FACTA UNIVERSITATIS
Series: **Mechanical Engineering** Vol. 21, Nº 4, 2023, pp. 615 - 630
https://doi.org/10.22190/FUME231011044Z

Original scientific paper

Q-LEARNING, POLICY ITERATION AND ACTOR-CRITIC REINFORCEMENT LEARNING COMBINED WITH METAHEURISTIC ALGORITHMS IN SERVO SYSTEM CONTROL

Iuliu Alexandru Zamfirache[1], Radu-Emil Precup[1,2], Emil M. Petriu[3]

[1]Politehnica University of Timisoara, Department of Automation and Applied Informatics, Timisoara, Romania
[2]Romanian Academy – Timisoara Branch, Center for Fundamental and Advanced Technical Research, Timisoara, Romania
[3]University of Ottawa, School of Electrical Engineering and Computer Science, Ottawa, Canada

Abstract. *This paper carries out the performance analysis of three control system structures and approaches, which combine Reinforcement Learning (RL) and Metaheuristic Algorithms (MAs) as representative optimization algorithms. In the first approach, the Gravitational Search Algorithm (GSA) is employed to initialize the parameters (weights and biases) of the Neural Networks (NNs) involved in Deep Q-Learning by replacing the traditional way of initializing the NNs based on random generated values. In the second approach, the Grey Wolf Optimizer (GWO) algorithm is employed to train the policy NN in Policy Iteration RL-based control. In the third approach, the GWO algorithm is employed as a critic in an Actor-Critic framework, and used to evaluate the performance of the actor NN. The goal of this paper is to analyze all three RL-based control approaches, aiming to determine which one represents the best fit for solving the proposed control optimization problem. The performance analysis is based on non-parametric statistical tests conducted on the data obtained from real-time experimental results specific to nonlinear servo system position control.*

Key words: *Reinforcement Learning, Policy Iteration, Actor-Critic, Q-learning, Gravitational Search Algorithm, Grey Wolf Optimizer*

Received: October 11, 2023 / Accepted November 25, 2023
Corresponding author: Radu-Emil Precup
Politehnica University of Timisoara, Department of Automation and Applied Informatics, Bd. V. Parvan 2, 300223 Timisoara, Romania
Romanian Academy – Timisoara Branch, Center for Fundamental and Advanced Technical Research, Bd. Mihai Viteazu 24, 300223 Timisoara, Romania
E-mail: radu.precup@aut.upt.ro

© 2023 by University of Niš, Serbia | Creative Commons License: CC BY-NC-ND

1. INTRODUCTION

The gap between the Machine Learning (ML) and the Automatic Control (AC) fields is becoming a highly interesting research subject in recent years. The main reason behind this interest is the lack of automation in identifying the optimal parameters of controllers specific to AC systems. If the parameters of the controllers are not tuned systematically, one of the main shortcomings is their manual tuning, where a lot of experience is needed, but it is difficult to acquire it if no accurate process models are available or few information on the process is available as well. The optimal values of controller parameters are usually found through manual experiments and observations which take time and other resources. This leads to the need to conduct automatically the optimal tuning of controller parameters. The goal is to start the control and, based on the feedback collected from the controlled process, to automatically adjust (namely tune) the controller parameters so that the process achieves some predefined objective assessed in terms of adequately defined performance indices involved in performance specifications.

Reinforcement Learning (RL) has been considered initially to belong to the field of ML, and it uses only information from interaction with the environment where this technique actually operates. As highlighted in [1], this specific feature also makes ML belong to the field of data-driven model-free control as part of AC. The general RL problem is usually formulated using the mathematical framework of Markov Decision Processes (MDPs) to solve an optimization problem that specifies the performance specifications, and it makes use of Dynamic Programming (DP) to solve that optimization problem at hand. RL operates with agents, which carry out actions in the environment; using the received reward signal (measuring the immediate effect of RL agent's actions), the RL agents adjust their knowledge about themselves and the environment. As shown in [2], by applying this process incrementally, the RL agents will become better in setting actions that maximize or minimize the rewards. The techniques specific to RL represent good candidates to solve optimal reference tracking problems [3] and fill the gap between ML and AC. In this context, the RL agent is included in a control system, i.e. the RL plays the role of a controller, it automatically learns how to change the values of its parameters in controlling a process using the feedback (or reward) received from the controlled process [4]. Besides the agent, the environment and the reward signal, the other important elements of an RL problem are the policy function used by the RL agent to generate actions to be executed on the environment, and the value function that measures the long term effect of the actions.

Based on the policy function, three types of RL agents are generally used leading to three different RL-based control system structures: Value-based, Policy-based and Actor-Critic. The value-based agent iteratively learns a value function in terms of the technique called Value Iteration (VI) in RL, and stores the value of the state vector or the state-action pair. A concrete implementation of the VI approach is the Q-learning technique described in this paper where the goal is to iteratively find the Q-function, i.e. the value function. The policy-based agent learns a policy function which is incrementally improved and used to generate actions, a technique is known as Policy Iteration (PI) in RL. The Actor-Critic agent learns both the value function (called the critic) and the policy function (called the actor). As shown in [2], all types of RL agents contain a policy function to define their behavior.

Deep Reinforcement Learning (DRL) is obtained as a combination of RL and Deep Learning (DL) [2]. One version of DRL-based control system structure treated in this paper is Deep Q-Learning (DQL) obtained by merging the DL and the Q-learning algorithm that belongs to the Value-based structure [5]. The second RL structure treated in this paper is the Policy-based structure, namely the PI RL-based control. The third RL structure used for comparison in this paper is the Actor-Critic framework, which represents a combination of DQL and PI.

All these RL-based structures make use of Neural Networks (NNs) and their orientation to control leads to NN-based control system structures. Metaheuristic Algorithms (MAs) are optimization algorithms that have been employed recently in order to mitigate the two major drawbacks of the widely used Gradient Descent (GD) algorithm to train NNs, namely slow convergence which often leads to relatively poor control system performance and impossibility to avoid local optima which can lead to unstable and unreliable systems. This is the motivation of the research results presented in this paper.

The most popular MAs, also called nature-inspired optimization algorithms, involved in AC and RL problems are the Gravitational Search Algorithm (GSA), the Particle Swarm Optimization (PSO) and the Grey Wolf Optimizer (GWO) algorithms. GSA, PSO and GWO are also viewed as swarm intelligence algorithms. All these algorithms are used in [6] to carry out the optimal tuning of fuzzy controller parameters. An analysis on the usage of MAs in the optimal control of industrial applications is conducted in [7].

A brief discussion on the combinations of MAs and RL in the close relation to NN training is given as follows. Genetic Algorithms (GAs) are presented in [8] and [9] as an alternative to the GD algorithm-based training of NNs, and other GAs and other MAs are reported in [10].

The hyperparameters involved in RL are optimally tuned in [11] via PSO. The PSO algorithm is combined with RL in [12] to train the critic in a fault tolerant tracking Actor-Critic RL-based control system structure, in [13] to design fault tolerant AC of nonlinear processes, to design fuzzy interpretable RL policies in [14], to solve noisy optimization problems in [15], to build a swarm of RL agents which work together for better performance in [16], and to find optimal actions in [17]. A Q-learning is applied in [18] in combination with PSO to design optimal paths for mobile robots, in [19] to improve the performance of positioning strategies for underwater vehicles, and in [20] to enhance the quality of predictions of ground responses with respect to tunneling. GWO is employed in training feed-forward multi-layered NNs in [21]. DQL is combined with a GSA and compared to GWO and PSO algorithms in carrying out the NN-based optimal reference tracking control of servo systems in [22]. PI RL is combined with a GWO algorithm and compared to a PSO algorithm in the same NN-based Optimal Reference Tracking Control Problem (ORTCP) in [23]. A combination of Actor-Critic RL and GWO, aiming to build a control structure where the progress of the NN-based actor is guided by a GWO-based critic as provided in [24]. The approaches presented in [22-24] are accompanied by comparisons with the classical GD algorithm.

