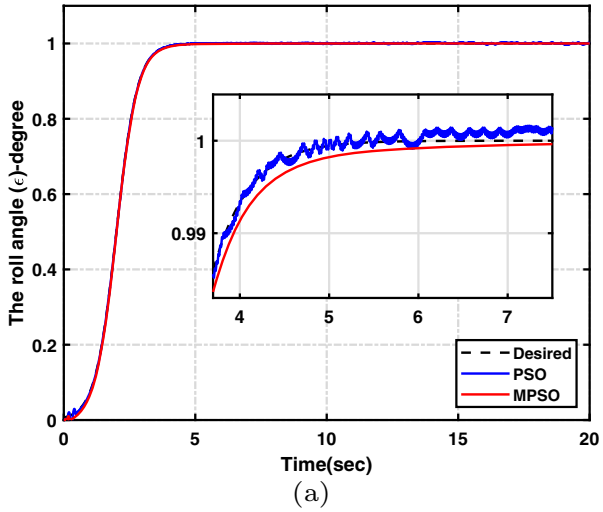
Comparing the performance of PSO and MPSO algorithms on Figs. 4, 5, and 6, it can be seen that their results are almost the same for the roll and yaw angles, with the PSO algorithm having less settling time. But the difference increases for the pitch angle, where the MPSO algorithm has much less tracking error. Consequently, the control signals for the PSO algorithm fluctuate more, especially for the pitch angle. The main reason for this fluctuation in the control signal is the incapability of the PSO algorithm in defining separate search spaces for different parameters and modifying these search spaces based on the value of each parameter. In other words, it easily gets trapped into local minimums when there are many local minimums or many parameters to optimize.

To further investigate the algorithms' performance, the results are compared in Table 3 based on RMSE, the integral of time-weighted absolute error (ITAE), and the integral of the absolute values of control signals (IACS) which is calculated as follows:

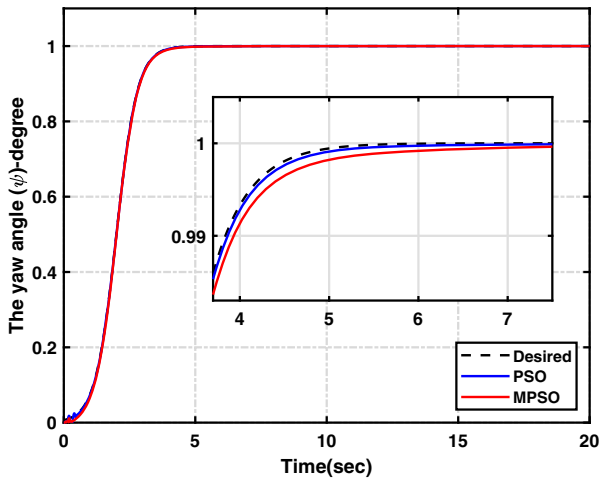
$$IACS = \int_0^T (|v_1| + |v_2| + |u_2|)\, dt, \tag{22}$$

where $v_1$, $v_2$, and $u_2$ are the roll, yaw, and pitch control signals and $T$ is the final time instant. Notice that IACS and ITAE are only metrics to compare the performance of algorithms and not the objective function. They have been computed after optimizing the parameters of controllers and applying them to the model. As the results in Table 3 show, the MPSO has a lower value for all three metrics and, therefore, a better performance.

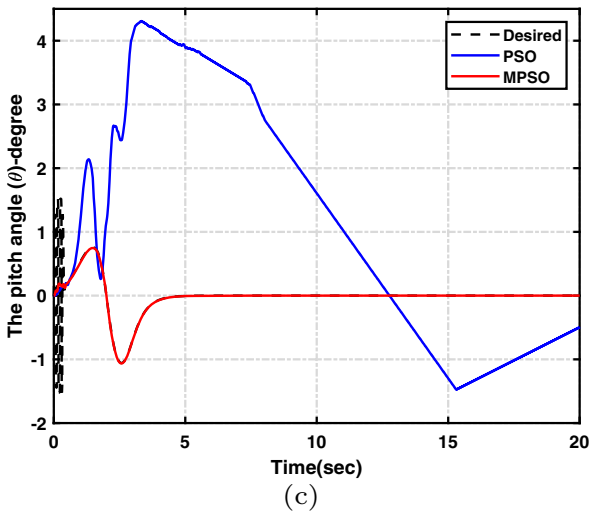
To demonstrate the robustness of the controllers, we tested them under parametric uncertainties. For this purpose, we decreased the system's mass to half of its nominal case, and in another case, we increased it to one and a half times the nominal case, as it was done in [7]. Figures 7, 8, 9, 10, 11, and 12 depict the results. The roll angle for the PSO algorithm fluctuates more than the nominal case, leading to more fluctuation in the control signal. Moreover, the pitch angle for the PSO algorithm does not track the desired trajectory. But for the MPSO algorithm, results are identical to the nominal case, demonstrating the controller's robustness facing uncertainties. The only difference is the magnitude of the $u_2$ to adjust the mass change. It decreases when the system's mass




(a)

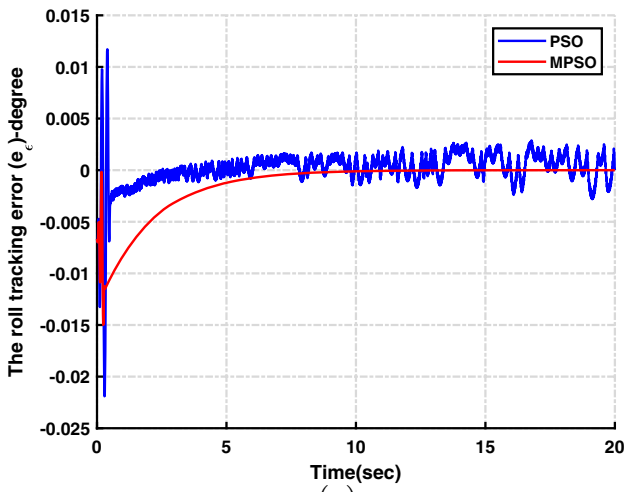



(b)

