Robotics and Autonomous Systems 84 (2016) 64–75

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Parameter tuning of PID controller with reactive nature-inspired algorithms

Dušan Fister [a], Iztok Fister Jr. [b,*], Iztok Fister [b], Riko Šafarič [b]

[a] Faculty of Mechanical Engineering, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia
[b] Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia

HIGHLIGHTS

- PSO is the most reactive nature-inspired algorithm among BA, HBA, GA, DE, CS and PSO.
- Population based nature-inspired algorithms (e.g., PSO, BA, HBA, DE and CS) can be used for online implementation of PID parameter tuning.
- Low population sizes in nature-inspired algorithms are sufficient for PID tuning to obtain reactive response of SCARA robot.

ARTICLE INFO

Article history:
Received 25 March 2016
Accepted 19 July 2016
Available online 27 July 2016

Keywords:
PID controller
Stochastic nature-inspired population-based algorithm
Evolutionary algorithms
Swarm intelligence-based algorithms

ABSTRACT

A PID controller is an electrical element for reducing the error value between a desired setpoint and an actual measured process variable. The PID controller operates according to its input parameters, which need to be set before its run. The optimal values of these parameters must be found during the so-called tuning process. Today, this process can be automatized using stochastic, nature-inspired, population-based algorithms, such as evolutionary and swarm intelligence-based algorithms. Unfortunately, these algorithms are too time consuming, and so the reactive, nature-inspired algorithms using a limited number of fitness function evaluations are proposed in this paper. Two reactive evolutionary algorithms (differential evolution and genetic algorithm), and four reactive, swarm intelligence-based algorithms (bat, hybrid bat, particle swarm optimization and cuckoo search), were used to tune the PID controller in our comparative study. Only ten individuals and ten iterations (generations) were used in order to select the most appropriate optimization algorithm for fast tuning of controller parameters. The results were compared using statistical analysis and showed that particle swarm optimization is the best option for such a task.

1. Introduction

A PID controller is an electrical element for reducing an error value between a desired setpoint and an actual process variable. The desired setpoint can be set by a function generator, while the actual process variable is measured by a sensor. A set of input parameters is required for proper controller service. Therefore, the optimal input parameters need to be searched for in a so-called tuning process. Only tuned parameters ensure correct behavior of the electrical and mechanical systems, long-term service, and damage prevention. The PID controller can be described as a closed-loop system, i.e., a system in which the actual process variable has to be controlled. There are many examples of closed-loop systems, such as:

- robot mechanism control,
- temperature control,
- level control,
- direction control, etc.

In this paper, we propose parameter tuning of the PID controller controlling the robot arm mechanism. This arm simulates the movement of a human arm and consists of two joints powered by two motors. This type of robot arm is also referred to as a Selective Compliance Assembly Robot Arm (SCARA) and was designed by Hiroshi Makino in 1980. The structure of the robot arm enables precise positioning in industrial robotics and electronics. Usually, SCARA is accompanied by another motor or hydraulic piston for vertical movement of the robot's top. The main task of the SCARA is to capture objects, manipulate them in 3-D space, and then put

* Corresponding author.
E-mail addresses: dusan.fister@student.um.si (D. Fister), iztok.fister1@um.si (I. Fister Jr.), iztok.fister@um.si (I. Fister), riko.safaric@um.si (R. Šafarič).

http://dx.doi.org/10.1016/j.robot.2016.07.005
0921-8890/

them into another position. We should note that only the positional part of the robot without vertical manipulator was used in our laboratory experiments.

The task of optimization is to search for the optimal input variables by known model and output variables [1]. There are numerous optimization problems that can be divided into many classes, e.g., continuous, numeric, discrete (also combinatorial), multi-objective, constrained, etc. Not all algorithms achieve the same results for all classes of optimization problems. This is in accordance with the No-Free Lunch theorem (NFL) [2], which states that the results of two optimization algorithms are equal when compared to the all classes of problems. In our case, it handles about the relatively simple problem belonging to a class of discrete/numerical problems being solved in four dimensional search space.

Recently, the problem of parameter tuning of the PID controller has been solved using a different stochastic, nature-inspired, population-based algorithms and even computational intelligence algorithms, including fuzzy systems, artificial neural networks, and artificial immune systems [3]. A survey of these algorithms can be found in [4–6]. We propose reactive algorithms, e.g. algorithms that use only little time to converge to the final result. Ten individuals and ten iterations were used to realize, how well reactive algorithms can perform, if they would be applied even for online tuning of controller parameters.

In general, the stochastic, nature-inspired, population-based algorithms are inspired by two aspects of the natural world. The first is Darwinian evolutionary theory [7], whereby only the more adapted individuals can survive in the unforgiving struggle for existence. This inspiration led to the emergence of evolutionary algorithms (EA), where the better solutions, generated by using operators crossover and mutation, can survive and transfer their values in the next generation in the simulated evolution. Alan Turing was the first engineer to incorporate the principles of the natural selection into an algorithm [8] and his first work in artificial intelligence was the *Intelligent Machinery*. Based on Turing's results, John Holland implemented a genetic algorithm (GA) in 1988 which even today remains the most widely-used evolutionary algorithm [9]. Differential evolution (DE), developed by Storn and Price in 1995 [10], was one of the youngest EAs especially suited to continuous global optimization.

The second inspiration for the development of the optimization algorithms emerged in 1995, when a Particle Swarm Optimization (PSO) was developed by Russel Eberhart and James Kennedy [11]. It was based on social relations among individuals in swarm. Many types of biological species have been mimicked since then, including birds, fish, ants, bees, cuckoos, bats, and termites. They all rely on a randomly generated population of particles which are continuously being moved around a search space using variation operators. The velocity of a single particle is calculated for every dimension of the problem and is later added to the appropriate position, thus exploring the problem search space. In 2009, an optimization algorithm called cuckoo search (CS) was developed by Yang and Deb [12] based on the behavior of the cuckoo, which dumps its eggs into random nests. A year later, the bat algorithm (BA), which mimics the phenomenon of echolocation in micro bats, was proposed in 2010 by Yang [13].

It is well known that the stochastic, nature-inspired, population-based algorithms are extremely time consuming for finding near-optimal solutions. The number of fitness function evaluations is the main factor responsible for the time complexity of these algorithms. Typically, it is expressed as a product of population size multiplied by the maximum number of generations. However, for the robot arm operating in an environment, it is important how rapidly a reaction to environmental change is performed. In order to make these kinds of algorithms more reactive, the number of fitness function evaluations needs to be limited. Therefore, these algorithms are used as reactive, nature-inspired algorithms in the study.

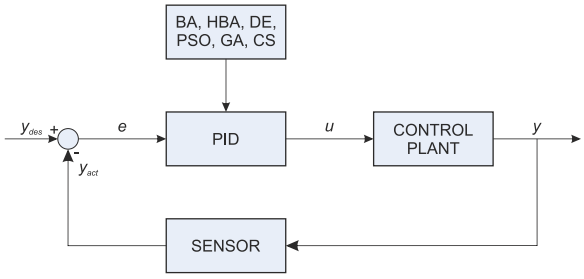


Fig. 1. SCARA robot arm mechanism.

The purpose of this paper is to compare different reactive, nature-inspired algorithms for tuning parameters of the PID controller in order to discover the most suitable algorithm for use in solving this class of problem. The reactive, nature-inspired algorithms such as BA, HBA, PSO, DE, GA, and CS are compared in order to show which of them is most useful in working with small population sizes and small generation numbers.

The remainder of the paper is organized as follows. Section 2 describes the system equipment of a highly nonlinear SCARA robot mechanism, i.e., the computer control hardware and simple PID position controller used during the development and testing of the optimization algorithms. In Section 3, we discuss stochastic, nature-inspired, population-based algorithms. Section 4 deals with a description of the experiments and the results of the nature-inspired algorithms used in the comparative study. The paper concludes with a summary of the work and the suggestions for further development.

2. Description of the 2-DOF SCARA robot arm

Robotics arose from the human desire to supplant human labor with machines for long-running, boring, and even dangerous tasks. Especially in Japan, robots already do housework, while their work on conveyor belts is indispensable in industry. Today, we cannot imagine painting cars without robots. Thus, a robotic arm successfully replaces the human arm and even outdoes it when needed. However, some form of feedback is necessary in order to move the arm in a specific environment. The PID controller is the most common device for using feedback in natural and man-made systems.