Building upon the short discussion of the state-of-the-art presented above and on the authors' results given in [22-24], this paper carries out the performance analysis of three fresh NN- and RL-based control system structures and approaches, (i), (ii) and (iii), which combine RL and MAs. *The approach (i)*: the MA is employed to initialize the parameters (namely, the weights and the biases) of the NNs involved in DQL by replacing the traditional way of initializing the NNs based on random generated values. *The approach (ii)*: the MA is employed to train the policy NN in PI RL-based control. *The approach (iii)*: the MA is employed to act as a critic in an Actor-Critic setup, guiding the progress of the NN-based actor. This contribution is advantageous and important in the context of the literature discussed above as the discussed structures and approaches ensure the relatively simple and transparent mitigation of the GD algorithm ensuring performance enhancement. Seven combined algorithms are discussed, namely GSA, GWO and PSO in (i) and GWO and PSO in (ii) and (iii). These algorithms are also compared to the popular GD algorithm. The comparison is based on the real-time results and data obtained from experiments concerning the position control of a nonlinear servo system, and all these experiments were conducted in the authors' laboratories.

The authors avoided using too many abbreviations, which makes the readability of the paper poor. However, several abbreviations are used in order to keep a reasonable length of the paper.

The paper treats the following topics: the RL-based control problems and the control system structures are described in the next section and the combined RL-MAs are briefly presented in Section 3. The performance comparison expressed in terms of non-parametric statistical tests conducted on experimental results is given in Section 4, and the concluding remarks are pointed out in Section 5.

2. Reinforcement Learning-Based Control Problems and Control System Structures

Both in RL and in control theory, in order for an agent to learn how to automatically control a system, it needs to come up with a solution to an optimization problem. The three approaches (i), (ii) and (iii) mentioned in the previous section will be briefly treated as follows in sections 2.1, 2.2 and 2.3, respectively.

2.1. Optimal reference tracking control problem in the first approach

This approach, (i), focused on DQL, is expressed as the optimization problem [22]

$$\mathbf{a}^* = \arg\min_{\mathbf{a}\in D_{\mathbf{a}}} J_1(\mathbf{a}), \tag{1}$$

where **a** is the vector (represented as a column matrix) that contains the sequence of actions on all iterations

$$\mathbf{a} = [a(1)\ldots a(t_d)\ldots a(t_{\max})]^T \in \Re^{t_{\max}}, \tag{2}$$

$t_d = 1\ldots t_{\max}$ is the discrete time moment, $t_{\max}$ is the maximum number of iterations, $J_1(\mathbf{a})$ is the cost function, $\mathbf{a}^*$ is the optimal value of **a**, namely the solution to the optimization problem defined in Eq. (1), it consists of the optimal sequence of elements, i.e. actions $a^*(t_d)$, $t_d = 1\ldots t_{\max}$:

$$\mathbf{a}^* = [a^*(1)\ldots a^*(t_d)\ldots a^*(t_{\max})]^T \in \Re^{t_{\max}}, \tag{3}$$

and $D_{\mathbf{a}}$ is the feasible domain of **a**.

The definition of the cost function in Eq. (1), which imposes the performance specifications as the aggregate objective of carrying out a tradeoff to small overshoot and small settling time values, is [22]

$$J_1(\mathbf{a}(t_d)) = | e(t_d, \mathbf{a}(t_d)) | + \mu_o \, | e(t_d, \mathbf{a}(t_d)) - e(t_d - 1, \mathbf{a}(t_d)) |, \tag{4}$$

where μ_o is the weight parameter involved in the tradeoff.

The DQL-based control system structure is illustrated in Fig. 1. According to Fig. 1, which is actually a state feedback control system structure, the RL agent generates control actions $a(t_d)$ (they are control signals $u(t_d) = a(t_d)$ in the AC formulation) based on the new state (vector) of the controlled process $\mathbf{s}(t_d + 1)$ and on the reward $r(t_d)$ (it is the controlled output $y(t_d) = r(t_d)$ in the AC formulation), reflected in the control error $e(t_d)$, which depends on y_d – the desired output (or the reference input or the set-point) assumed to be constant. The new control signals or actions will be applied to the process until $J_1(\mathbf{a})$ is minimized, i.e. $y(t_d)$ tracks y_d in terms of Eqs. (1-4). As shown in Fig. 1, the RL agent represents actually the controller in the AC formulation, the environment is the controlled process in the AC formulation, the disturbance input is absent to simplify the problem setting, and the Q-function NN with the nonlinear map Q_w is referred to as the Q-function NN Q_w.

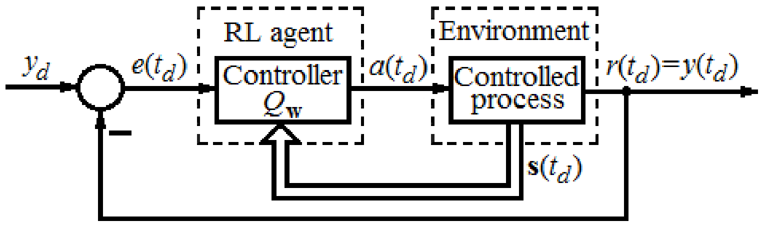


Fig. 1 Informational structure of DQL-based CS [22]

2.2 Optimal reference tracking control problem in the second approach

This approach, (ii), focused on PI RL, is expressed as the optimization problem [23]

$$\boldsymbol{\theta}^* = \arg\min_{\rho \in D_\theta} J_1(\boldsymbol{\theta}), \tag{5}$$

where $\boldsymbol{\theta}$ is the parameter vector used to build the control policy NN, $\boldsymbol{\theta}^*$ is the solution to the optimization problem defined in Eq. (5), namely the optimal value of $\boldsymbol{\theta}$, D_θ is the feasible domain of $\boldsymbol{\theta}$, and $J_1(\boldsymbol{\theta})$ is the cost function

$$J_1(\boldsymbol{\theta}) = | e(t_d, \boldsymbol{\theta}) | + \mu_o \, | e(t_d, \boldsymbol{\theta}) - e(t_d - 1, \boldsymbol{\theta}) |, \tag{6}$$

which is similar to that defined in Eq. (4), thus justifying the fair comparison of the two approaches implemented using different algorithms.

The PI RL-based control system structure is illustrated in Fig. 2, which is similar to the structure in Fig. 1. The integrator with the additional state variable $s_I(t_d)$ is inserted in order to guarantee zero steady-state control errors in certain regimes.

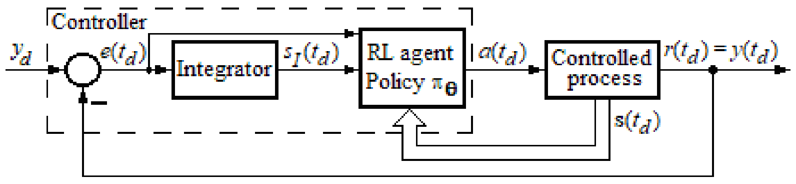


Fig. 2 Informational structure of PI RL-based CS (adapted from [23])

The notation π_θ is used in Fig. 2 for the nonlinear map function of the policy NN. The rest of elements in Fig. 2 are described in relation with the control system given in Fig. 1.

2.3. Optimal reference tracking control problem in the third approach

This approach, (iii), focused on Actor-Critic RL, is expressed as the optimization problem [24]

$$\boldsymbol{\omega}^* = \arg\min_{\boldsymbol{\omega}\in D_\omega} J_1(\boldsymbol{\omega}), \tag{7}$$

where $\boldsymbol{\omega}$ represents the configuration, i.e. the parameter vector of the NN-based actor, $\boldsymbol{\omega}^*$ is the optimal value of $\boldsymbol{\omega}$ and the solution to the problem described in Eq. (7), D_ω is the feasible domain of $\boldsymbol{\omega}$, and $J_1(\boldsymbol{\omega})$ is the cost function, which is defined similarly to Eqs. (4-6), leading to a fair comparison between this approach and the ones discussed in sections 2.1 and 2.2, as

$$J_1(\boldsymbol{\omega}) = | e_t(\boldsymbol{\omega}) | + \mu_o \cdot | e_t(\boldsymbol{\omega}) - e_{t-1}(\boldsymbol{\omega}) |, \tag{8}$$

where the parameter μ_o is used to control (weight) the overshoot value.