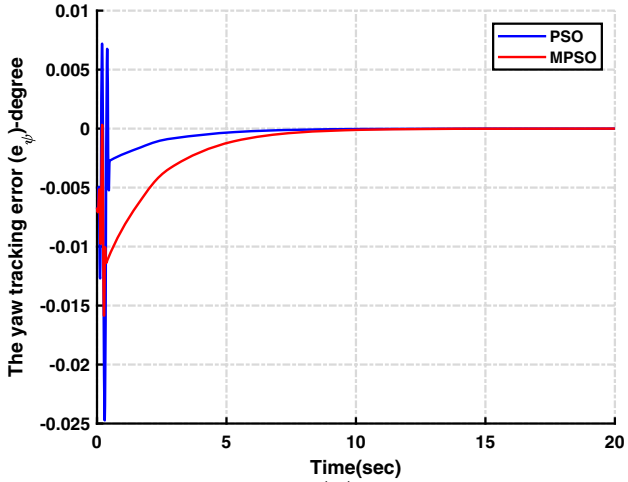



(c)

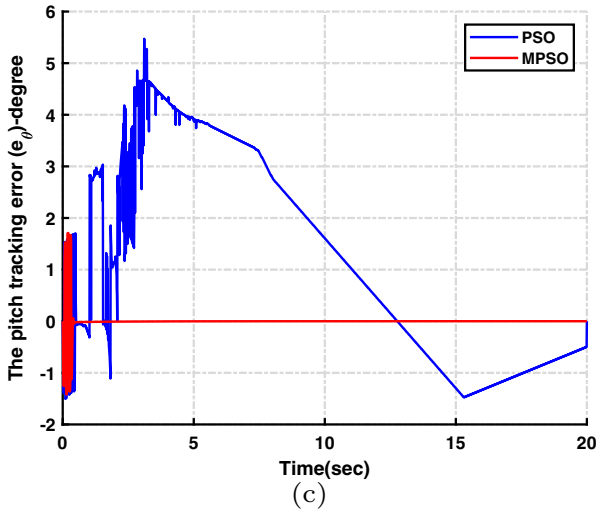
**Fig. 7** Simulation results under the helicopter's mass change (half mass): outputs of the system




(a)

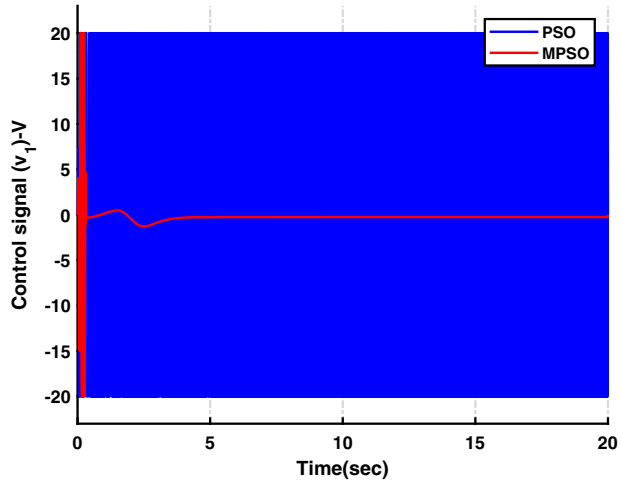



(b)

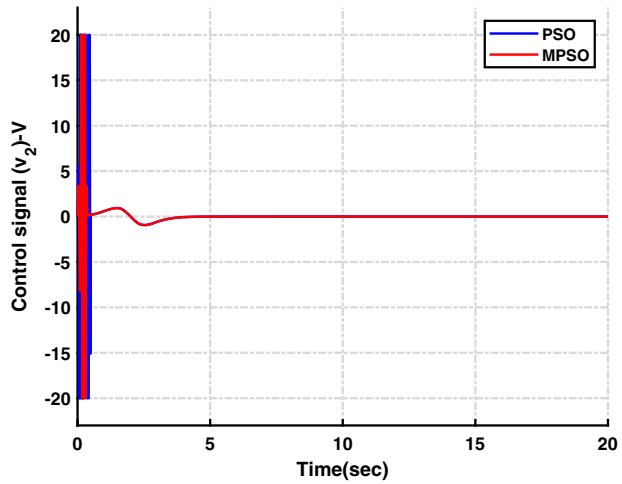



(c)

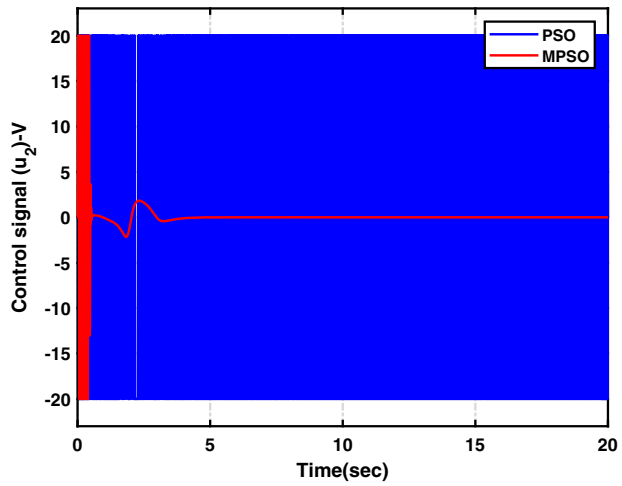
**Fig. 8** Simulation results under the helicopter's mass change (half mass): motion tracking errors




(a)

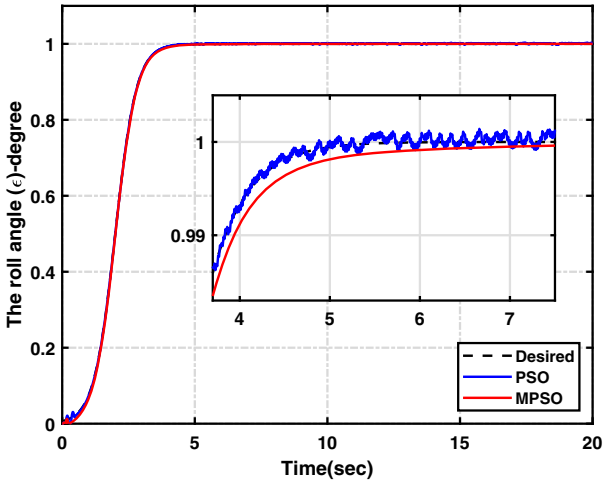



(b)