In engineering applications, this controller appears in many different forms, i.e., as a stand-alone controller, as a part of distributed systems, or built into embedded systems [14]. A lot of technological changes influenced the development of the controller, in particular the introduction of microprocessors. These provide additional features, such as automatic parameter tuning, gain scheduling, and continuous adaptation. A robotic arm is moved and positioned using a closed-control loop that consists of a PID controller, a control plant, and a sensor. The PID controller is part of a system that controls the electro-mechanical part of the SCARA robotic arm (control plant). The control plant consists of electrical motors to lift and lower the arm. The mechanical part obeys the mechanical laws. The sensor obtains feedback from the control plant (Fig. 1). The input of the PID controller is an error value e, which is transformed into the output signal u, according to Eq. (1), as follows

$$u(t) = K_P \cdot e(t) + \frac{1}{T_i} \cdot \int e(t)\, dt + T_d \cdot \frac{de(t)}{dt}, \tag{1}$$

where $e(t)$ means

$$e(t) = y_{des}(t) - y_{act}(t). \tag{2}$$

From Eq. (1), three gains can be recognized, i.e., proportional (K_P), integral ($\frac{1}{T_i}$) and derivative (T_d). Those three gains represent weights which directly influence the robotic movement. In order to optimize the response of the robot, these three parameters need to be adapted. This type of PID controller is used for analogue control. For SCARA control, Eq. (1) should be transformed into discrete form, presented by Eq. (3). Moreover, a simplification can be done in order to enhance robotic movement, therefore the integrator part ($\frac{1}{T_i} = 0$) should be eliminated. The reason for simplifying the Eq. (1) is hidden in the structure of robot, which contains the natural integrator in the controlled plant. Therefore, it is useless to impose another integrator into control equation. While the first equation uses the analogue operator of time, the second one uses the sample time for the sequencing of sensor readings, as follows:

$$u(k) = q_0 \cdot e(k) - q_1 \cdot e(k-1), \tag{3}$$

where

$$q_0 = K_P + \frac{T_d}{T_s},\ q_1 = \frac{T_d}{T_s}. \tag{4}$$

The $u(k)$ is an input of the control plant, written in discrete form, while the $u(k-1)$ presents the control signal from the previous sample time. The $e(k)$ stands for an error of the actual sample time and $e(k-1)$ for an error from the previous sample time. q_0 and q_1 elements represent the input of PID controller, set by an optimization algorithm. For a discrete z-transform approximation, the relation $s = \frac{1}{T_s}(1 - z^{-1})$ is used, which is allowed to be used, if the sample time T_s is ten times lower than the shortest time constant in control plant.

For the linear control plant, more common tuning methods are applicable. A sample of these would include Bode plotting [15], the root locus method [16] and the Ziegler–Nichols method [14]. Unfortunately, these methods cannot be used on our highly nonlinear control plant. The fact is, when one axis moves, with its torque affects also on the other one, and vice versa. This phenomena, called mechanical coupling, cannot be satisfactory controlled by simple linear control techniques, since the control plant parameters are not constant, but are changed due to changing inertia. Faster the robot arms move, the greater the mechanical coupling affects. Considering maximum possible velocity of robot as our goal, coupling becomes an immense challenge to respect. Therefore, we must rely on either manual (experimental) method, which is very time consuming, or automatic method using an optimization algorithm.

In the remainder of this section, a robot arm mechanism presenting the control plant, and the microprocessor dedicated to tuning its parameters, are described in details.

2.1. Robot arm mechanism

The SCARA robot arm mechanism (see Fig. 2) has two degrees of freedom (2 DOF). Rotation of the first robot link around the first robot axis is presented as the first DOF, and rotation of the second robot link around the second robot axis on the tip of the first link is presented as the second DOF. Both robot links are driven by direct current (DC) motors and gear-boxes with transmission ratios $N_1 = 60/10$ for the first DOF and $N_2 = 173/19$ for the second DOF. By using gear-boxes, the nonlinear inertia influences due to robot links load mass are decreased, but not eliminated (see Eq. (7)). Additional dynamic nonlinearities are brought to the system as viscous and Coulomb friction, which is proportionally large (up to 20% of maximum torque at the nominal speed of both motors) in the SCARA robot mechanism.

Robot links are driven by a DC-motor with the commercial name ESCAP 28D11-219P with nominal voltage of 12 V, nominal current 1.5 A, nominal velocity 600 rd/s, inertia moment $J_m = 17.6 \times 10^{-7}$ kg m^2, viscous friction coefficient $B_m = 1 \times 10^{-7}$ N ms/rd and nominal torque 28.4×10^{-3} N m. The DC-motors used are equipped with sinusoidal incremental encoders. The sinus signals for both motors from the incremental encoders are transmitted by power electronics to 400 pulses per robot joint rotation. Pulses from the incremental encoders are used for the position feedback information of a robot arm joint positions.

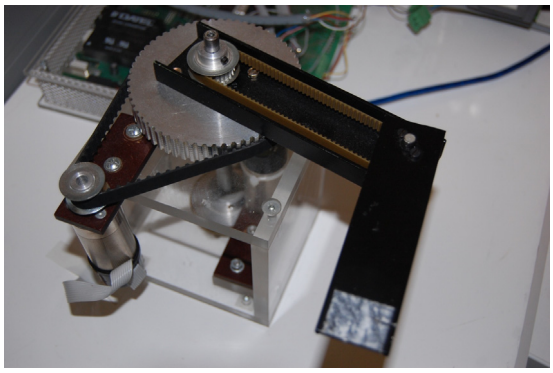


Fig. 2. SCARA robot arm mechanism.

A nonlinear direct dynamic model of the general SCARA robot mechanism is derived from the Lagrange equation of motion, as follows:

$$\mathbf{M}(\boldsymbol{\theta}) \cdot \ddot{\boldsymbol{\theta}} + \mathbf{h}\left(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}\right) = \mathbf{T_m} - \mathbf{F_v}\left(\dot{\boldsymbol{\theta}}\right) - \mathbf{F_C}\left(\dot{\boldsymbol{\theta}}\right) - \mathbf{T_d}, \tag{5}$$

where $\mathbf{M}$ is an inertial matrix, $\mathbf{h}$ is a torque vector due to the centrifugal, centripetal and Coriolis forces and $\mathbf{T_m}$ is a vector of the drive torque applied to the robot's motors. $\mathbf{F_v}$ and $\mathbf{F_C}$ presents viscous friction and Coulomb friction torques, while $\mathbf{T_d}$ stands for a torque vector due to unknown disturbances. $\boldsymbol{\theta}$ presents the vector of robot's arm position, $\dot{\boldsymbol{\theta}}$ the vector of robot's arm velocity and $\ddot{\boldsymbol{\theta}}$ the vector of robot's arm acceleration.

Vectors $\mathbf{T_m}, \boldsymbol{\theta}, \dot{\boldsymbol{\theta}}, \ddot{\boldsymbol{\theta}}, \mathbf{h}$ and matrices $\mathbf{M}, \mathbf{F_v}, \mathbf{F_C}$ for the 2-DOF SCARA robot mechanism are written as:

$$\mathbf{T_m} = \begin{bmatrix} T_{1m} \\ T_{2m} \end{bmatrix}; \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}; \dot{\boldsymbol{\theta}} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}; \ddot{\boldsymbol{\theta}} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix}; \tag{6}$$

$$\mathbf{M}(\boldsymbol{\theta}) = \begin{bmatrix} J_{m1} \cdot N_1 + \dfrac{a_1 + a_2 \cdot \cos\theta_2}{N_1} & \dfrac{a_3 + a_2 \cdot \cos\theta_2}{N_1} \\ \dfrac{a_3 + a_2 \cdot \cos\theta_2}{N_2} & J_{m2} \cdot N_2 + \dfrac{a_3}{N_2} \end{bmatrix}; \tag{7}$$

$$\mathbf{h}(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \begin{bmatrix} \dfrac{-a_2 \cdot (2 \cdot \dot{\theta}_1 \cdot \dot{\theta}_2 + \dot{\theta}_2^2) + \sin\theta_2}{N_1} \\ \dfrac{a_2 \cdot \dot{\theta}_1^2 \cdot \sin\theta_2}{N_2} \end{bmatrix}; \tag{8}$$

$$\mathbf{F_v}(\dot{\boldsymbol{\theta}}) = \begin{bmatrix} B_{m1} \cdot N_1 + B_{GB1} & 0 \\ 0 & B_{m2} \cdot N_1 + B_{GB2} \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}; \tag{9}$$

$$\mathbf{F_C}(\dot{\boldsymbol{\theta}}) = \begin{bmatrix} F_{m1} + F_{GB1} & 0 \\ 0 & F_{m2} + F_{GB2} \end{bmatrix} \cdot \begin{bmatrix} sign(\dot{\theta}_1) \\ sign(\dot{\theta}_2) \end{bmatrix}; \tag{10}$$