Fig. 3 presents the Actor-Critic RL-based control system structure. The integrator is used for the same reason as the on described in Fig. 2. The GWO-based critic is constantly monitoring the NN-based actor and only step in if the actor deviates from achieving the predefined control objectives.

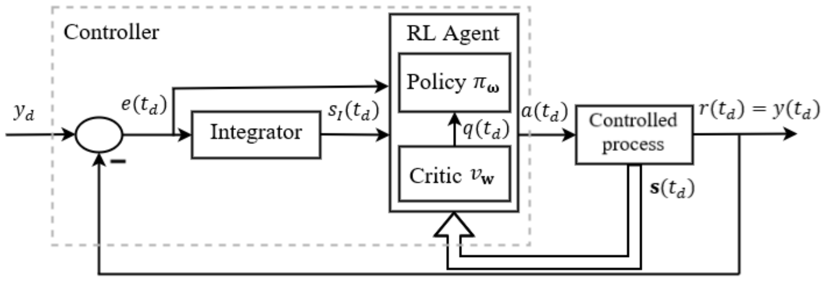


Fig. 3 Informational structure of Actor-Critic RL-based CS (adapted from [24])

The NN-based actor is represented in Fig. 3 as π_ω and the GWO-based critic as v_w. The rest of elements in Fig. 3 are described in relation with the control systems given in Fig. 1 and Fig. 2.

3. Combined Reinforcement Learning-Metaheuristic Algorithms

Using the presentation in [22] focusing on GSA, *the steps of the DQL-based control approach combined with GSA* are presented in Fig. 4.

The NN architecture used for to implement the NNs involved in the algorithm is presented in Fig. 5. The dynamic regime considered to solve the optimization problem defined in Eq. (1) is set to zero external disturbances and constant reference input. The optimization process will stop when the control error does not decrease and its absolute value is under a threshold value for a certain number of consecutive steps.

The NNs involved in the RL-based control process are initialized with all the MAs considered in [22]. Fig. 4 highlights the case where the parameter vector $\mathbf{w}_0$ of the Q-function NN Q_{w} and the parameter vector $\boldsymbol{\rho}_0$ of the target NN $\hat{Q}_{\boldsymbol{\rho}}$ are initialized with GSA. The DQL algorithm is using the experience replay buffer technique to introduce diversity in the training process [22].

Using the presentation in [23] focusing on GWO, *the steps of the PI RL-based control approach combined with MAs* are presented in Fig. 6 using the description given in [23] from a control perspective.

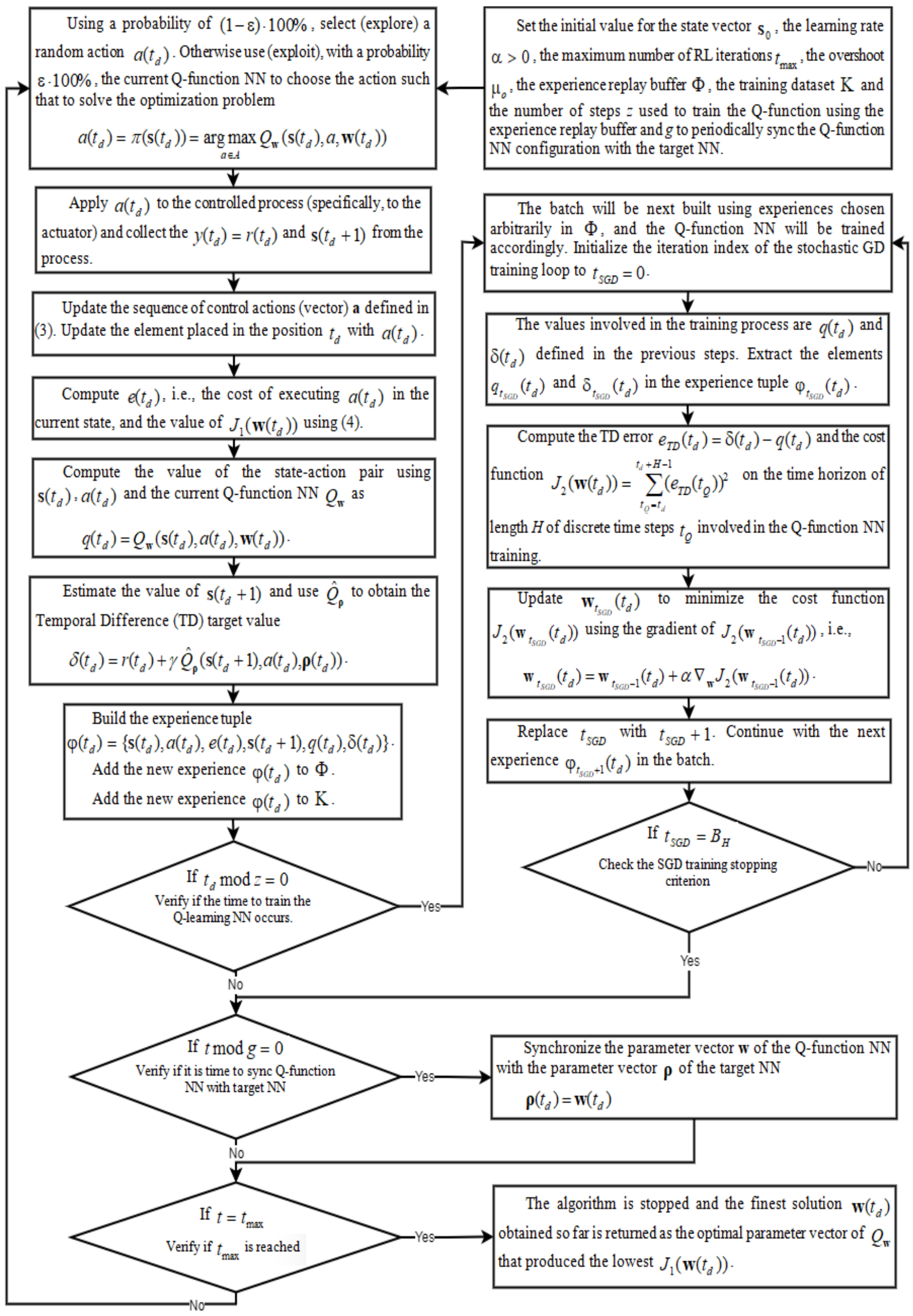
The algorithm presented in Fig. 6 iteratively searches for better configurations, i.e. the parameter vector $\boldsymbol{\theta}$, for the control policy NN $\pi_{\boldsymbol{\theta}}$. The algorithm stops when the CP's output continually follows the reference input for a number of λ steps. More details about how the algorithm is implemented are provided in [23].

Using the presentation in [24] focusing on GWO, *the steps of the Actor-Critic RL-based control approach combined with MAs* are presented in Fig. 7.

The GWO-based critic will step in only if the actor NN deviates from following the predefined optimal control objectives. It will reconfigure the actor NN if the control error value will not decrease during the RL-based control process. The algorithm will stop when the final iteration in the RL process is reached.

4. Experimental Results and Their Discussion

This section presents the experimental results for the GSA-based DQL (GSA-DQL) algorithm, the GWO-based DQL (GWO-DQL) algorithm, the PSO-based DQL (PSO-DQL) algorithm based on (i) and suggested in [22], the GD-based DQL (GD-DQL) algorithm, the GWO-based PI RL (GWO-PI RL) algorithm suggested in [23], the PSO-based PI RL (PSO-PI RL) algorithm suggested in [17] and also applied in [23] and the GD-based PI RL (GD-PI RL) algorithm. This section also describes the results of the GWO-based Actor-Critic (GWO-AC), the PSO-based Actor-Critic (PSO-AC) and the GD-based Actor-Critic (GD-AC) algorithms discussed in [24]. Details on the design of the metaheuristic algorithms are given in [22-24].



Set the initial value for the state vector $\mathbf{s}_0$, the learning rate $\alpha > 0$, the maximum number of RL iterations $t_{\max}$, the overshoot μ_o, the experience replay buffer Φ, the training dataset K and the number of steps z used to train the Q-function using the experience replay buffer and g to periodically sync the Q-function NN configuration with the target NN.