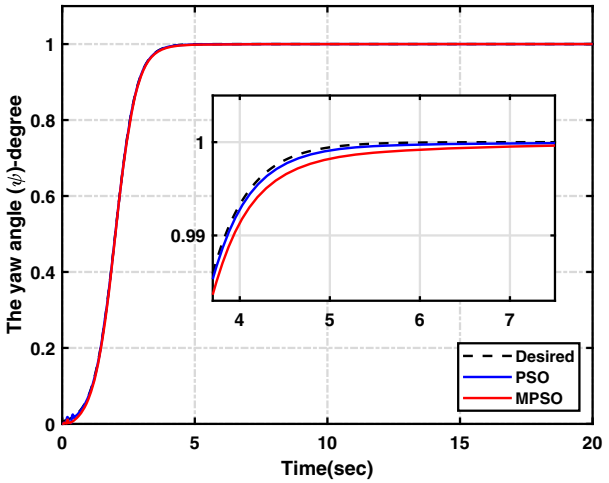



(c)

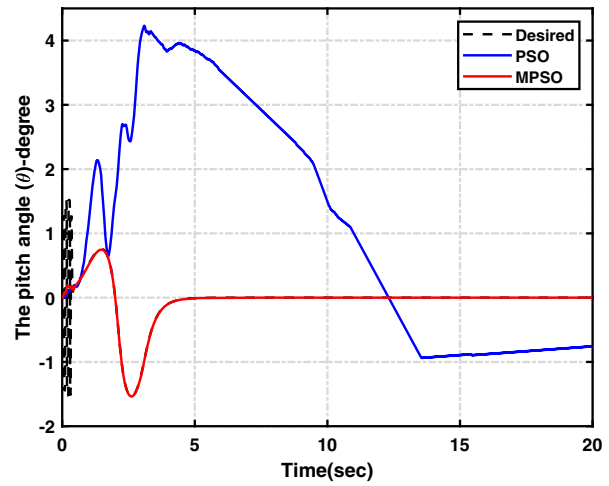
**Fig. 9** Simulation results under the helicopter's mass change (half mass): control signals




(a)

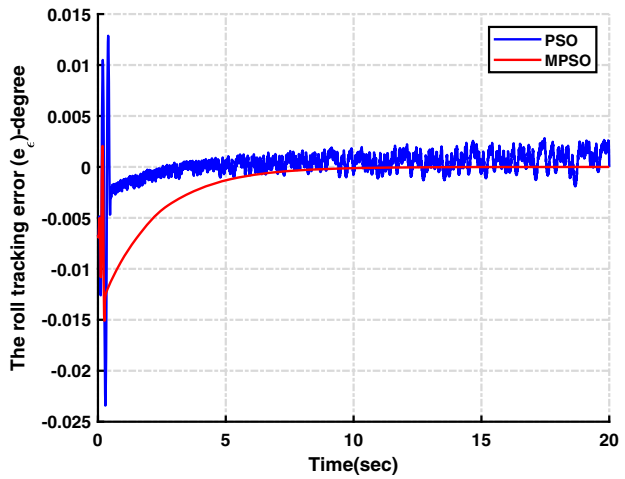



(b)

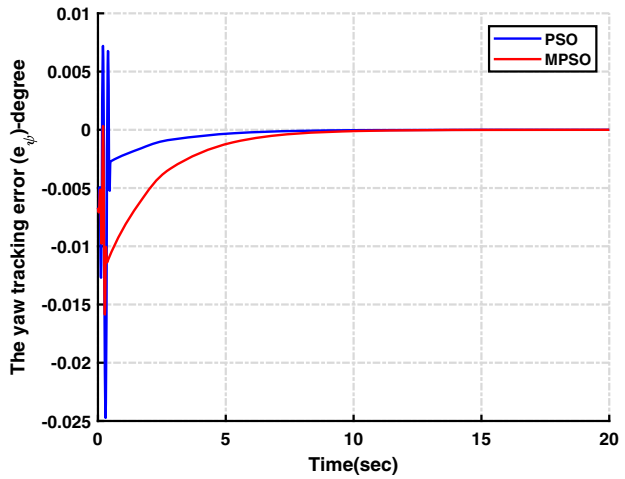



(c)

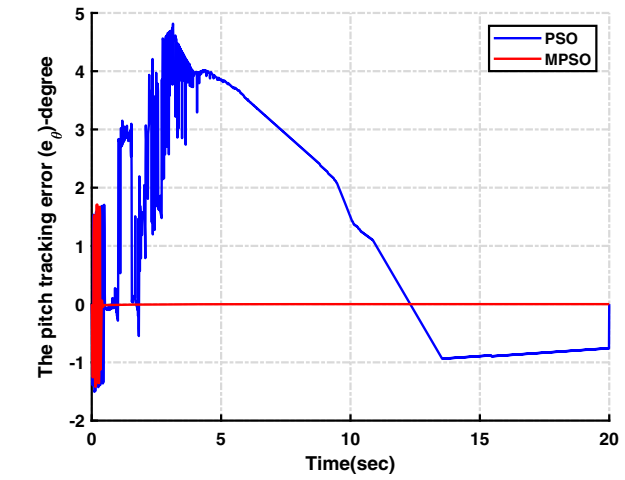
**Fig. 10** Simulation results under the helicopter's mass change (one and a half times mass): outputs of the system




(a)