where parameters for $i = 1, 2$ are as follows: $J_{m,i}$ inertia moment of DC motors, $B_{m,i}$ viscous friction of the ith motor, $F_{m,i}$ Coulomb friction of the ith motor, $B_{GB,i}$ viscous friction of the ith gear-box, $F_{GB,i}$ Coulomb friction of the ith gear-box, and

$$a_1 = I_{Z1} + I_{Z2} + I_{Z3} + I_{Z4} + m_2 \cdot l_{1T}^2 + (m_3 + m_4) \cdot l_1^2 + m_4 \cdot l_{2T}^2, \tag{11}$$

$$a_2 = m_4 \cdot l_{2T}, \tag{12}$$

$$a_3 = I_{Z4} + m_4 \cdot l_{2T}^2, \tag{13}$$

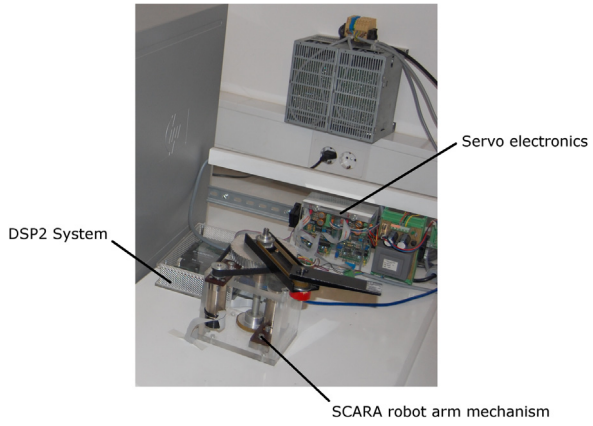


Fig. 3. Hardware implementation hosting the application.

where inertia moments of the links and other elements (bearings, sprockets, belt pulleys) are

$$I_{Z1} = 1.28 \times 10^{-5}\ \text{kg m}^2, \quad I_{Z2} = 1.159 \times 10^{-4}\ \text{kg m}^2,$$
$$I_{Z3} = 1.28 \times 10^{-5}\ \text{kg m}^2, \quad I_{Z4} = 1.28 \times 10^{-5}\ \text{kg m}^2.$$

Here, length of the links are $l_1 = 0.12$ m and $l_2 = 0.095$ m, masses of the links and the other elements are $m_1 = 0.0667$ kg, $m_2 = 0.053$ kg, $m_3 = 0.0737$ kg, $m_4 = 0.024$ kg and distances between the center of gravity and center of rotation of the links are $l_{1T} = 0.057$ m and $l_{2T} = 0.045$ m.

2.2. Computer control system

The developed algorithms consisting of the off-line optimization algorithm for position PID controllers of both axes, the on-line implementation of the control algorithms and the Graphic User Interface (GUI), are being executed on a DSP2 robotic controller card. This DSP2 system is connected via a serial RS-232 bus to a personal computer (PC) and via wires to robot servo electronics. Servo electronics is designed to generate the required current for the DC driving motors of the robot axes and for measuring the position of the robot axes using a combination of incremental encoders. Hardware implementation of the complete set-up is shown in Fig. 3.

The DSP2 system is composed of a DSP2 controller [17] and a DSP2 add-on robotic board. The key component of the DSP2 controller is the high performance floating point digital signal processor (DSP) used for control and optimization algorithm execution. The DSP2 system contains all the necessary peripherals for 4-axes robot control, meaning that it has 16 digital inputs, 8 digital outputs, 4 analogue inputs/outputs and 4 incremental encoder interfaces.

A DSP2 system, connected to a lab PC through the serial port implements a control algorithm, developed by using MATLAB/Simulink. Through the analogue and digital I/O signals, the DSP2 system drives a two-axes SCARA robot (Fig. 2). ComVIEW VI communicates between the PC and the DSP2 system for monitoring signals and tuning parameters.

3. Nature-inspired algorithms

Today, stochastic, nature-inspired, population-based algorithms encompass evolutionary algorithms (EA) and swarm intelligence-based (SI) algorithms. Although both families of algorithms follow different inspirations from nature, a more detailed implementation analysis reveals that they have many characteristics in common. At first glance, algorithms of these two families consist of the following elements:

- initialization of solutions,
- fitness function evaluation,
- replacing the worst solutions,
- generation of new solutions.

In addition to these elements, the termination condition as well as the representation of solutions need to be defined in order to complete these algorithms. Although the implementation details of specific elements (e.g., initialization, evaluation, replacement) do not differ significantly, the generation of new solutions presents the main difference in implementation between these two families of nature-inspired algorithms. While the new solutions are generated using three operators in EAs (parent selection, crossover, and mutation), in general only one operator (move) is used in SI-based algorithms.

The population-based algorithms operate with a population of n vectors of dimension D, where each vector represents a solution to the optimization problem. Generally, the optimization problem is defined as a quadruple $OP = \langle I, S, f, goal \rangle$ [18], where I presents a set of instances resulting from the input, S is a set of feasible solutions, f is an objective function, and *goal* denotes whether the minimum or maximum of the objective function is searched for.

In our case, the input vector denotes the parameters of the simplified PID controller, as follows

$$\mathbf{x} = [q_{1,0}, q_{1,1}, q_{2,0}, q_{2,1}], \tag{14}$$

where $q_{1,0}$ and $q_{1,1}$ are the controller input parameters for the first axis and $q_{2,0}$ and $q_{2,1}$ for the second axis of a robotic manipulator. The task of optimization is to maximize the fitness function, i.e., $max(f_i)$, where the fitness function is evaluated by three different measured values obtained as feedback $\mathbf{y}$ from the control plant, i.e.,

- $Over_i$: actual overshoot,
- Ess_i: actual steady state error and
- $Time_i$: actual settling time.

Feedback from the control plant can be obtained by exciting the robot arm with the reference signal. There are many reference signals that provide different responses from the robot arm, i.e.,

- a step function,
- a trapezoidal function,
- a sine function,
- a sine^2 function.

While the step function enables the hard controlling of the robot arm, the other three functions are devoted to soft control, of which the sine^2 provides the softest.

For our application, step function control was used. Here, the reference signal is raised from zero to one immediately for each time interval of five seconds. The robot arm must respond to this change of the reference signal (Fig. 4).

If the actual value overcomes the reference signal, this indicates an actual overshoot. When the actual signal settles down—arrives to the vicinity of the reference signal and no longer deviates from it, the settling time can be measured. Finally, for our example a steady state error can be detected after five seconds. The reference and actual values are then subtracted and the absolute value of this subtraction yields the result of the steady state error.

This, in turn, results in the following fitness function:

$$f(\mathbf{y}) = \sum_{i=1}^{2} \frac{1}{2}(E_{1i}(1 - |P_i - Over_i|) + E_{2i}(1 - Time_i) + E_{3i}(1 - Ess_i)), \tag{15}$$

where E_{ij} stands for initialized constants representing weights that determine the influence of the specific output for each axis in Eq. (15) and P_i is a desired overshoot, which is in our case set to zero

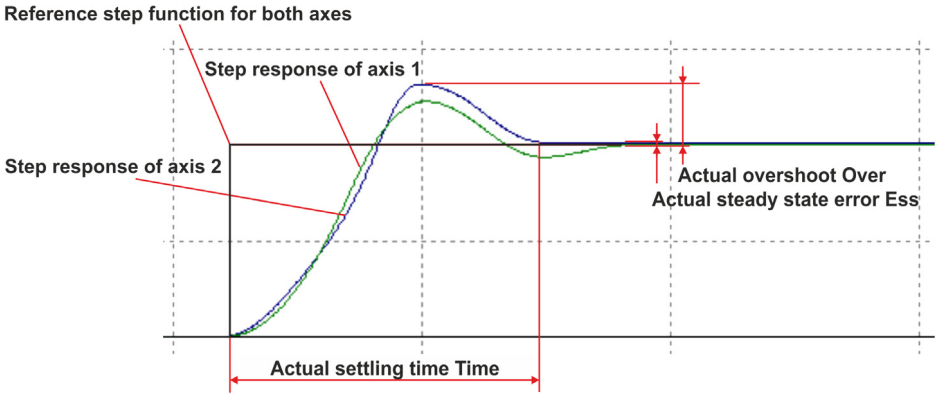


Fig. 4. Step response of SCARA mechanism.

($P_i = 0$). Obviously, the sum of these three constants of specific axis is equal to one, in other words

$$\sum_{i=1}^{3} E_{ij} = 1, \tag{16}$$

where i is the specific output variable and j the specific axis.