Using a probability of $(1-\varepsilon)\cdot 100\%$, select (explore) a random action $a(t_d)$. Otherwise use (exploit), with a probability $\varepsilon \cdot 100\%$, the current Q-function NN to choose the action such that to solve the optimization problem

$$a(t_d) = \pi(\mathbf{s}(t_d)) = \arg\max_{a \in A} Q_{\mathbf{w}}(\mathbf{s}(t_d), a, \mathbf{w}(t_d))$$

Apply $a(t_d)$ to the controlled process (specifically, to the actuator) and collect the $y(t_d) = r(t_d)$ and $\mathbf{s}(t_d+1)$ from the process.

Update the sequence of control actions (vector) $\mathbf{a}$ defined in (3). Update the element placed in the position t_d with $a(t_d)$.

Compute $e(t_d)$, i.e., the cost of executing $a(t_d)$ in the current state, and the value of $J_1(\mathbf{w}(t_d))$ using (4).

Compute the value of the state-action pair using $\mathbf{s}(t_d)$, $a(t_d)$ and the current Q-function NN $Q_{\mathbf{w}}$ as

$$q(t_d) = Q_{\mathbf{w}}(\mathbf{s}(t_d), a(t_d), \mathbf{w}(t_d)).$$

Estimate the value of $\mathbf{s}(t_d+1)$ and use $\hat{Q}_{\boldsymbol{\rho}}$ to obtain the Temporal Difference (TD) target value

$$\delta(t_d) = r(t_d) + \gamma \hat{Q}_{\boldsymbol{\rho}}(\mathbf{s}(t_d+1), a(t_d), \boldsymbol{\rho}(t_d)).$$

Build the experience tuple
$\varphi(t_d) = \{\mathbf{s}(t_d), a(t_d), e(t_d), \mathbf{s}(t_d+1), q(t_d), \delta(t_d)\}$.
Add the new experience $\varphi(t_d)$ to Φ.
Add the new experience $\varphi(t_d)$ to K.

If $t_d \bmod z = 0$
Verify if the time to train the Q-learning NN occurs.

The batch will be next built using experiences chosen arbitrarily in Φ, and the Q-function NN will be trained accordingly. Initialize the iteration index of the stochastic GD training loop to $t_{SGD} = 0$.

The values involved in the training process are $q(t_d)$ and $\delta(t_d)$ defined in the previous steps. Extract the elements $q_{t_{SGD}}(t_d)$ and $\delta_{t_{SGD}}(t_d)$ in the experience tuple $\varphi_{t_{SGD}}(t_d)$.

Compute the TD error $e_{TD}(t_d) = \delta(t_d) - q(t_d)$ and the cost function $J_2(\mathbf{w}(t_d)) = \sum_{t_Q = t_d}^{t_d+H-1} (e_{TD}(t_Q))^2$ on the time horizon of length H of discrete time steps t_Q involved in the Q-function NN training.

Update $\mathbf{w}_{t_{SGD}}(t_d)$ to minimize the cost function $J_2(\mathbf{w}_{t_{SGD}}(t_d))$ using the gradient of $J_2(\mathbf{w}_{t_{SGD}-1}(t_d))$, i.e.,

$$\mathbf{w}_{t_{SGD}}(t_d) = \mathbf{w}_{t_{SGD}-1}(t_d) + \alpha \nabla_{\mathbf{w}} J_2(\mathbf{w}_{t_{SGD}-1}(t_d)).$$

Replace t_{SGD} with $t_{SGD}+1$. Continue with the next experience $\varphi_{t_{SGD}+1}(t_d)$ in the batch.

If $t_{SGD} = B_H$
Check the SGD training stopping criterion

If $t \bmod g = 0$
Verify if it is time to sync Q-function NN with target NN

Synchronize the parameter vector $\mathbf{w}$ of the Q-function NN with the parameter vector $\boldsymbol{\rho}$ of the target NN

$$\boldsymbol{\rho}(t_d) = \mathbf{w}(t_d)$$

If $t = t_{\max}$
Verify if $t_{\max}$ is reached

The algorithm is stopped and the finest solution $\mathbf{w}(t_d)$ obtained so far is returned as the optimal parameter vector of $Q_{\mathbf{w}}$ that produced the lowest $J_1(\mathbf{w}(t_d))$.

Fig. 4 Flow diagram of the combination of Q-learning and GSA

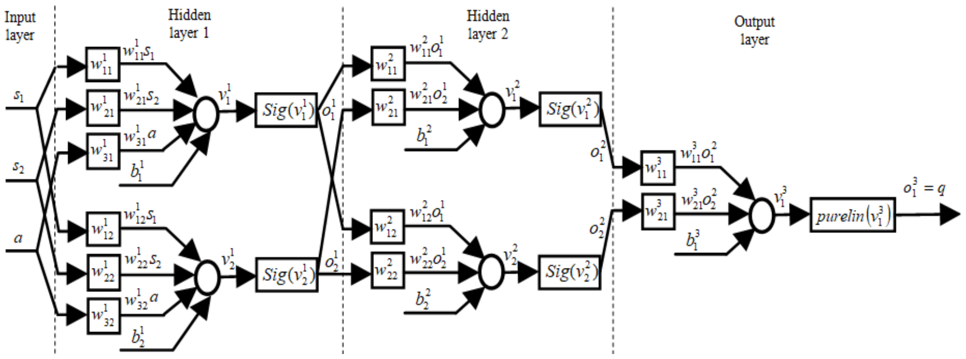
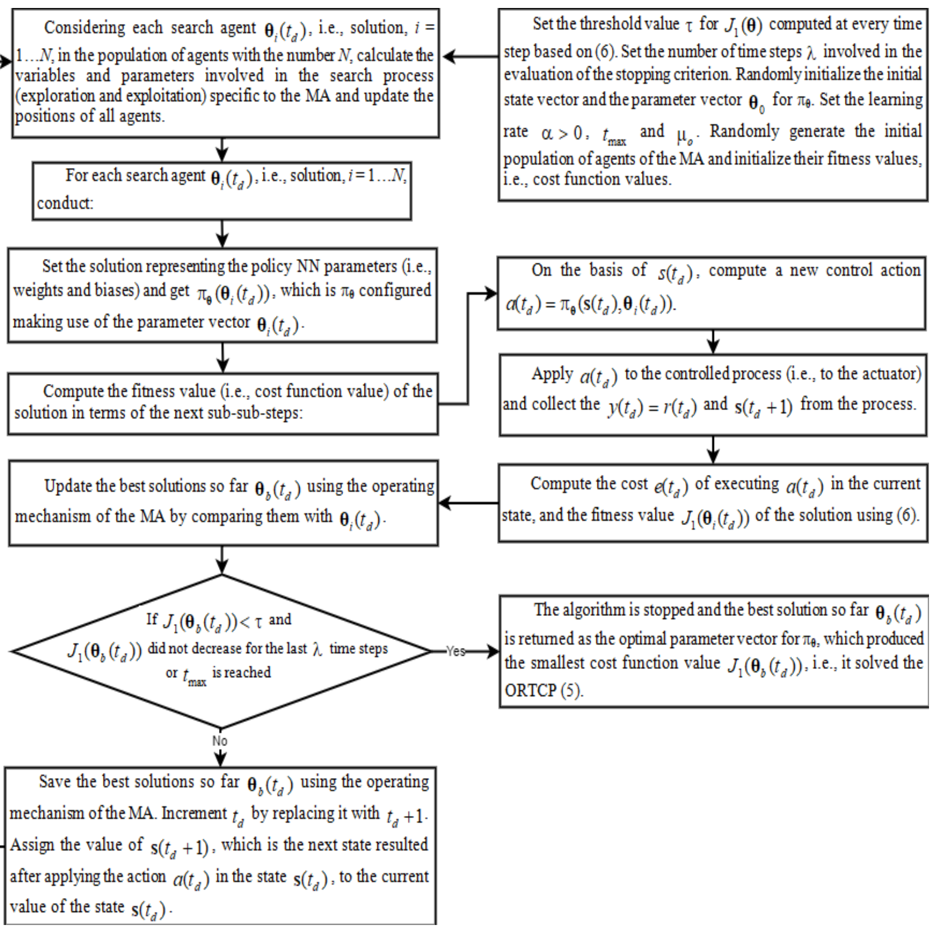


Fig. 5 Q-function NN architecture with two hidden layers [22]



Set the threshold value τ for $J_1(\boldsymbol{\theta})$ computed at every time step based on (6). Set the number of time steps λ involved in the evaluation of the stopping criterion. Randomly initialize the initial state vector and the parameter vector $\boldsymbol{\theta}_0$ for $\pi_{\boldsymbol{\theta}}$. Set the learning rate $\alpha > 0$, $t_{\max}$ and μ_o. Randomly generate the initial population of agents of the MA and initialize their fitness values, i.e., cost function values.