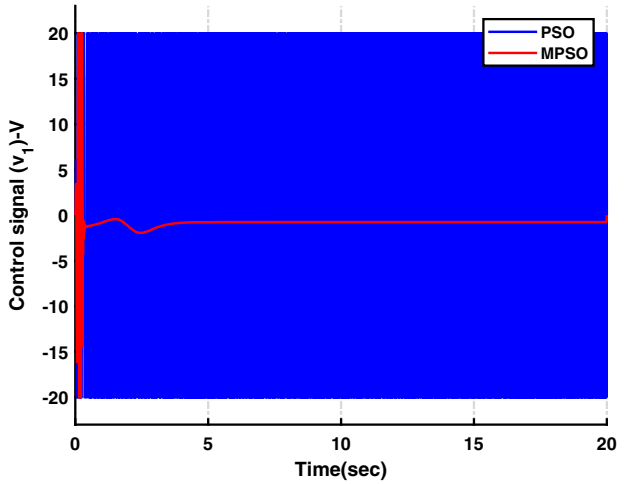



(b)

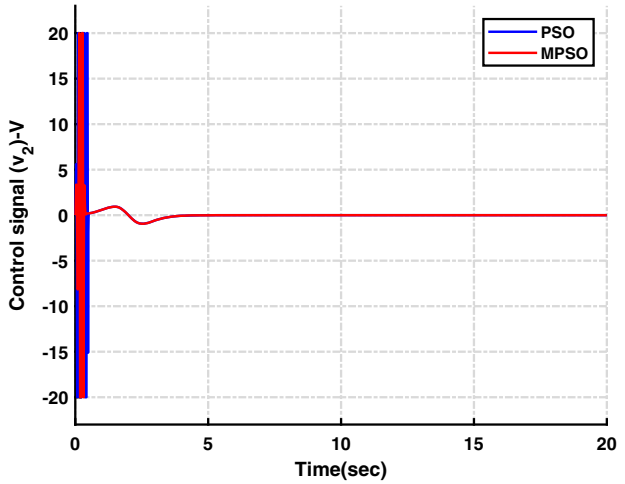



(c)

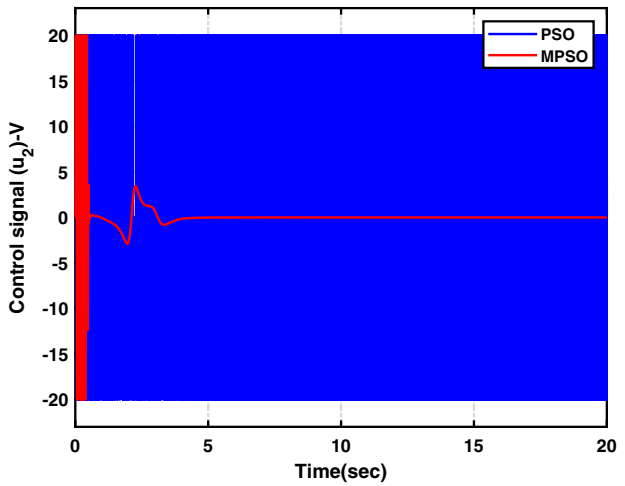
**Fig. 11** Simulation results under the helicopter's mass change (one and a half times mass): motion tracking errors




(a)

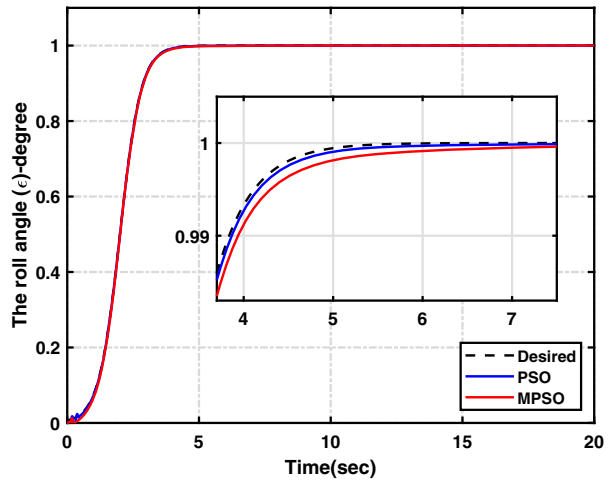



(b)

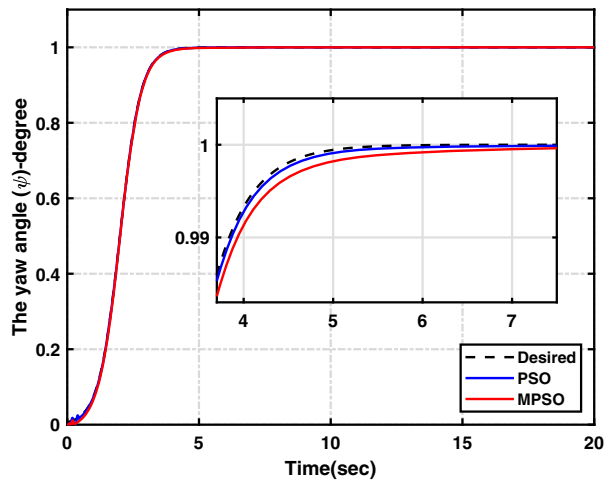



(c)

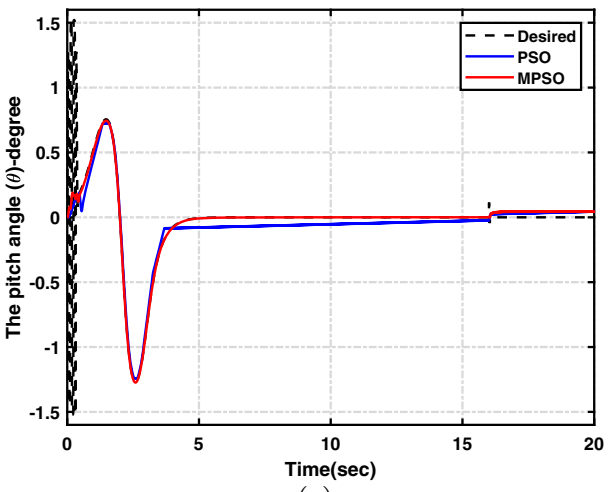
**Fig. 12** Simulation results under the helicopter's mass change (one and a half times mass): control signals




(a)