The developed nature-inspired algorithms demand a specific population model, because the simplified PID controller runs on the same processor as these algorithms do. As can be seen from Fig. 5, the initialized solutions are saved to a population of trial solutions, while the fitness values of candidate solutions are set to zero. Fitness function evaluation is then performed, and the generated values sent to the simplified PID controller. After running the control loop, the output values are obtained, from which the fitness suitability of trial solutions is calculated. The replace function then compares the fitness values of trial and candidate solutions, and the better of these are preserved for the next generation. In the last step, the variation operator takes the candidate solutions and generates the new solutions.

In the present study, the following stochastic, nature-inspired, population-based algorithms have been implemented:

- bat algorithm,
- hybrid bat algorithm,
- differential evolution,
- particle swarm optimization,
- genetic algorithm,
- cuckoo search.

In the remainder of the paper, the biological foundations of the algorithms are discussed, the generation operators presented, and the pseudo-codes of the implemented stochastic, nature-inspired population-based algorithms illustrated. The section concludes with a summary of the characteristics of these algorithms.

3.1. Bat algorithm

The bat algorithm (BA) was developed by Yang [19] and mimics a behavior of micro-bats that use the phenomenon of echolocation for orientation in the dark. This phenomenon consists of generating an ultrasonic pulse which bounces off obstacles and prey and echoes back to the bat. The bat then calculates its distance to either obstacle or prey. The BA treats bats as a swarm, moving throughout a search space and searching for prey.

Indirectly, the current best bat diverts the whole swarm towards regions rich with food. From the engineer's point of view, more food means higher value of fitness function and a higher quality solution to the problem. Movement of the bats in the search space is governed by a simple mathematical model of echolocation. The modeling process, described in [13], depends on three different vectors: frequency of pulse $Q_i^{(t)}$, velocity $\mathbf{v}_i^{(t)}$ and position $\mathbf{x}_i^{(t+1)}$ of the ith bat at generation t. This movement can be summarized in the following equations:

$$\begin{aligned} Q_i^{(t)} &= Q_{min}^{(t)} + \left(Q_{max}^{(t)} - Q_{min}^{(t)}\right) \cdot \beta, \\ \mathbf{v}_i^{(t+1)} &= \mathbf{v}_i^{(t)} + \left(\mathbf{x}_i^{(t)} - \mathbf{x}_{best}^{(t)}\right) \cdot Q_i^{(t)}, \\ \mathbf{x}_i^{(t+1)} &= \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t)}. \end{aligned} \tag{17}$$

Output pulse frequency can vary in the interval $Q_i^{(t)} \in [Q_{min}, Q_{max}]$. The random number $\beta \in [0, 1]$ specifies the output pulse and $\mathbf{x}_{best}^{(t)}$ presents the current best solution. These equations represent a strategy for exploring the new solutions. Additionally, the algorithm proposes the local search strategy, expressed as follows:

$$\mathbf{x}_{new} = \mathbf{x}_{old} + \epsilon \cdot L(s, \alpha), \tag{18}$$

where $\epsilon > 0$ is the step size scaling factor and $L(s, \alpha)$ the Lévy flight alpha-stable distribution with parameters scale s and exponent $\alpha \in (0, 2]$. The distribution reduces to Gaussian for $\alpha = 2$ and to Cauchy for $\alpha = 1$. This strategy is more exploitative, and represents a kind of random walk that is primarily focused on exploring the vicinity of the current best solution. Both exploration strategies are balanced in the search process using the parameter pulse rate $r_i^{(t)}$.

Algorithm 1 Original Bat algorithm

```
1: INITIALIZE_solutions_randomly;
2: FIND_the_best_solution;
3: while TERMINATION_CONDITION_not_meet do
4:    EVALUATE_each_trial_solution;
5:    REPLACE_worse_solutions_conditionally;
6:    FIND_the_best_solution;
7:    GENERATE_new_trial_solutions;
8:    if rand(0, 1) > r_i then
9:       IMPROVE_the_best_solution_using_Lévy_flight;
10:   end if{local search step}
11: end while
```

The pseudo-code of the BA algorithm is presented in Algorithm 1. Two peculiarities distinguish the BA algorithm from the other stochastic, nature-inspired algorithms: explicit control of exploration/exploitation strategies and conditional replacement of the worst solution. The former is implemented in the 'GENERATE' (exploration strategy in line 7) and 'IMPROVE' (exploitation strategy in line 9) functions, balanced using the parameter pulse rate (line 8), while the later is implemented in the

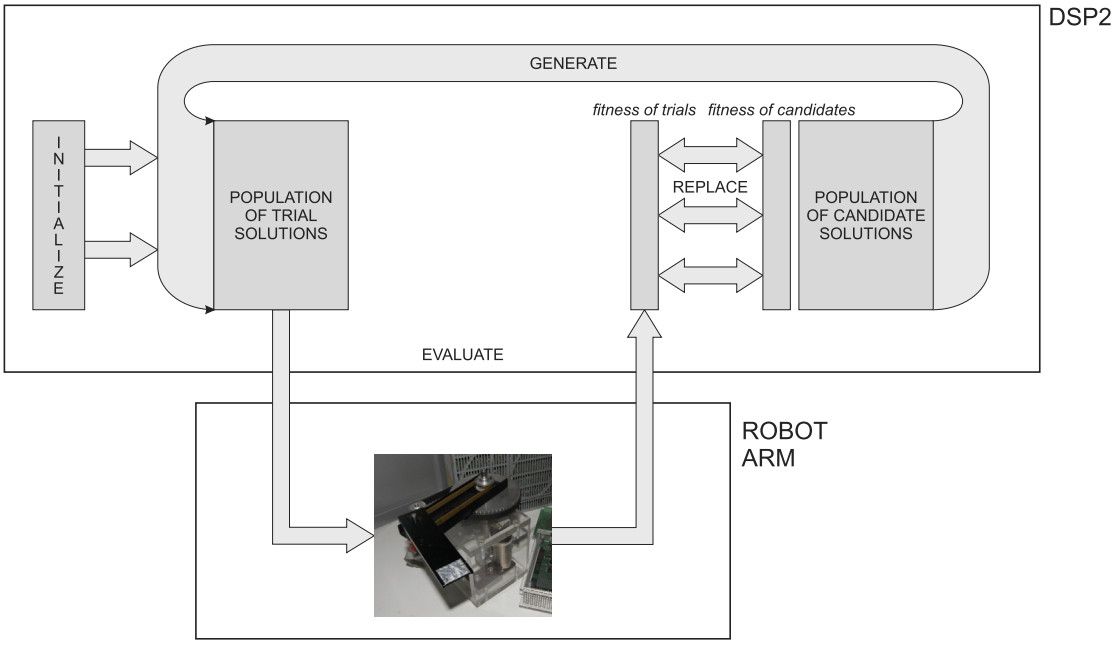


Fig. 5. Population model of the nature-inspired algorithms.

'REPLACE' function (line 5). The motivation behind the conditional replacement of the worse solution is borrowed from a simulated annealing (SA) [20], where the best solution replaces the worst under some probability, thus avoiding becoming stuck in local optima.

3.2. Hybrid bat algorithm

The hybrid bat algorithm (HBA) was developed by Fister Jr. et al. in [21]. In fact, this is a modified version of the BA algorithm, where the random walk improvement strategy was replaced with the 'DE/rand/1/bin' mutation strategy. This strategy increases the exploration power of the original BA algorithm on the one hand, but eliminates the local search improvement strategy on the other.

Algorithm 2 Hybrid bat algorithm

```
1:  INITIALIZE_solutions_randomly;
2:  FIND_the_best_solution;
3:  while TERMINATION_CONDITION_not_meet do
4:     EVALUATE_each_trial_solution;
5:     if rand(0, 1) > r_i then
6:        IMPROVE_the_best_solution_using_"DE/rand/1/bin";
7:     end if{local search step}
8:     REPLACE_worse_solutions_conditionally;
9:     FIND_the_best_solution;
10:    GENERATE_new_solutions;
11: end while
```

The pseudo-code of the HBA algorithm is illustrated in Algorithm 2, from which it can be seen that the only change between the modified and the original algorithm is the call of 'IMPROVE' function in line 9.

3.3. Differential evolution

Differential evolution (DE) is an evolutionary algorithm introduced by Storn and Price in 1995 [10] and is appropriate for continuous and combinatorial optimization. Although it uses metaphors for variation operators similar to typical EAs (i.e., crossover, mutation and selection), they are implemented by mathematical operations and therefore do not follow from any real natural inspiration. DE is a population-based algorithm consisting of n real-coded vectors, where each vector represents a candidate solution.