Considering each search agent $\boldsymbol{\theta}_i(t_d)$, i.e., solution, $i = 1\ldots N$, in the population of agents with the number N, calculate the variables and parameters involved in the search process (exploration and exploitation) specific to the MA and update the positions of all agents.

For each search agent $\boldsymbol{\theta}_i(t_d)$, i.e., solution, $i = 1\ldots N$, conduct:

Set the solution representing the policy NN parameters (i.e., weights and biases) and get $\pi_{\boldsymbol{\theta}}(\boldsymbol{\theta}_i(t_d))$, which is $\pi_{\boldsymbol{\theta}}$ configured making use of the parameter vector $\boldsymbol{\theta}_i(t_d)$.

Compute the fitness value (i.e., cost function value) of the solution in terms of the next sub-sub-steps:

On the basis of $s(t_d)$, compute a new control action $a(t_d) = \pi_{\boldsymbol{\theta}}(\mathbf{s}(t_d), \boldsymbol{\theta}_i(t_d))$.

Apply $a(t_d)$ to the controlled process (i.e., to the actuator) and collect the $y(t_d) = r(t_d)$ and $\mathbf{s}(t_d + 1)$ from the process.

Compute the cost $e(t_d)$ of executing $a(t_d)$ in the current state, and the fitness value $J_1(\boldsymbol{\theta}_i(t_d))$ of the solution using (6).

Update the best solutions so far $\boldsymbol{\theta}_b(t_d)$ using the operating mechanism of the MA by comparing them with $\boldsymbol{\theta}_i(t_d)$.

If $J_1(\boldsymbol{\theta}_b(t_d)) < \tau$ and $J_1(\boldsymbol{\theta}_b(t_d))$ did not decrease for the last λ time steps or $t_{\max}$ is reached

Yes

The algorithm is stopped and the best solution so far $\boldsymbol{\theta}_b(t_d)$ is returned as the optimal parameter vector for $\pi_{\boldsymbol{\theta}}$, which produced the smallest cost function value $J_1(\boldsymbol{\theta}_b(t_d))$, i.e., it solved the ORTCP (5).

No

Save the best solutions so far $\boldsymbol{\theta}_b(t_d)$ using the operating mechanism of the MA. Increment t_d by replacing it with $t_d + 1$. Assign the value of $\mathbf{s}(t_d + 1)$, which is the next state resulted after applying the action $a(t_d)$ in the state $\mathbf{s}(t_d)$, to the current value of the state $\mathbf{s}(t_d)$.

Fig. 6 Flow diagram of the combination of Policy Iteration and GWO

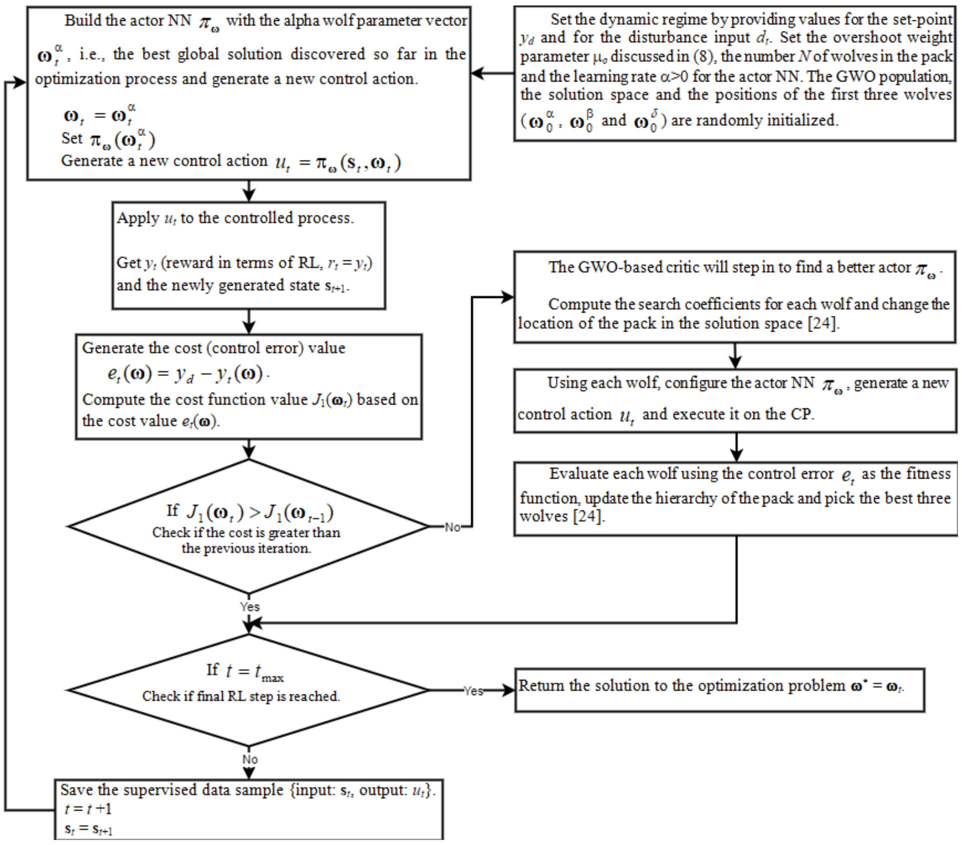

Build the actor NN $\pi_{\boldsymbol{\omega}}$ with the alpha wolf parameter vector $\boldsymbol{\omega}_t^{\alpha}$, i.e., the best global solution discovered so far in the optimization process and generate a new control action.
$\boldsymbol{\omega}_t = \boldsymbol{\omega}_t^{\alpha}$
Set $\pi_{\boldsymbol{\omega}}(\boldsymbol{\omega}_t^{\alpha})$
Generate a new control action $u_t = \pi_{\boldsymbol{\omega}}(\mathbf{s}_t, \boldsymbol{\omega}_t)$
Set the dynamic regime by providing values for the set-point y_d and for the disturbance input d_t. Set the overshoot weight parameter μ_o discussed in (8), the number N of wolves in the pack and the learning rate α>0 for the actor NN. The GWO population, the solution space and the positions of the first three wolves ($\boldsymbol{\omega}_0^{\alpha}$, $\boldsymbol{\omega}_0^{\beta}$ and $\boldsymbol{\omega}_0^{\delta}$) are randomly initialized.
Apply u_t to the controlled process.
Get y_t (reward in terms of RL, $r_t = y_t$) and the newly generated state $\mathbf{s}_{t+1}$.
Generate the cost (control error) value $e_t(\boldsymbol{\omega}) = y_d - y_t(\boldsymbol{\omega})$.
Compute the cost function value $J_1(\boldsymbol{\omega}_t)$ based on the cost value $e_t(\boldsymbol{\omega})$.
If $J_1(\boldsymbol{\omega}_t) > J_1(\boldsymbol{\omega}_{t-1})$
Check if the cost is greater than the previous iteration.
No
Yes
The GWO-based critic will step in to find a better actor $\pi_{\boldsymbol{\omega}}$.
Compute the search coefficients for each wolf and change the location of the pack in the solution space [24].
Using each wolf, configure the actor NN $\pi_{\boldsymbol{\omega}}$, generate a new control action u_t and execute it on the CP.
Evaluate each wolf using the control error e_t as the fitness function, update the hierarchy of the pack and pick the best three wolves [24].
If $t = t_{\max}$
Check if final RL step is reached.
Yes
Return the solution to the optimization problem $\boldsymbol{\omega}^* = \boldsymbol{\omega}_t$.
No
Save the supervised data sample {input: $\mathbf{s}_t$, output: u_t}.
$t = t+1$
$\mathbf{s}_t = \mathbf{s}_{t+1}$