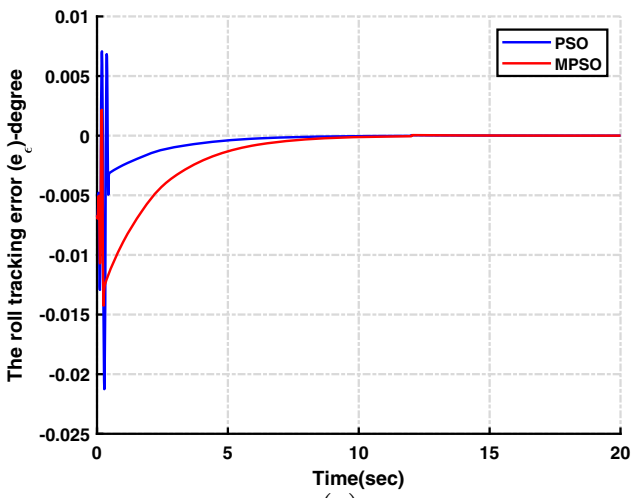



(b)

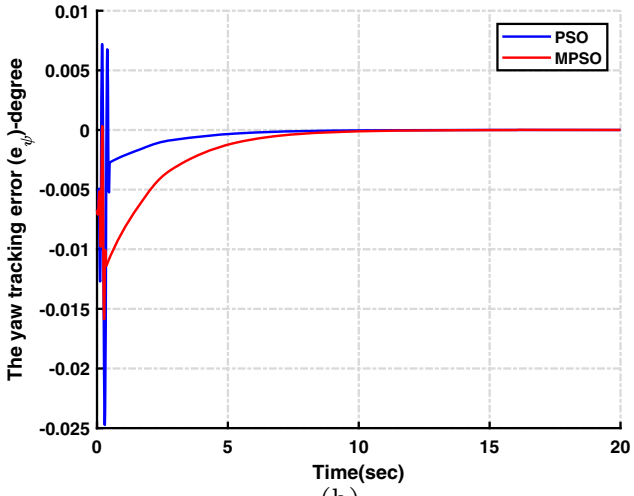



(c)

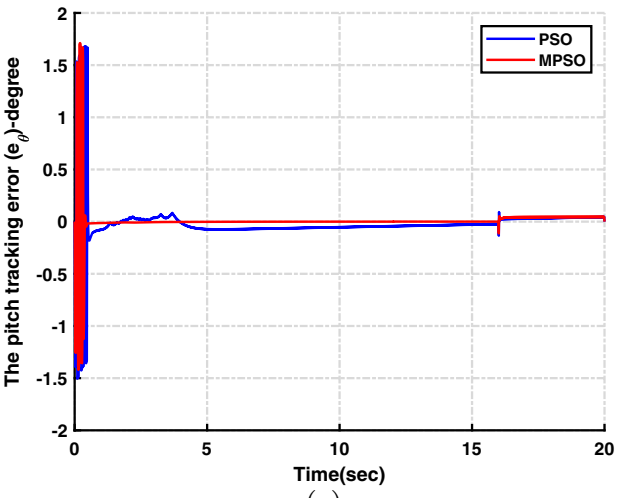
**Fig. 13** Simulation results under disturbances: outputs of the system




(a)




(b)




(c)

**Fig. 14** Simulation results under disturbances: motion tracking errors

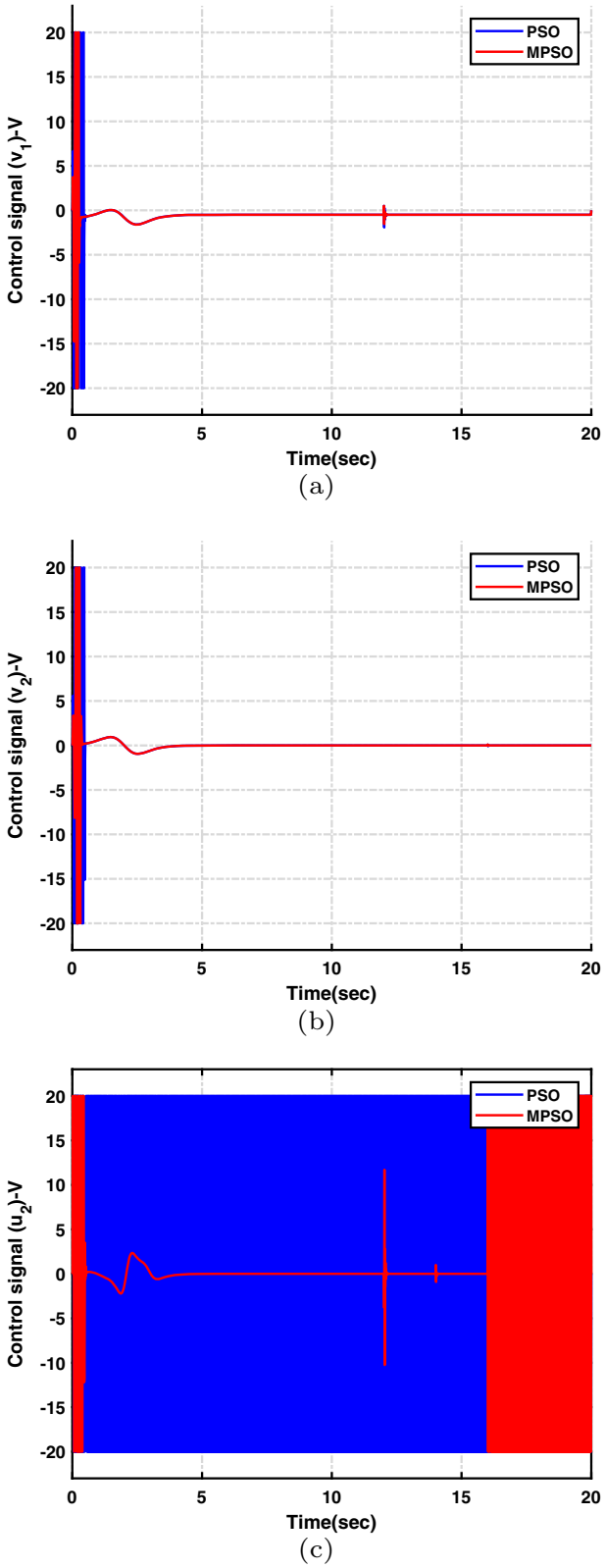



**Fig. 15** Simulation results under disturbances: control signals

is half of its nominal value and increases when the mass has increased one and a half times.