The variation operator in DE supports a differential mutation and a differential crossover. In particular, the differential mutation randomly selects two solutions and adds the scaled difference between these to the third solution. This mutation can be expressed as follows:

$$\mathbf{u}_i^{(t)} = \mathbf{x}_{r0}^{(t)} + F \cdot (\mathbf{x}_{r1}^{(t)} - \mathbf{x}_{r2}^{(t)}), \quad \text{for } i = 1, \ldots, Np, \tag{19}$$

where F denotes the scaling factor as a positive real number that scales the rate of modification, and $r0$, $r1$, $r2$ are randomly selected values in the interval $[1, n]$. Note that Price and Storn proposed $F \in [0.0, 2.0]$ in the original DE, though the interval $F \in [0.1, 1.0]$ is typically used in the DE community.

A uniform crossover is used in the DE as a differential crossover, where the trial vector is built from parameter values copied from two different solutions. Mathematically, this crossover can be expressed as follows:

$$w_{i,j}^{(t+1)} = \begin{cases} u_{i,j}^{(t)} & \text{rand}_j(0, 1) \leq CR \vee j = j_{rand}, \\ x_{i,j}^{(t)} & \text{otherwise}, \end{cases} \tag{20}$$

where $CR \in [0.0, 1.0]$ controls the fraction of parameters that are copied to the trial solution. Note that the relation $j = j_{rand}$ ensures that the trial vector is different from the original solution $\mathbf{x}_i^{(t)}$.

A differential selection is in fact a generalized one-to-one selection that can be mathematically expressed as follows:

$$\mathbf{x}_i^{(t+1)} = \begin{cases} \mathbf{w}_i^{(t)} & \text{if } f(\mathbf{w}_i^{(t)}) \leq f(\mathbf{x}_i^{(t)}), \\ \mathbf{x}_i^{(t)} & \text{otherwise}. \end{cases} \tag{21}$$

In a technical sense, crossover and mutation can be performed in several ways in differential evolution. Therefore, a specific notation is used to describe the varieties of these methods (also strategies) generally. For example, 'rand/1/bin' denotes that the base vector is randomly selected, 1 vector difference is added to it, and the number of modified parameters in the mutant vector

follows binomial distribution. A detailed description of the other DE mutation strategies, as well as exponential crossover, is found in [22,23].

Algorithm 3 The original DE algorithm

```
INITIALIZE_solutions_randomly;
while TERMINATION_CONDITION_not_meet do
    EVALUATE_each_candidate_solution;
    SELECT_better_between_candidate_and_trial;
    GENERATE_new_trial_solutions;
end while
```

The pseudo-code of a DE algorithm is presented in Algorithm 3, where the 'GENERATE' function implements operator mutation, crossover, and selection according to Eqs. (19)–(21).

3.4. Particle swarm optimization

Particle swarm optimization (PSO) was one of the first members of SI-based algorithm family. It was proposed by Kennedy and Eberhart [24] in 1995. The PSO algorithm mimics the behavior of flocks of birds. Therefore, it is a population-based algorithm, and its population consists of n particles comprised of real-coded elements representing the solution to the problem in question.

The PSO algorithm explores a new solution by moving the particles throughout a search space in the direction of the current best solution. Thus, two sets of particles are managed by the algorithm: the local best solutions $\mathbf{p}_i^{(t)}$ and the current positions of the particles $\mathbf{x}_i^{(t)}$. Moreover, the best solution in the population $\mathbf{g}^{(t)}$ is determined for each generation. The new particle position is generated as follows:

$$\begin{aligned}\mathbf{v}_i^{(t+1)} &= \mathbf{v}_i^{(t)} + C_1 U(0,1)(\mathbf{p}_i^{(t)} - \mathbf{x}_i^{(t)}) \\ &\quad + C_2 U(0,1)(\mathbf{g}^{(t)} - \mathbf{x}_i^{(t)}), \\ \mathbf{x}_i^{(t+1)} &= \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)},\end{aligned} \tag{22}$$

where $U(0, 1)$ denotes a random value drawn from the uniform distribution in interval $[0, 1]$, and C_1 and C_2 are learning factors.

The pseudo-code of the original PSO algorithm is illustrated in Algorithm 4.

Algorithm 4 Pseudo code of the PSO algorithm

```
INITIALIZE_particles_randomly;
while TERMINATION_CONDITION_not_meet do
    EVALUATE_each_particle_solution;
    REPLACE_worse_local_best_solutions;
    FIND_the_global_best_solution;
    GENERATE_new_particle_solutions;
end while
```

We should note that Eq. (22) is implemented in the 'GENERATE' function, while the 'FIND' function determines the current best solution in each generation.

3.5. Genetic algorithm

The genetic algorithm (GA) was one of the first optimization algorithms belonging to the family of evolutionary algorithms (EAs) [25]. This family of algorithms mimics Darwinian evolution [7] through problem solving. According to Darwinian theory, the fittest individuals have a better chance of surviving in the struggle for existence. Similarly, the better solutions have a higher probability of survival and thus a better chance to transfer their characteristics to the next generation during the simulated evolutionary cycle.

In each generation, the solutions undergo operator crossover and mutation [9]. Many operators have been developed since the creation of this algorithm and its application to a variety of problems. Although the original GAs employed a binary representation of solutions, contemporary GAs also support real-coded solutions, and suitable operators were developed in conjunction with this. However, in our study, tournament parent selection with size two, arithmetic crossover, random mutation, and steady-state population model [26] are used.

The pseudo-code of the original GA algorithm is illustrated in Algorithm 5.

Algorithm 5 Genetic algorithm

```
INITIALIZE_solutions_randomly;
while TERMINATION_CONDITION_not_meet do
    EVALUATE_each_trial_solution;
    SELECT_SURVIVOR_solutions;
    SELECT_PARENT_solutions;
    RECOMBINE_pairs_of_parents;
    MUTATE_the_resulting_offspring;
end while
```

As can be seen from Algorithm 5, the generated solutions consist of three functions in the GA algorithm: 'SELECT_PARENT' selects two parents entering the crossover operation, 'RECOMBINE' performs the arithmetic crossover that creates two offspring from two parents by calculating the mean of two corresponding elements, and 'MUTATE' generates a random value from the interval of feasible values for a suitable element. The steady-state population model means that for each generation a portion of the best offspring solutions replaces a portion of the worst parent solutions in the 'SELECT_SURVIVOR' function.

3.6. Cuckoo search

Cuckoo search (CS) is a contemporary stochastic, nature-inspired, population-based algorithm which was developed by Yang and Deb in late 2009 [12]. CS belongs to the SI-based algorithm family [27] inspired by the natural behavior of some cuckoo species along with their obligate brood parasitism. This means that some cuckoo species lay their eggs in the nests of other birds so that the latter will care for them as if they were their own.

A solution in the original cuckoo search algorithm corresponding to cuckoo nests represents a position of the cuckoo nest in the search space. Each nest is evaluated according to the fitness function. The egg can be put in the nest by a local random walk or by simply being ejected from the nest. Being the least fit, the egg has to be built at a random position. The local random walk is primarily intended for exploitation of the current solutions by using Lévy flight distribution expressed, as

$$x_i^{(t+1)} = x_i^{(t)} + \epsilon L(s, \lambda), \tag{23}$$

where the term $L(s, \lambda)$ determines the Lévy flight distribution with scale s and exponent $\lambda \in (0, 2]$ and $\epsilon > 0$ is a step size scaling factor. Notice that this equation is similar to Eq. (18) for the BA algorithm.

Abandoning the old nest and building the new one at a random position represents the global random walk that is intended primarily for exploration of the search space. This is mathematically expressed, as

$$x_i^{(t+1)} = (Ub_j - Lb_j) * U_j(0, 1) + Lb_j, \tag{24}$$

where Lb_j and Ub_j are the lower and upper bounds of the specific variable, respectively, and $U_j(0, 1)$ is a random number drawn from

Table 1
Parameter settings of the algorithms in the study.

BA		HBA		DE		PSO		GA		CS	
Par.	Set	Par.	Set	Par.	Set	Par.	Set	Par.	Set	Par.	Set
Q	[0.5, 1.5]	F	0.9	F	0.9	C_1	1.0	p_c	0.8	α	1.0
β	[0, 1]	CR	0.5	CR	0.5	C_2	1.0	p_m	0.01	λ	1.5
r_i	0.1	r_i	0.1			w	.729	PS	$T = 2$	p_a	0.1
A_i	0.9	A_i	0.9			v	[0, 2]	PM	SS	s	1.0

a uniform distribution in interval [0, 1]. The balance between the global random walk and the local random walk occurs according to the probability p_a.