Fig. 7 Flow diagram of the combination of Actor-Critic and GWO

All these algorithms are built as NN-based state feedback controllers, and the experiments are conducted using the nonlinear servo system laboratory equipment described in [24] and illustrated in Fig. 8. The process model is introduced in [25] as

$$\mu(t) = \begin{cases} -1, & \text{if } u(t) \le -u_b, \\ (u(t)+u_c)/(u_b-u_c), & \text{if } -u_b < u(t) < -u_c, \\ 0, & \text{if } -u_c \le |u(t)| \le u_a, \\ (u(t)-u_a)/(u_b-u_a), & \text{if } u_a < u(t) < u_b, \\ 1, & \text{if } u(t) \ge u_b, \end{cases}$$
$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1/T_\Sigma \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ k_P/T_\Sigma \end{bmatrix} \mu(t), \tag{9}$$
$$y(t) = [1 \quad 0]\,\mathbf{x}(t),$$

where the significance of variables and parameters is [25]: $t \geq 0$ is the continuous time, $k_p > 0$ is the servo system gain, $T_\Sigma > 0$ is a small time constant, the pulse with modulated control signal $u(t)$ applied to the Direct Current (DC) motor has a duty cycle $-1 \leq u(t) \leq 1$, the state vector of the system is $\mathbf{s}(t) = [s_1(t)\ s_2(t)]^T = \mathbf{x}(t) = [x_1(t)\ x_2(t)]^T$, and u_a, u_b and u_c, $0 < u_a < u_b$, $0 < u_c < u_b$ are the parameters of the dead zone static nonlinearity variable $\mu(t)$ modeled in the first relation in Eq. (9). The parameter values related to Eq. (9), obtained by least squares identification, are $u_a = 0.15$, $u_b = 1$, $u_c = 0.15$, $k_p = 140$ and $T_\Sigma = 0.92$s [25].

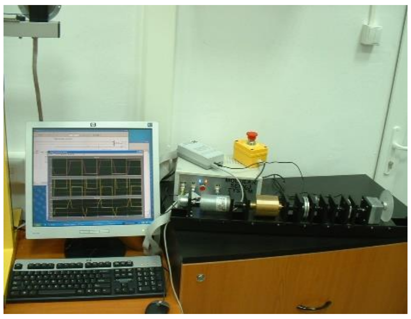


Fig. 8 Photo of nonlinear servo system experimental setup in authors' laboratories [22]

Each algorithm is assessed on how well it solves an ORTCP – defined in Eq. (1) or Eq. (5) – in servo system position control using the following performance indices: the minimum value of the cost function $J_{1\min}$, plus the empirical performance indices the settling time $t_s(s)$, the 0-to-100% rise time $t_1(s)$, the peak time $t_m(s)$ and the percent overshoot $\sigma_1(\%)$.

The dynamic regime related to the optimization problems defined in Eq. (1) and Eq. (5) and employed as ORTCPs in the AC formulation is characterized by: $y_d = 40$ rad step type modification of the set-point, randomly generated initial state variables, zero disturbance input, and time horizon of 30 s. These values are kept permanently during the controller design and tuning in order to guarantee the correct evaluation and comparison of the cost function values.

Since the MAs are essentially characterized by random features, the experiments conducted with the algorithms that include MAs were repeated nine times (i.e. episodes) using the same initial conditions and averaging the results. The nine episodes were organized, as proceeded in [22-24], in three stages, and each stage contains three episodes. The experiments involving the two GD-based algorithms were run only once in each of the three stages they do not operate with randomly generated values. In order to ensure a fair comparison of all algorithms, the statistical results are obtained and formulated after processing the results of running 30 real-time experiments with 3000 iterations/experiment for all NN-based controllers. As specified in [22], in the same context of random features, the initialization step in every experiment conducted with the algorithms based on (i) was executed three times and the resulted values of the parameter vectors $\boldsymbol{\theta}$, $\mathbf{w}$ and $\boldsymbol{\rho}$ were averaged before using them to train the NN-based state feedback controllers.

As considered in [22-24], the parameters involved in the optimization problems were set to $t_{\max} = 1500$ and $\mu_o = 1.5$. The sampling period for control was set to 0.01 s. The parameter vectors of the NN-based state feedback controllers after training contain 17 elements (weights and biases). The augmented state vector contains randomly generated values in the state space $S = [0,60] \times [0,100] \times [0,30]$ at each stage, with the first two intervals corresponding to the state variables in Eq. (14) and the third interval to the integrator state.

The parameter specific to the stochastic GD-based algorithms was set to $\alpha = 0.01$. The number of agents used in the MAs was set to $N = 10$, the initial value of the gravitational constant specific to GSA was set to $g_{grav}^{0} = 1.9$, and the other parameter settings of MAs are described in [6].

The expressions and the obtained values of the parameter vectors of all algorithms are specified in [22-24]. In this context of ensuring full transparency, the data processed in the statistical analysis conducted in order to compare the performance of all algorithms is freely available in the *Data_FUME.m* Matlab file [26]. The readers are invited to examine several samples of CS responses in [22-24].

The results obtained after conducting the variance (ANOVA) test of the minimum cost function value $J_{1\min}$ evaluated after running these ten algorithms are presented in Fig. 9. Fig. 9 shows that the best performance is achieved by the GSA-DQL algorithm, followed by the GSO-AC algorithm and the PSO-DQL algorithm. Table 1 summarizes the results of the non-parametric statistical analysis.

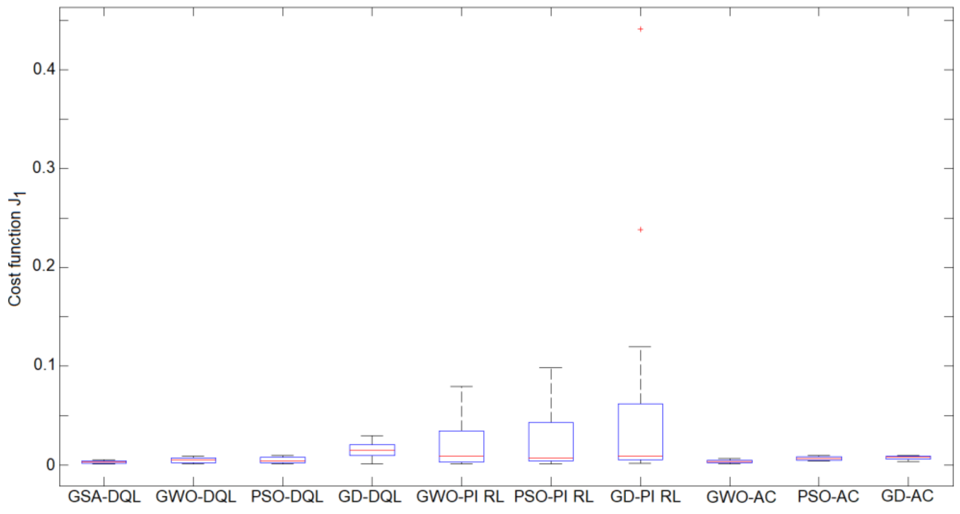


Fig. 9 ANOVA test of minimum cost function value J_1 for all algorithms

Table 2 gives a comparison of the performance indices as far as the CS behavior is concerned for all seven algorithms under discussion. The bold values indicate the best (namely smallest) performance indices among all algorithms.

The best algorithm as far as the settling time is concerned is GWO-AC, followed by PSO-AC and GWO-PI RL. The best algorithm as far as the 0-to-100% rise time is concerned is PSO-PI RL, followed by GD-PI RL and GD-DQL. The best algorithm as far as the peak time is concerned is GD-PI RL, followed by PSO-PI RL and GWO-PI RL. The best algorithm as far as the overshoot is concerned is GWO-AC, followed by GWO-PI RL and GSA-DQL. These conclusions drawn from Table 2 indicate that there are certain differences of the empirical performance indices, which make the control system designers choose the right algorithm in order to achieve the performance specifications imposed to the control system, and to ensure a convenient tradeoff to them. These conclusions will be different for other dynamic regimes and might also need to be subjected to a tradeoff.