Moreover, we performed an additional test; we applied step disturbances of 1, 1, and 0.1 to the roll, pitch, and yaw angles at 12, 14, and 16 seconds, respectively. Figures 13, 14, and 15 show the results. Although applying the disturbance to the yaw angle leaves a small pitch tracking error, the controller is able to decay the roll and pitch tracking errors to zero without fluctuation or unstable behavior.

All these algorithms are implemented to optimize the adaptive fuzzy logic controllers of the 3-DOF helicopter model. The population size is selected to be 30 for all algorithms. For those algorithms incapable of defining the upper and lower bound of search space for each parameter, [0 200] is chosen as the search space. For other algorithms (MPSO, ACO, and ICA), Table 4 shows the upper and lower bound of search space for each parameter. The cost function is also the same for all (RMSE). Other parameters of these algorithms are all included in Table 5. For PSO and MPSO algorithms, the parameters' values are included in Sect. 5.

### 5.1 Further analysis of the MPSO algorithm performance against other metaheuristics

To further investigate the effectiveness of the MPSO to optimize the 3-DOF helicopter control, we compare its results with the following algorithms' results:

- MPSO algorithm [13]
- Standard PSO algorithm [23]
- Genetic algorithm (GA) [26]
- Ant colony optimization (ACO) [27]
- Imperialist competitive algorithm (ICA) [28]
- Gray wolf optimizer (GWO) [29]
- Bat algorithm (BA) [30]
- Differential evolution (DE) [31]

Each algorithm has been run 25 times in MATLAB. The best values of the fitness function (the lowest ones) and the mean of fitness function values for the 25 runs with 500 iterations are reported in Table 6. The elapsed time in the table refers to the average time required to complete one run of each algorithm. According to the results in the table, the MPSO algorithm has the lowest cost function value among others. As the cost function is RMSE, the MPSO provides controllers' gains and parameters that yield better accuracy. It also has the second-fastest elapsed time. In other words, its computational complexity is lower than the other algorithms except for the PSO algorithm.

Figure 16a shows the RMSE evolution of algorithms, with a zoom between iterations 0–100 in Fig. 16b to make it more clear; the convergence speed of the MPSO algorithm is faster than other ones, and it converges to its optimal value (0.1879)

**Table 4** Upper and lower bound of search space for each parameter in MPSO, ACO, and ICA

| Parameters | Lower bound | Upper bound |
|---|---|---|
| $K_\epsilon, K_\psi, K_\theta$ | 0 | 200 |
| $K_p, K_d$ | 0 | 100 |
| Centers for all fuzzy membership functions | −10 | 10 |
| $\Gamma_\epsilon, \Gamma_\psi, \Gamma_\theta$ | 0 | 100 |

**Table 5** Parameters of algorithms

| GA parameters | Value | ACO parameters | Value |
|---|---|---|---|
| Number of population | 30 | Number of ants | 30 |
| Percent of crossover | 0.8 | q | 0.1 |
| Percent of mutation | 0.3 | $\rho$ | 0.85 |
| | | $\alpha$ | 0.8 |
| | | $\beta$ | 0.8 |
| ICA parameters | Value | GWO parameters | Value |
| Number of initial countries | 30 | Number of population | 30 |
| Number of initial imperialists | 8 | a | decreases linearly from 2 to 0 |
| Revolution rate | 0.3 | | |
| $\beta$ | 2 | | |
| $\gamma$ | 0.5 | | |
| $\xi$ | 0.02 | | |
| BA parameters | Value | DE parameters | Value |
| Number of population | 30 | Number of population | 30 |
| Loudness | 0.5 | CR | 0.5 |
| Pulse rate | 0.5 | F | 0.3 |
| Minimum frequency | 0 | | |
| Maximum frequency | 2 | | |

**Table 6** Comparison of MPSO and other metaheuristics—500 iterations

| Algorithm | Best fitness function value | Mean of fitness function values | Elapsed time (minute) |
|---|---|---|---|
| MPSO | 0.1879 | 0.1949 | 65 |
| PSO | 0.2366 | 0.3115 | 64 |
| GA | 0.4554 | 0.7568 | 76 |
| ACO | 0.5780 | 0.7592 | 73 |
| ICA | 0.4261 | 0.5234 | 68 |
| GWO | 0.2001 | 0.2641 | 81 |
| BA | 0.9914 | 1.2729 | 75 |
| DE | 0.4067 | 0.7036 | 74 |

in less than 50 iterations. Second to the MPSO algorithm is the GWO with the fitness function value of 0.2001 reached at iteration 430. (The fitness function values are presented in Table 6.) The highest fitness function value at iteration 500 belongs to the BA (0.9914), which indicates its incapability in finding the global optimum or at least a better local optimum. Apart from that, GA has the slowest convergence rate, reaching its optimal value at iteration 490.

The algorithms are compared using Wilcoxon and Friedman tests to analyze the results statistically. Tables 7 and 8 depict the results of the comparison. According to $p$ values in Table 7, the null hypothesis is rejected with a confidence level of 0.99, meaning that MPSO shows a significant improvement over other algorithms. The high statistic numbers also verify that there is a significant difference between the performance of the algorithms. Moreover, the ranks computed