Algorithm 6 Original Cuckoo search algorithm

```
INITIALIZE_solutions_randomly;
while TERMINATION_CONDITION_not_meet do
    EVALUATE_each_trial_solution;
    REPLACE_worse_solutions_randomly;
    GENERATE_new_trial_solutions;
    if rand(0, 1) < p_a then
        REINITIALIZE_worst_nest;
    end if
end while
```

Interestingly, the CS algorithm employs a 'one-to-random' replacement operator that is somewhat similar to the 'one-to-one' selection operator in DE [10]. However, the randomly selected candidate solution j competes with the ith trial solution for a place in the next generation by means of the 'one-to-random' operator, while in the case of fitness improvement the ith trial solution replaces the corresponding ith candidate solution with the 'one-to-one' operator.

4. Experiments and results

The purpose of our experimental work was twofold: we sought, on the one hand, to show that the quality of results achieved by parameter tuning of the PID controller depends on type of nature-inspired algorithm used, and on the other, to show how the results of parameter tuning improve by increasing the generations. In line with these objectives, three experiments were performed:

- a comparative study of the selected nature-inspired algorithms for parameter tuning of the simplified PID controller,
- a convergence analysis of the selected nature-inspired algorithms for the parameter tuning of the simplified PID controller, and
- an estimation of a time complexity of the nature-inspired algorithms for the parameter tuning of the simplified PID controller.

The algorithms used in our comparative study were: BA, HBA, DE, PSO, GA and CS. The parameter settings of these algorithms as used during the experiment work are illustrated in Table 1. The meaning of the parameters in the table is discussed in Section 3. We should note that the abbreviations *PS* and *PM* in relation to the GA algorithm refer to parent selection and population model, respectively. In fact, the GA for tuning the parameters of the simplified PID controller used the tournament parent selection of a size two and a steady-state (*SS*) population model, where the two best offspring solutions replace the two worst solutions.

Because we worked with reactive nature-inspired algorithms, we used the same small population size $n = 10$ and employed the same small generation numbers $MAX\_GEN = 10$ in order to test their responses. In this way, all algorithms terminate when the maximum number of fitness function evaluations $MAX\_FE = 100$ is achieved. Each algorithm was run ten times to determine its stochastic nature. The obtained values of fitness function for both axes, as well as their mean values, were recorded for each run. In order to restrict the size of the search space, the input parameters captured values from the interval $q_0 \in [0, 400]$ and $q_1 \in [0, 20]$ for each of the observed axes.

In the remainder of the paper, we present the PC configuration used to conduct the experiments. Then, the results of both experiments are illustrated and discussed.

4.1. PC and DSP2 configuration

All runs were carried out on an HP computer, with the following configuration:

- Processor: Intel(R) Core(TM) i5-4570 CPU @ 3.20 GHz,
- RAM: 8 GB,
- Operating system: Windows 8.1, 64-bit.

Configuration of the DSP2 was as follows:

- Processor: high performance floating point — Texas Instruments TMS320C32,
- Instruction cycle time: 33 ns,
- Instructions per second: 30 MIPS.

All tested algorithms were implemented in the MATLAB and Simulink Student Suite-Release 2014a.

4.2. Comparative study

The aim of this study was to illustrate the behavior of six reactive, nature-inspired, population-based algorithms, i.e., BA, HBA, DE, PSO, GA and CS, by tuning the parameters of the simplified PID controller. The best fitness values for both axes obtained for each run after controlling are presented in Table 2, along with the corresponding mean values. Finally, the minimum, maximum, average, and standard deviation values obtained in ten independent runs are presented under "Total". The best results are highlighted in bold.

As can be seen from Table 2, the best results according to the maximum fitness value achieved for the observed algorithms are more or less similar because, with the exception of the PSO algorithm that achieved the maximum fitness value $f_{\max}(\mathbf{y}) = 0.9831$, which is above the interval, these moved within the interval $f_{\max}(\mathbf{y}) \in [0.9747, 0.9788]$. According to the average values, the best results were also obtained by the PSO (i.e., $f(\bar{\mathbf{y}}) = 0.9753$), while the worst results were achieved by the HBA (i.e., $f(\bar{\mathbf{y}}) = 0.9577$). The main reason for this laid is found in the third run, where the HBA obtained the fitness value $f(\mathbf{y}) = 0.9204$. Actually, eliminating the local search improvement strategy from the algorithm negatively affected the results of the tuning process of the SCARA robot.

In order to statistically estimate the quality of the results obtained by the various reactive nature-inspired algorithms for parameter tuning of the simplified PID controller, Friedman tests [28] were conducted. The Friedman test is a two-way analysis of variances by ranks, where the statistical test is conducted in the second step using the calculated rankings. Here, a low rank means

Table 2
Comparison of results.

Run	Fitness	BA	HBA	DE	PSO	GA	CS
1	Axis-1	0.9729	0.9706	0.9754	0.9816	0.9804	0.9670
	Axis-2	0.9726	0.9595	0.9762	0.9846	0.9600	0.9744
	Mean	0.9727	0.9651	0.9758	**0.9831**	0.9702	0.9707
2	Axis-1	0.9795	0.9741	0.9760	0.9445	0.9729	0.9655
	Axis-2	0.9497	0.9368	0.9717	0.9810	0.9556	0.9773
	Mean	0.9646	0.9555	0.9738	0.9627	0.9642	0.9714
3	Axis-1	0.9702	0.9220	0.9777	0.9710	0.9726	0.9690
	Axis-2	0.9847	0.9189	0.9800	0.9893	0.9845	0.9804
	Mean	0.9775	0.9204	**0.9788**	0.9801	**0.9786**	**0.9747**
4	Axis-1	0.9757	0.9574	0.9754	0.9736	0.9455	0.9586
	Axis-2	0.9802	0.9718	0.9681	0.9918	0.9653	0.9807
	Mean	**0.9780**	0.9646	0.9717	0.9827	0.9554	0.9696
5	Axis-1	0.9778	0.9746	0.9722	0.9742	0.9638	0.9615
	Axis-2	0.9592	0.9354	0.9764	0.9750	0.9443	0.9333
	Mean	0.9685	0.9550	0.9743	0.9746	0.9540	0.9474
6	Axis-1	0.9724	0.9584	0.9706	0.9686	0.9782	0.9703
	Axis-2	0.9790	0.9713	0.9526	0.9798	0.9640	0.9559
	Mean	0.9757	0.9648	0.9616	0.9742	0.9711	0.9631
7	Axis-1	0.9746	0.9683	0.9762	0.9805	0.9752	0.9621
	Axis-2	0.9810	0.9327	0.9735	0.9668	0.9727	0.9752
	Mean	0.9778	0.9505	0.9749	0.9737	0.9740	0.9687
8	Axis-1	0.9647	0.9463	0.9797	0.9665	0.9774	0.9485
	Axis-2	0.9887	0.9849	0.9620	0.9865	0.9409	0.9723
	Mean	0.9767	0.9656	0.9709	0.9765	0.9592	0.9604
9	Axis-1	0.9787	0.9839	0.9726	0.9748	0.9721	0.9759
	Axis-2	0.9733	0.9689	0.9745	0.9805	0.9569	0.9624
	Mean	0.9760	**0.9764**	0.9736	0.9777	0.9645	0.9692
10	Axis-1	0.9748	0.9707	0.9810	0.9706	0.9662	0.9769
	Axis-2	0.9430	0.9468	0.9574	0.9651	0.9568	0.9636
	Mean	0.9589	0.9588	0.9692	0.9678	0.9615	0.9703
Total	Min	0.9589	0.9204	0.9616	0.9627	0.9540	0.9474
	Max	0.9780	0.9764	0.9788	**0.9831**	0.9786	0.9747
	Avg	0.9726	0.9577	0.9725	**0.9753**	0.9653	0.9665
	StDev	0.0066	0.0150	0.0047	0.0064	0.0081	0.0079

a better algorithm [29]. The second step is performed only if a null hypothesis from the Friedman test is rejected. Note, the null hypothesis states that the median of the rankings of all algorithms is equal.

According to Demšar [30], the Friedman test is safer and more robust non-parametric test for the comparison of several algorithms over multiple classifiers (also datasets) that, together with the corresponding Nemenyi post-hoc test, allows for a neat presentation of the statistical results [31]. The main drawback of the Friedman test is that it makes all of the multiple comparisons over datasets and therefore it is unable to establish a proper comparison among some of the algorithms considered [29]. Consequently, a Wilcoxon two-paired, non-parametric test is applied as a post-hoc test after determining the control method (i.e., the algorithm with the lowest rank) using the Friedman test. On the other hand, the Nemenyi test is very conservative and may not reveal any difference in most experiments [32]. Therefore, the Nemenyi test is used for graphical representation of the results, while the Wilcoxon test shows which of the algorithms in the test are more powerful. In this study, both tests were conducted using a significance level 0.05.