Table 1 Ranks, statistics and related p-values of ten algorithms

Algorithm	Friedman test	Friedman aligned test	Quade test
GSA-DQL	2.2667	62.3667	2.1161
GWO-DQL	3.2333	73.8333	2.8731
PSO-DQL	3.1	74.9667	2.8796
GD-DQL	5.2667	113.3667	4.7376
GWO-PI RL	4.4333	128.4333	4.7914
PSO-PI RL	4.6	133.6667	4.9634
GD-PI RL	5.1	151.8667	5.6387
GWO-AC	1.1333	17.3667	1.1484
PSO-AC	2.3333	56.1667	2.3161
GD-AC	2.5333	62.9667	2.5355
Statistic	89.97	95.58	13.90
p-value	$1.67\cdot10^{-15}$	$1.11\cdot10^{-16}$	$2.07\cdot10^{-20}$

Table 2 Comparison of cntrol system performance indices with controllers tuned by ten algorithms

Algorithm	t_s (s)	t_1 (s)	t_m (s)	σ_1 (%)
GSA-DQL	24.3027	10.89	15.8133	1.1847
GWO-DQL	24.7913	10.466	15.6457	1.51
PSO-DQL	25.456	9.418	14.6713	1.8077
GD-DQL	25.7247	8.7743	13.2433	1.8287
GWO-PI RL	23.4215	9.4544	11.9362	0.7658
PSO-PI RL	25.259	**6.7314**	9.3833	4.8424
GD-PI RL	24.0382	7.6027	**6.7273**	23.2914
GWO-AC	**21.7897**	12.041	16.2593	**0.7051**
PSO-AC	22.955	13.4543	16.9083	1.7128
GD-AC	24.5883	13.267	16.6147	1.5147

The integrator is introduced to ensure zero steady-state error; experimental data to confirm this conclusion is also included in [22-24]. A sample of control system response is illustrated in Fig. 10, which confirms both the zero steady-state control error and the effects of the dead zone static nonlinearity in the servo system model given in Eq. (9).

The analysis part of the experimental results does not include the training convergence curves, in order to keep a reasonable length of the paper. However, convergence curves can be relatively easily derived using the information extracted from the system responses as those given in Fig. 10. Nevertheless, the conclusions of this comparison can be different if other nonlinear processes are subjected to the NN-based ORTCP approaches and algorithms discussed in this paper as, for instance, various representative applications that include decision-making processes [27], man-computer symbiosis [28], 3D printing objects [29], medical systems [30]–[33], drilling processes [34], fuzzy control [35]–[37], evolving controllers [38], fuzzy cognitive maps [39], traffic systems [40], quantum computing [41], and telesurgical applications [42].

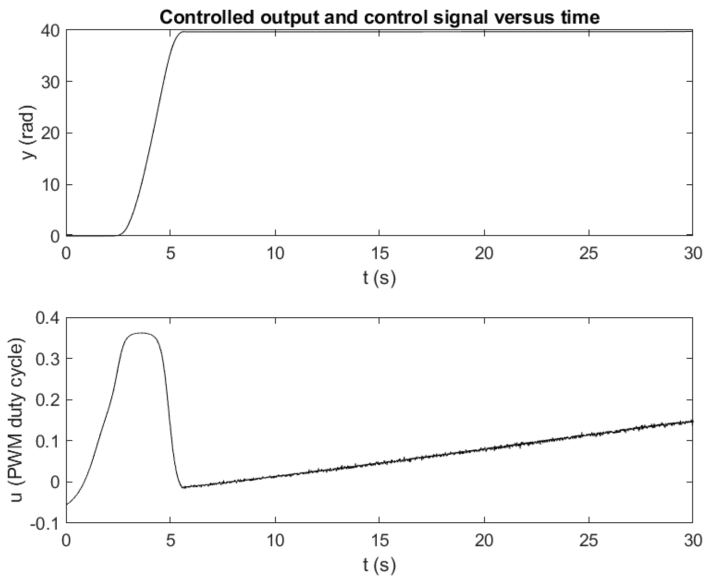


Fig. 10 Real-time experimental system responses *y* and *u* when using the PSO-PI RL algorithm [23]

5. Conclusions

This paper carried out the performance analysis of three control system structures and approaches, which combine Reinforcement Learning (RL) and four representative Metaheuristic Algorithms (MAs) resulting in seven RL algorithms that include the classical stochastic Gradient Descent algorithms. The algorithms, suggested in the recent authors' papers [22-24], were implemented as Neural Network (NN)-based state feedback controllers and applied to the position control of a nonlinear servo system.

The main benefit of these approaches is the transparency of presentation in an easily understandable way of the NN training and the controller implementation. The disturbance rejection is ensured by the presence of the integrator in the control system structures. The main limitation of the approaches treated in this paper is the relatively large number of parameters of the NN-based state feedback controllers accounting for the application considered in the previous section. Nevertheless, the stability analysis is a sensitive subject, in order to use stability conditions as constraints in the optimization problems, but it is discussed in [1] in the context of data-driven control.

Future research will be focused on applying the algorithms and controllers to other complicated and challenging processes. The simplification of the NN architectures will be tackled as well, aiming the cost-effective training, design, tuning and implementation.

Acknowledgement: *This work was supported by a grant of the Romanian Ministry of Education and Research, CNCS - UEFISCDI, project number PN-III-P4-ID-PCE-2020-0269, within PNCDI III, and by the NSERC of Canada. The support of Dr. Raul-Cristian Roman in conducting the real-time experiments on the servo system equipment is duly acknowledged.*