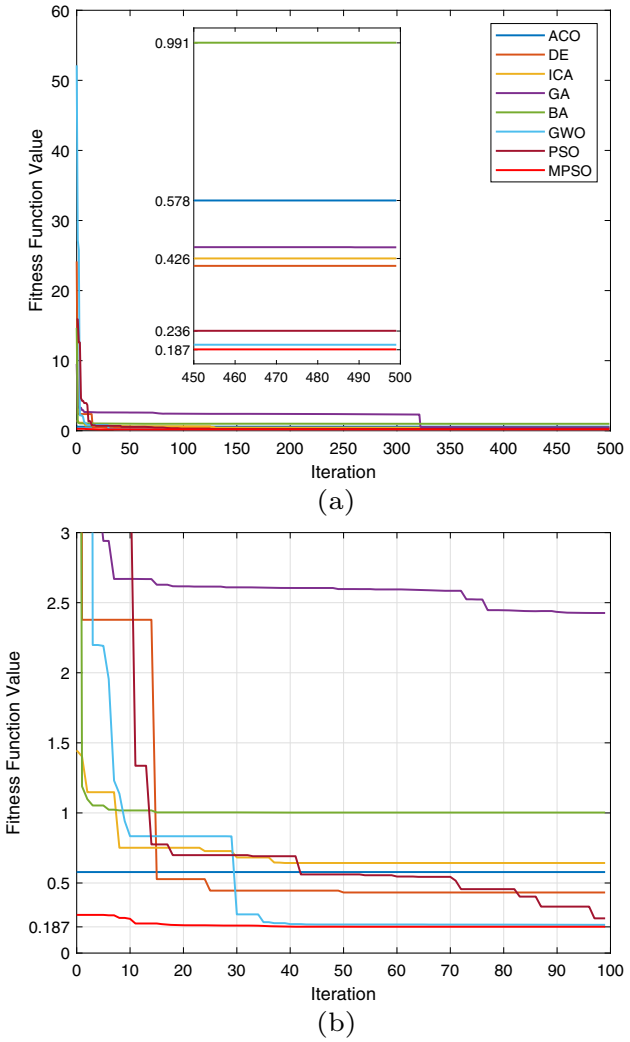



**Fig. 16** RMSE evolution of the MPSO and other metaheuristics: **a** Iteration numbers 0 to 500, **b** iteration numbers 0 to 100

**Table 7** Wilcoxon signed-rank test results

| Comparison | Statistic | $p$ value |
|---|---|---|
| MPSO vs PSO | 4129.0 | 0.00 |
| MPSO vs GA | 3273.0 | 0.00 |
| MPSO vs ACO | 8215.0 | 0.00 |
| MPSO vs ICA | 7075.0 | 0.00 |
| MPSO vs GWO | 8201.0 | 0.00 |
| MPSO vs BA | 5535.0 | 0.00 |
| MPSO vs DE | 18826.0 | 0.00 |

**Table 8** Friedman ranks

| | MPSO | PSO | GA | ACO | ICA | GWO | BA | DE |
|---|---|---|---|---|---|---|---|---|
| Ranks | 1.000 | 3.312 | 7.256 | 5.930 | 5.116 | 2.218 | 7.268 | 3.900 |

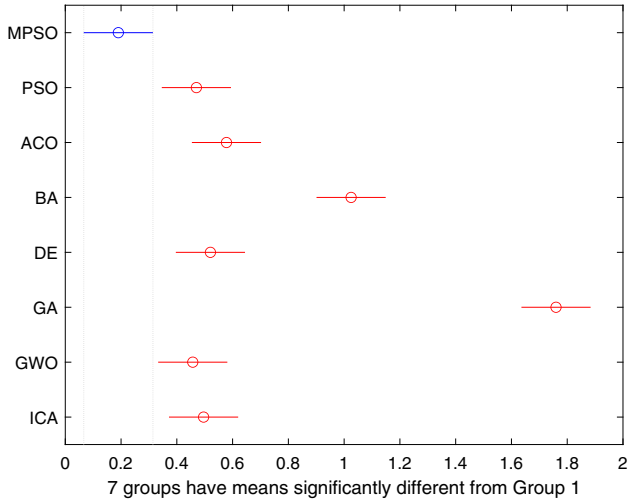



**Fig. 17** Multiple comparisons of means in Bonferroni test

through the Friedman test in Table 8 show that MPSO is the best performing algorithm, while BA is the worst.

We also performed the multi-comparison Bonferroni test with a confidence level of 0.05. The resulting $p$ value is $1.64 \times 10^{-104} < 0.05$, which indicates that the differences between algorithms are significant. The first seven rows of the matrix of pairwise comparison results ($c$) are as follows:

$$
\begin{array}{rrrrrr}
1.0000 & 2.0000 & -0.5280 & -0.2799 & -0.0318 & 0.0119 \\
1.0000 & 3.0000 & -0.6359 & -0.3878 & -0.1397 & 0.0000 \\
1.0000 & 4.0000 & -1.0828 & -0.8348 & -0.5867 & 0.0000 \\
1.0000 & 5.0000 & -0.5783 & -0.3302 & -0.0821 & 0.0009 \\
1.0000 & 6.0000 & -1.8177 & -1.5696 & -1.3215 & 0.0000 \\
1.0000 & 7.0000 & -0.5149 & -0.2669 & -0.0188 & 0.0218 \\
1.0000 & 8.0000 & -0.5538 & -0.3057 & -0.0576 & 0.0033
\end{array}
$$

and the multi-comparison graph is shown in Fig. 17. The first seven rows of matrix $c$ show that all comparisons involving the first group (MPSO) have confidence intervals that exclude zero, meaning that the differences between the means of group 1 and other groups are significantly different from zero at the 5% significance level. This can also be confirmed from Fig. 9. However, the differences between PSO, ACO, DE, GWO, and ICA are not significant since the 95% confidence interval for the difference between them includes zero. Therefore, the hypothesis that the true difference is zero cannot be rejected for them.

Thus, we can conclude that the MPSO algorithm is an efficient optimization tool for the 3-DOF helicopter control and other control systems. Indeed, in a previous paper ([13]), the MPSO algorithm has been implemented to design an interval type 2 fuzzy disturbance observer for a real ball and beam system.

## 6 Conclusion

This paper proposed optimizing a fuzzy adaptive controller for the 3-DOF helicopter system through the MPSO algorithm. The controller has many parameters to be defined, including the membership functions' parameters of the fuzzy part and gains of the adaptive part. As the system is highly nonlinear, a slight change in the value of these parameters can significantly affect the controllers' performance. The MPSO algorithm has a high ability to avoid the local minimums, making it a suitable algorithm for optimizing the controller. We assessed the algorithm's performance by comparing it with the standard PSO algorithm and six other well-known metaheuristic algorithms. The results show the fast convergence rate and low computational complexity of the MPSO algorithm. In particular, the standard PSO algorithm does not achieve satisfactory results, particularly in the presence of uncertainties and disturbances. On the other hand, the controller optimized through the MPSO algorithm shows robustness properties to uncertainties and disturbance. Applying the MPSO algorithm to other control structures and further improving its performance can be considered in future research.