The results of the statistical tests are illustrated in Fig. 9, which is comprised of two diagrams. The first is a table showing the numerical results of three statistical tests: the Friedman non-parametric test, the Nemenyi, and Wilcoxon non-parametric test. The values were obtained by comparing the best fitness values for each of the observed axes together with their corresponding mean values. As a result, each observed algorithm (i.e., classifier) consists of $3 \times 10 = 30$ values. The second diagram graphically illustrates the results of the Nemenyi post-hoc statistical test.

Table 3
The best input parameters obtained by the observed nature-inspired algorithms.

Alg.	Axis-1		Axis-2		Efficiency [%]
	$q_{1,0}$	$q_{1,1}$	$q_{2,0}$	$q_{2,1}$	
BA	255.47	20.00	102.76	11.70	97.80
HBA	30.70	1.22	154.97	9.47	97.64
DE	215.17	18.01	120.81	7.26	97.88
PSO	151.44	10.89	114.95	6.55	98.31
GA	56.42	3.27	108.57	7.69	97.86
CS	136.55	8.88	89.84	5.81	97.47

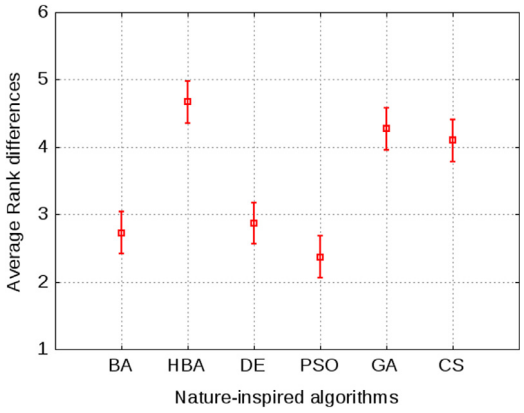
As can be seen from Fig. 6(a), the PSO algorithm achieved the best results due to the minimum ranking value according to the Friedman non-parametric test. Therefore, this algorithm became the control method to which the other algorithms were compared. The control method is denoted by the '‡' sign in the table. The interval in the column 'CD' for the Nemenyi test denotes the confidence interval according to which the significant difference between two algorithms can be determined. As such, two algorithms are considered significantly different if their critical differences (CD) do not overlap. The significant differences are denoted in the table by the '†' sign.

We should point out that both post-hoc statistical tests yielded the same results, where the PSO algorithm (i.e., the control method) significantly outperformed the results of the other two algorithms (i.e., HBA, GA) and CS. Interestingly, the differences between BA, DE and PSO are not significant.

The same conclusions can also be drawn from Fig. 6(b), where two algorithms are considered significantly different, when the lines denoting their critical differences do not overlap. As a

Algs.	Fri.	Nemenyi		Wilcoxon	
		CD	S.	p-value	S.
BA	3.85	[2.42,3.04]		0.09195	
HBA	3.59	[4.36,4.98]	†	6.92E-006	†
DE	3.62	[2.56,3.18]		0.1294	
PSO	3.47	[2.06,2.68]	‡	∞	‡
GA	3.12	[3.96,4.58]	†	0.0009518	†
CS	3.17	[3.79,4.41]	†	0.000345	†

(a) Table.



(b) Graph.

Fig. 6. Statistical analysis of the observed nature-inspired algorithms.

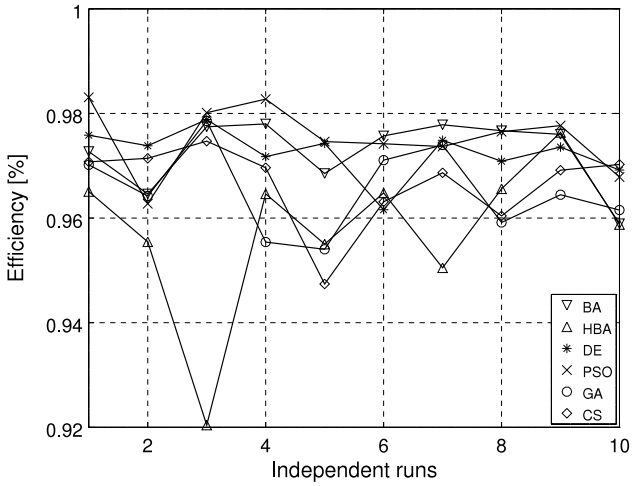


Fig. 7. The best values of various algorithms during the generations.

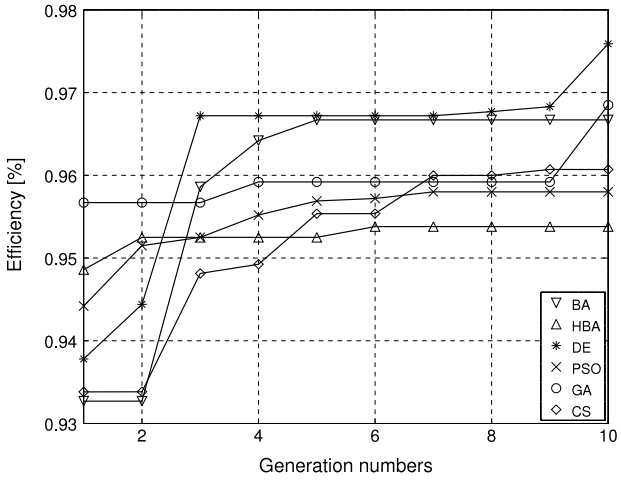


Fig. 8. Convergence graph.

result, the PSO, DE, and BA significantly outperformed the other algorithms (i.e., the HBA, GA and CS) in the test.

Finally, the best input parameters as found by the observed nature-inspired algorithms are aggregated in Table 3. Interestingly, no unique global optima were suggested by the observed reactive algorithms. It seems that searching for the correct ratios between input parameters $q_{1,0}$ and $q_{1,1}$, as well as $q_{2,0}$ and $q_{2,1}$, is more important than a single global optima. As a matter of fact, these ratio values approach the approximate value of $\approx$15.

The best fitness values obtained in each of the ten independent runs are illustrated graphically in Fig. 7.

From Fig. 7 it can be seen that the quality of results obtained by the observed reactive nature-inspired algorithms do not depend on the initial values of the parameters in general. However, an exception to this is the HBA, which obtained the worst result in the third run. Obviously, this algorithm is sensitive to initial conditions. Consequently, the poor initial parameters can cause the algorithm to become stuck in local optima.

4.3. Convergence graph

The convergence graph shows how the best results of the tuning parameters increase by increasing the number of generations with respect to the observed reactive algorithms. This graph also shows how strong the selected parameters (i.e., population size, maximum number of generations, etc.) limit their exploratory power.

The convergence graph capturing the algorithms observed in our study is shown in Fig. 8, from which it can be seen that all the algorithms in the tests improved their initial results significantly, except the HBA, for which a straight curve is shown. In examining the other algorithms in the graph, two kinds of behavior can be detected: first, the GA starts to improve the results obtained in later generations. This means that both the population size and the number of generations should be increased in order to improve the results. On the other hand, the BA converge to the best results very quickly. This fact proves that this algorithm can be suitable in situations where rapid reaction in real-time is demanded from the system.

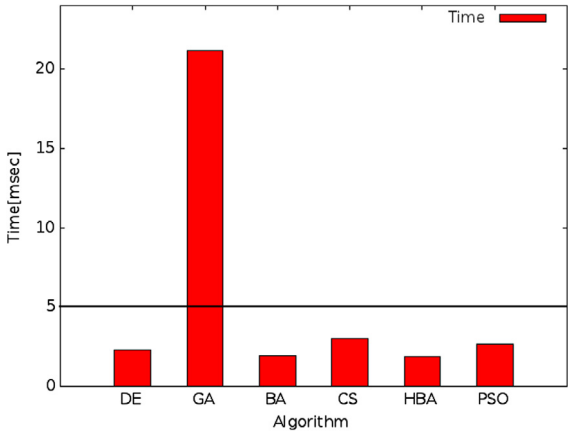
4.4. Time complexity

The purpose of this experiment was to show the time complexity of the stochastic, population-based algorithms used. Thus, we focused on the reactivity of these algorithms. As we know, the online response of the processor must be lower than 5 ms. In line with this, the time complexity of each of the implemented algorithms was measured in milliseconds. Thus, the time duration of the fitness function evaluation was omitted from the measurement. All algorithms were run with parameter settings as presented in Table 1, and used the limited population size $n = 10$ and the limited generation number $MAX\_GEN = 10$. Thus, 10 independent runs were performed for each algorithm on the DSP2 with 30 MIPS.