REFERENCES

1. Precup, R.-E., Roman, R.-C., Safaei, A., 2021, *Data-Driven Model-Free Controllers, 1st Edition*. CRC Press, Taylor & Francis, Boca Raton, FL.
2. Sutton, R.S., Barto, A.G., 2017, *Reinforcement Learning: An Introduction, 2nd Edition*. MIT Press, Cambridge, MA, London.
3. Sutton, R.S., Barto, A.G., Williams, R.J., 1992, *Reinforcement learning is direct adaptive optimal control*, IEEE Control Systems Magazine, 12(2), pp. 19-22.
4. Busoniu, L., de Bruin, T., Tolić, D., Kober, J., Palunko, I., 2018, *Reinforcement learning for control: performance, stability, and deep approximators*, Annual Reviews in Control, 46(1), pp. 8-28.
5. Ganaie, M.A., Hu, M.-H., Malik, A.K., Tanveer. M., Suganthan, P.N., 2022, *Ensemble deep learning: A review*, Engineering Applications of Artificial Intelligence, 115, paper 105151.
6. Precup, R.-E., David, R.-C., 2019, *Nature-inspired Optimization Algorithms for Fuzzy Controlled Servo Systems*. Butterworth-Heinemann, Elsevier, Oxford.
7. Precup, R.-E., Angelov, P., Costa, B.S.J., Sayed-Mouchaweh, M., 2015, *An overview on fault diagnosis and nature-inspired optimal control of industrial process applications*, Computers in Industry, 74, pp. 75-94.
8. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R., 2019, *Designing neural networks through neuroevolution*, Nature Machine Intelligence, 1, pp. 24-35.
9. Sehgal, A., La, H.M., Louis, S.J., Nguyen, H., 2019, *Deep reinforcement learning using genetic algorithm for parameter optimization*, Proc. 2019 3rd IEEE International Conference on Robotic Computing, Naples, Italy, pp. 596-601.
10. Ajani, O.S., Mallipeddi, R., 2022, *Adaptive evolution strategy with ensemble of mutations for Reinforcement Learning*, Knowledge-Based Systems, 245, paper 108624.
11. Goulart, D.A., Pereira, R.D., 2020, *Autonomous pH control by reinforcement learning for electroplating industry wastewater*, Computers & Chemical Engineering, 140, paper 106909.
12. Lin, H.-W., Wu, Q.-Y., Liu, D.-R., Zhao, B., Yang, Q.-M., 2019, *Fault tolerant control for nonlinear systems based on adaptive dynamic programming with particle swarm optimization*, Proc 10th International Conference on Intelligent Control and Information Processing, Marrakesh, Morocco, pp. 322-326.
13. Liu, X., Zhao, B., Liu, D., 2020, *Fault tolerant tracking control for nonlinear systems with actuator failures through particle swarm optimization-based adaptive dynamic programming*, Applied Soft Computing, 97(A), paper 106766.
14. Hein, D., Hentschel, A., Runkler, T., Udluft, S., 2017, *Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies*, Engineering Applications of Artificial Intelligence, 65, pp. 87-98.
15. Piperagkas, G.S., Georgoulas, G., Parsopoulos, K.E., Stylios, C.D., Likas, A.C., 2012, *Integrating particle swarm optimization with reinforcement learning in noisy problems*, Proc. 14th Annual Conference on Genetic and Evolutionary Computation, Philadelphia, PA, USA, pp. 65-72.
16. Iima, H., Kuroe, Y., 2008, *Swarm reinforcement learning algorithms based on particle swarm optimization*, Proc. 2008 IEEE International Conference on Systems, Man and Cybernetics, Singapore, Singapore, pp. 1110-1115.
17. Hein, D., Hentschel, A., Runkler, T., Udluft, S., 2016, *Reinforcement learning with Particle Swarm Optimization Policy (PSO-P) in continuous state and action spaces*, International Journal of Swarm Intelligence Research, 7(3), pp. 23-42.
18. Meerza, S.I., Islam, M., Uzzal, M.M., 2019, *Q-learning based particle swarm optimization algorithm for optimal path planning of swarm of mobile robots*, Proc. 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology, Dhaka, Bangladesh, pp. 1-5.
19. Gao, Y.-Z., Ye, J.-W., Chen, Y.-M., Liang, F.-L., 2009, *Q-learning based on particle swarm optimization for positioning system of underwater vehicles*, Proc. 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, Shanghai, China, vol. 2, pp. 68-71.
20. Zhang, P., Li, H., Ha, Q.P., Yin, Z.-Y., Chen, R.-P., 2020, *Reinforcement learning based optimizer for improvement of predicting tunneling-induced ground responses*, Advanced Engineering Informatics, 45, paper 101097.

21. Mirjalili, S., 2015, *How effective is the grey wolf optimizer in training multi-layer perceptrons*, Applied Intelligence, 43(1), pp. 150-161.
22. Zamfirache, I. A., Precup, R.-E., Roman, R.-C., Petriu, E.M., 2022, *Reinforcement learning-based control using Q-learning and gravitational search algorithm with experimental validation on a nonlinear servo system*, Information Sciences, 583, pp. 99-120.
23. Zamfirache, I. A., Precup, R.-E., Roman, R.-C., Petriu, E.M., 2022, *Policy iteration reinforcement learning-based control using a grey wolf optimizer algorithm*, Information Sciences, 585, pp. 162-175.
24. Zamfirache, I. A., Precup, R.-E., Roman, R.-C., Petriu, E.M., 2023, *Neural network-based control using actor-critic reinforcement learning and grey wolf optimizer with experimental servo system validation*, Expert Systems with Applications, 225, paper 120112.
25. Precup, R.-E., David, R.-C., Petriu, E.M., 2017, *Grey wolf optimizer algorithm-based tuning of fuzzy control systems with reduced parametric sensitivity*, IEEE Transactions on Industrial Electronics, 64(1), pp. 527-534.
26. Zamfirache, I. A., Precup, R.-E., Petriu, E.M., Oct. 2022, Data obtained by 30 independent runs of all algorithms. [Online]. Available: http://www.aut.upt.ro/~rprecup/Data_FUME.m.
27. Božanić, D., Tešić, D., Marinković, D., Milić, A., 2021, *Modeling of neuro-fuzzy system as a support in decision-making processes*, Reports in Mechanical Engineering, 2(1), pp. 222-234.
28. Filip, F.G., 2021, *Automation and computers and their contribution to human well-being and resilience*, Studies in Informatics and Control, 30(4), pp. 5-18.
29. Milićević, I., Popović, M., Dučić, N., Vujičić, V., Stepanić, P., Marinković, D., Ćojbašić, Ž., 2022, *Improving the mechanical characteristics of the 3D printing objects using hybrid machine learning approach*, Facta Universitatis, Series: Mechanical Engineering, DOI: 10.22190/FUME220429036M.
30. Bejinariu, S.I., Costin, H., Rotaru, F., Niţă, C., Luca, R., Lazăr, C., 2014, *Parallel processing and bio-inspired computing for biomedical image registration*, Computer Science Journal of Moldova, 22(2), pp. 253-277.
31. Rigatos, G., Siano, P., Selisteanu, D., Precup, R.-E., 2017, *Nonlinear optimal control of oxygen and carbon dioxide levels in blood*, Intelligent Industrial Systems, 3(2), pp. 61-75.
32. Gerger, M., Gumuscu, A., 2022, *Diagnosis of Parkinson's disease using spiral test based on pattern recognition*, Romanian Journal of Information Science and Technology, 25(1), pp. 100-113.
33. Ogutcu, S., Inal, M., Celikhasi, C., Yildiz, U., Dogan N.O., Pekdemir, M., 2022, *Early detection of mortality in COVID-19 patients through laboratory findings with factor analysis and artificial neural networks*, Romanian Journal of Information Science and Technology, 25(3-4), pp. 290-302.
34. Haber-Haber, R., Haber, R., Schmittdiel, M., del Toro, R.M., 2007, *A classic solution for the control of a high-performance drilling process*, International Journal of Machine Tools and Manufacture, 47(15), pp. 2290-2297.
35. Precup, R.-E., Preitl, S., Balas, M., Balas, V., 2004, *Fuzzy controllers for tire slip control in anti-lock braking systems*, Proc. 2004 IEEE International Conference on Fuzzy Systems, Budapest, Hungary, vol. 3, pp. 1317-1322.
36. Tomescu, M.L., Preitl, S., Precup, R.-E., Tar, J.K., 2007, *Stability analysis method for fuzzy control systems dedicated controlling nonlinear processes*, Acta Polytechnica Hungarica, 4(3), pp. 127-141.
37. Precup, R.-E., Preitl, S., Petriu, E.M., Bojan-Dragos, C.-A., Szedlak-Stinean, A.-I., Roman, R.-C., Hedrea E.-L., 2020, *Model-based fuzzy control results for networked control systems*, Reports in Mechanical Engineering, 1(1), pp. 10-25.
38. Škrjanc, I., Blažič, S., Angelov, P., 2014, *Robust evolving cloud-based PID control adjusted by gradient learning method*, Proc. 2014 IEEE Conference on Evolving and Adaptive Intelligent Systems, Linz, Austria, pp. 1-6.
39. Vaščák, J., Hvizdoš, J., Puheim, M., 2016, *Agent-based cloud computing systems for traffic management*, Proc. 2016 International Conference on Intelligent Networking and Collaborative Systems, Ostrava, Czech Republic, pp. 73-79.
40. Zhang, L.-Y., Ma, J., Liu, X.-F., Zhang, M., Duan, X.-K., Wang, Z., 2022, *A novel support vector machine model of traffic state identification of urban expressway integrating parallel genetic and C-means clustering algorithm*, Tehnički vjesnik - Technical gazette, 29(3), pp. 731-741.
41. Osaba, E., Villar-Rodriguez, E., Oregi, I., Moreno-Fernandez-de-Leceta, A., 2021, *Hybrid quantum computing-tabu search algorithm for partitioning problems: preliminary study on the traveling salesman problem*, Proc. 2021 IEEE Congress on Evolutionary Computation, Kraków, Poland, pp. 351-358.
42. Precup, R.-E., Haidegger, T., Preitl, S., Benyó, B., Paul, A.S., Kovács, L., 2012, *Fuzzy control solution for telesurgical applications*, Applied and Computational Mathematics, 11(3), pp. 378-397.