**Author Contributions** SN contributed to conceptualization, methodology, software, formal analysis, writing–original draft, data curation, and visualization; MJB contributed to conceptualization, methodology, resources, writing—review & editing, funding acquisition, and supervision; BR performed writing—review & editing.

**Funding** This work was funded by NSERC Discovery Grant.

### Declarations

**Conflict of interest** The authors have no competing interests to declare that are relevant to the content of this article.

## References

1. Chen Y, Yang X, Zheng X (2018) Adaptive neural control of a 3-dof helicopter with unknown time delay. Neurocomputing 307:98–105
2. Chen M, Shi P, Lim C-C (2015) Adaptive neural fault-tolerant control of a 3-dof model helicopter system. IEEE Transact Syst, Man, Cybernetics: Syst 46(2):260–270
3. Castañeda H, Plestan F, Chriette A, de León-Morales J (2016) Continuous differentiator based on adaptive second-order sliding-mode control for a 3-dof helicopter. IEEE Trans Industr Electron 63(9):5786–5793
4. Pounds PE, Dollar AM (2014) Stability of helicopters in compliant contact under pd-pid control. IEEE Trans Rob 30(6):1472–1486
5. Uddin MM, Sarker P, Theodore CR, Chakravarty UK (2018) Active vibration control of a helicopter rotor blade by using a linear quadratic regulator. In: ASME 2018 International mechanical engineering congress and exposition . American Society of Mechanical Engineers Digital Collection
6. Dutta L, Kumar Das D (2021) Adaptive model predictive control design using multiple model second level adaptation for parameter estimation of two-degree freedom of helicopter model. Int J Robust Nonlinear Control 31(8):3248–3278
7. Chaoui H, Yadav S, Ahmadi RS, Bouzid AEM (2020) Adaptive interval type-2 fuzzy logic control of a three degree-of-freedom helicopter. Robotics 9(3):59
8. Chaoui H, Khayamy M, Aljarboua AA (2017) Adaptive interval type-2 fuzzy logic control for pmsm drives with a modified reference frame. IEEE Trans Industr Electron 64(5):3786–3797
9. Salahshour E, Malekzadeh M, Gholipour R, Khorashadizadeh S (2019) Designing multi-layer quantum neural network controller for chaos control of rod-type plasma torch system using improved particle swarm optimization. Evol Syst 10(3):317–331
10. Gonzalez H, Arizmendi C, Garcia J, Anguo A, Herrera C (2018) Design and experimental validation of adaptive fuzzy pid controller for a three degrees of freedom helicopter. In: 2018 IEEE international conference on fuzzy systems (FUZZ-IEEE), pp. 1–6 . IEEE
11. Xue S, Li Z, Yang L (2019) Training a model-free reinforcement learning controller for a 3-degree-of-freedom helicopter under multiple constraints. Meas Control 52(7–8):844–854
12. Yang X, Zheng X (2019) Adaptive nn backstepping control design for a 3-dof helicopter: theory and experiments. IEEE Trans Industr Electron 67(5):3967–3979
13. Naderi S, Rezaie B, Faramin M (2020) Designing an interval type-2 fuzzy disturbance observer for a class of nonlinear systems based on modified particle swarm optimization. Appl Intell 50(11):3731–3747
14. Yu G-R, Hsieh P-H (2019) Optimal design of helicopter control systems using particle swarm optimization. In: 2019 IEEE international conference on industrial cyber physical systems (ICPS), pp. 346–351 . IEEE
15. Humaidi AJ, Hasan AF (2019) Particle swarm optimization-based adaptive super-twisting sliding mode control design for 2-degree-of-freedom helicopter. Meas Control 52(9–10):1403–1419
16. Tan K, Cheong C, Peng Y (2016) A genetic approach for real-time identification and control of a helicopter system. Int J Comput Intell Control 8(1):11–18
17. Ding L, Wu H, Yao Y (2015) Chaotic artificial bee colony algorithm for system identification of a small-scale unmanned helicopter. Int J Aerosp Eng **2015**
18. Hu Y, Yang Y, Li S, Zhou Y (2020) Fuzzy controller design of micro-unmanned helicopter relying on improved genetic optimization algorithm. Aerosp Sci Technol 98:105685
19. Azimi MM, Koofigar HR (2015) Adaptive fuzzy backstepping controller design for uncertain underactuated robotic systems. Nonlinear Dyn 79(2):1457–1468
20. Shakourzadeh S, Farrokhi M (2020) Fuzzy-backstepping control of quadruped robots. Intel Serv Robot 13(2):191–206
21. Mendel JM, John RI, Liu F (2006) Interval type-2 fuzzy logic systems made simple. IEEE Trans Fuzzy Syst 14(6):808–821
22. Teiar H, Boukaka S, Chaoui H, Sicard P (2014) Adaptive fuzzy logic control structure of pmsms. In: 2014 IEEE 23rd International symposium on industrial electronics (ISIE), pp. 745–750 . IEEE
23. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-international conference on neural networks, vol. 4, pp. 1942–1948 . IEEE
24. Jordehi AR (2015) A review on constraint handling strategies in particle swarm optimisation. Neural Comput Appl 26(6):1265–1275
25. Chang W-D, Shih S-P (2010) Pid controller design of nonlinear systems using an improved particle swarm optimization approach. Commun Nonlinear Sci Numer Simul 15(11):3632–3639
26. Zbigniew M (1996) Genetic algorithms+ data structures= evolution programs. In: Computational statistics, pp. 372–373. Springer
27. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. IEEE Comput Intell Mag 1(4):28–39

28. Atashpaz-Gargari E, Lucas C (2007) Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In: 2007 IEEE congress on evolutionary computation, pp. 4661–4667 . Ieee
29. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
30. Yang X-S (2010) A new metaheuristic bat-inspired algorithm. In: Nature inspired cooperative strategies for optimization (NICSO 2010), pp. 65–74. Springer
31. Storn R, Price K (1997) Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341–359