The results of the experiments are presented in Fig. 9, and are divided into two parts: a table depicting the numerical results and

Algorithm	Time[msec]
DE	2.297
GA	21.128
BA	1.930
CS	3.032
HBA	1.873
PSO	2.640

(a) Numerical results.



(b) Graphical results.

Fig. 9. Time complexity of the nature-inspired algorithms on DSP2 30 MIPS.

a diagram presenting the results graphically. In the diagram, a reference line is drawn that denotes a region where the algorithms having a time complexity under 5 ms are suitable for use in online applications.

As can be seen from the figure, all reactive algorithms except the GA are suitable for online applications on the DSP2 used. Interestingly, the HBA and BA are shown to be the fastest algorithms used in the study. Actually, this fact confirms our assumptions.

5. Conclusion

In general, tuning describes a process of finding the optimal algorithm parameter values before the run. Here, the optimal values of the simplified PID controller are sought. PID controllers are among the most popular control systems used in industry. Their parameters must be tuned in order to avoid instability and delay during system operation. The purpose of tuning their parameters is to find the safety margins in the phase and gain of the PID controller. Although the initial attempts to determine the optimal values of the parameters were performed manually, automatic tuning is used today. This automatic tuning of PID parameters captures the various reactive, nature-inspired, population-based algorithms.

This paper deals with a comparison of the reactive nature-inspired population-based algorithms for tuning the simplified PID controlled parameters. In line with this, six evolutionary and swarm-intelligence algorithms, like BA, HBA, DE, PSO, GA and CS, were taken into consideration. The results of these algorithms were statistically analyzed in order to obtain the best algorithm for this purpose. In this analysis, the online response of the observed algorithms was highlighted. Consequently, the reactive algorithms in tests use the small numbers of generations and the small population sizes. Under these conditions, the best results were achieved by the PSO algorithm. Also the results of the BA and DE algorithms were solid. However, the results obtained for the algorithms, like the GA, reflect the slow convergence rate and therefore demand the higher population sizes.

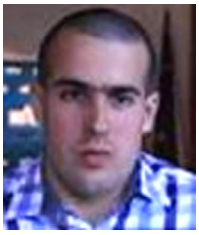
Future research should focus on testing the online response of the PID controller by sudden loads using the reactive, nature-inspired, population-based algorithms. The quick response of the majority of the algorithms observed suggests to us that these algorithms could be successfully applied to these kinds of problems as well.

References

[1] Agoston E. Eiben, John E. Smith, Introduction to Evolutionary Computation, Springer Verlag, London, 2003.
[2] David H. Wolpert, William G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.
[3] Andries P. Engelbrecht, Computational Intelligence: An Introduction, John Whiley & Sons, Ltd., 2007.
[4] Sahu Daya Sagar, Sunil Sharma, A survey paper on PID control system, Int. J. Eng. Trends Technol. (IJETT) 21 (7) (2015) 366–368.
[5] Himanshu B. Patel, Shalaka N. Chaphekar, Developments in PID controllers: Literature survey, Int. J. Eng. Innov. Res. 1 (5) (2012) 425–430.
[6] Soumya Ghosal, Rajkumar Darbar, Biswarup Neogi, Achintya Das, Dewaki N. Tibarewala, Application of swarm intelligence computation techniques in PID controller tuning: A review, in: SureshChandra Satapathy, P.S. Avadhani, Ajith Abraham (Eds.), Proceedings of the International Conference on Information Systems Design and Intelligent Applications 2012 (INDIA 2012), vol. 132, 2012, pp. 95–208.
[7] Charles R. Darwin, On the Origin of Species by Means of Natural Selection, Murray, London, 1871.
[8] Alan M. Turing, Intelligent Machinery, A Heretical Theory, in: The Turing Test: Verbal Behavior as the Hallmark of Intelligence, MIT Press, 1948.
[9] David E. Goldberg, John H. Holland, Genetic algorithms and machine learning, Mach. Learn. 3 (2) (1988) 95–99.
[10] Kenneth Price, Rainer M. Storn, Jouni A. Lampinen, Differential Evolution: A Survey of the State-of-the-Art, Springer-Verlag, Berlin, 2005.
[11] Russ C. Eberhart, James Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the sixth international symposium on micro machine and human science, New York, NY, 1995, pp. 39–43.
[12] Xin-She Yang, Suash Deb, Cuckoo search via Lévy flights, in: World Congress on Nature & Biologically Inspired Computing, 2009. NaBIC 2009, 2009, pp. 210–214.
[13] Xin-She Yang, A new metaheuristic bat-inspired algorithm, in: Nature Inspired Cooperative Strategies for Pptimization (NICSO 2010), Springer Verlag, London, 2010, pp. 65–74.
[14] Karl Johan Aström, Richard M. Murray, Feedback Systems: An Introduction for Scientists and Engineers, Princeton University Press, 2014.
[15] Lee H. Keel, Shankar P. Bhattacharyya, A Bode plot characterization of all stabilizing controllers, IEEE Trans. Automat. Control 55 (11) (2010) 2650–2654.
[16] Walter R. Evans, Control system synthesis by root locus method, Trans. Amer. Inst. Electr. Eng. 69 (1) (1950) 66–69.
[17] DSP2 Web Page. http://www.ro.feri.uni-mb.si/projekti/dsp2, 2015 (accessed 11.15).
[18] Michael R. Garey, David S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman & Co., New York, NY, USA, 1979.
[19] Xin-She Yang, Bat algorithm for multi-objective optimisation, Int. J. Bio-Inspired Comput. 3 (5) (2011) 267–274.
[20] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[21] Iztok Fister Jr., Dušan Fister, Xin-She Yang, A hybrid bat algorithm, Electrotechnical Rev. 80 (1–2) (2013) 1–7.
[22] Das Swagatam, Nagaratnam P. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–31.
[23] Ferrante Neri, Ville Tirronen, Recent advances in differential evolution: A survey and experimental analysis, Artif. Intell. Rev. 33 (1–2) (2010) 61–106.

[24] James Kennedy, Russell Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[25] Thomas Bäck, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms, Oxford University Press, London, UK, 1996.
[26] Deb Kalyanmoy, Multi-Objective Optimization Using Evolutionary Algorithms, John Wiley & Sons, Inc., New York, NY, USA, 2003.
[27] Iztok Fister Jr., Xin-She Yang, Iztok Fister, Janez Brest, Dušan Fister, A brief review of nature-inspired algorithms for optimization, Electrotehnical Rev. 80 (3) (2013) 116–122.
[28] Milton Friedman, A comparison of alternative tests of significance for the problem of m rankings, Ann. Math. Stat., American Statistical Association 11 (1940) 86–92.
[29] Joaquín Derrac, Salvador García, Daniel Molina, Francisco Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (1) (2011) 3–18.
[30] Janez Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[31] Peter B. Nemenyi, Distribution-free multiple comparisons (Ph.D. thesis), Princeton University, 1963.
[32] Salvador García, Francisco Herrera, An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, J. Mach. Learn. Res. (2008) 2677–2694.

Dušan Fister received his B.Sc. in 2015. Currently, he is working towards his M.Sc. degree. His research activities encompass robotics, swarm intelligence and pervasive computing.



Iztok Fister Jr. received his B.Sc. and M.Sc. from Computer Science in 2011 and 2013, respectively. Currently, he is working towards his Ph.D degree. His research activities encompasses swarm intelligence, pervasive computing and programming languages.

Iztok Fister received his Ph.D. degree from the Faculty of Electrical Engineering and Computer Science, University of Maribor, in 2007. He is Assistant Professor at the University of Maribor and member of the Institute of Electrical and Electronics Engineers (IEEE). He is editorial board member of SpringerPlus. His research includes program languages, operational researches, evolutionary algorithms and swarm intelligence, and has been covered by Swarm and evolutionary computation, Applied Soft Computing, International journal of bio-inspired computation, Nonlinear dynamics, Applied mathematics and computation, Journal of heuristics, Neurocomputing, Expert systems with applications and Chaos, solitons and fractals.

Riko Šafarič was born in 1961 and received a B.Sc. degree in 1985, an M.Sc. degree in 1989 and a Ph.D. degree in 1994, all in electrical engineering from the University of Maribor in Slovenia. He holds the position of Full Professor on Faculty of Electrical Engineering and Computer Sciences, University of Maribor, Slovenia. His research work is concerned with the nanorobotics, theoretical and experimental study of neural networks, and fuzzy and genetic algorithms for the control of robot mechanisms.