Q-learning and Policy Iteration Reinforcement Learning Combined with Metaheuristic Algorithms in Servo System Control*

Iuliu Alexandru Zamfirache, *Student Member, IEEE*, Radu-Emil Precup, *Senior Member, IEEE*, Raul-Cristian Roman, *Member, IEEE*, Emil M. Petriu, *Life Fellow, IEEE*, and Miruna-Maria Damian, *Student Member, IEEE*

***Abstract*—This paper carries out the performance analysis of two control system structures and approaches, which combine Reinforcement Learning (RL) and Metaheuristic Algorithms (MAs) as representative optimization algorithms. In the first approach, the MA 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 MA is employed to train the policy NN in Policy Iteration RL-based control. The Gravitational Search Algorithm (GSA), the Grey Wolf Optimizer (GWO) algorithm and the Particle Swarm Optimization (PSO) algorithm are used as MAs in the first approach, whereas the GWO algorithm and the PSO algorithm in the second one. The performance specifications, which ensure a tradeoff to small settling time and overshoot values, are expressed as an optimal reference tracking control problem, where the cost function is the absolute control error plus the weighted absolute control error increment. State feedback control system structures are used, with integrator included in the second approach in order to ensure zero steady-state control error. 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.**

I. 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 the recent book [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 (which measures 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. 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.

*Research 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.

I. A. Zamfirache, R.-E. Precup, R.-C. Roman and M.-M. Damian are with the Department of Automation and Applied Informatics, Politehnica University of Timisoara, Bd. V. Parvan 2, 300223 Timisoara, Romania (phone: +40 25640-26, -29, -30, -40, -52, fax: +40 256403214; e-mail: iuliu.zamfirache@student.upt.ro, radu.precup@upt.ro, raul-cristian.roman@upt.ro, miruna-maria.damian@student.upt.ro). R.-E. Precup is also with the Romanian Academy – Timisoara Branch, Center for Fundamental and Advanced Technical Research, Bd. Mihai Viteazu 24, 300223 Timisoara, Romania.

E. M. Petriu is with the School of Electrical Engineering and Computer Science, University of Ottawa, 800 King Edward, Ottawa, ON, K1N 6N5 Canada (e-mail: petriu@uottawa.ca).

Manuscript 128 submitted to 2023 European Control Conference (ECC).
Received November 19, 2022.

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.

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 and impossibility to avoid local optima. 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]. Both [22] and [23] report 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] and [23], this paper carries out the performance analysis of two fresh NN- and RL-based control system structures and approaches, (i) and (ii), 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. 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. Five combined algorithms are discussed, namely GSA, GWO and PSO in (i), and GWO and PSO in (ii), and these algorithms are 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 III. The performance comparison expressed in terms of non-parametric statistical tests conducted on experimental results is given in Section IV, and the concluding remarks are pointed out in Section V.

II. 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 two approaches (i) and (ii) mentioned in the previous section will be briefly treated as follows in Sections A and B, respectively.

A. *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

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

$t_d = 1 \dots t_{\max}$ 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 in (1), it consists of the optimal sequence of elements, i.e. actions $a^*(t_d)$, $t_d = 1 \dots t_{\max}$:

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

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

The definition of the cost function in (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]

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

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 (1) to (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 $Q_\mathbf{w}$ is referred to as the Q-function NN $Q_\mathbf{w}$.

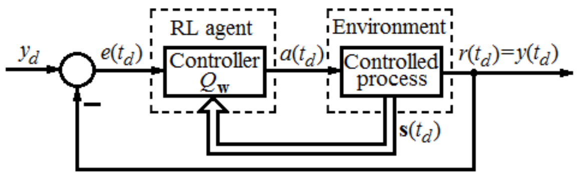

Fig. 1. Informational structure of DQL-based CS [22].

B. Optimal Reference Tracking Control Problem in the Second Approach

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

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

where $\boldsymbol{\theta}$ is the parameter vector, $\boldsymbol{\theta}^*$ is the solution to the (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

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

which is similar to that defined in (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.

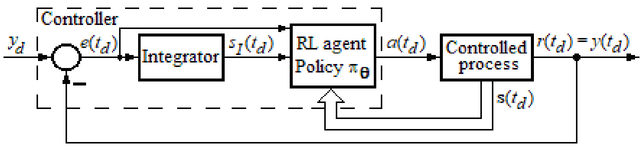

Fig. 2. Informational structure of PI RL-based CS (adapted from [23]).

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.

III. Combined Reinforcement Learning-Metaheuristic Algorithms

Using the presentation in [22] focusing on GSA, *the steps of the DQL-based control approach combined with MAs* are formulated as follows using the description given in [22] from a control perspective:

Step D1. Set the Q-function and the target NN architecture exemplified in Fig. 3 for two hidden layers. Set the dynamic regime considered to solve the optimization problem defined in (1). Use one of the MAs to initialize the parameter vector $\mathbf{w}_0$ of the Q-function NN $Q_\mathbf{w}$ and the parameter vector $\boldsymbol{\rho}_0$ of the target NN $\hat{Q}_\rho$. Randomly initialize the values of the elements of the initial state vector $\mathbf{s}_0$. Establish the learning rate $\alpha > 0$ involved in the stochastic GD algorithm, $t_{\max}$ and μ_o. Initialize the experience replay buffer with the notation Φ, the training dataset with the notation K, and the discrete action space with the notation A. The number of steps z is established, after which a set of samples will be arbitrary picked out from the experience replay buffer and the Q-function NN will be trained. Establish the number of steps g after which the Q-function NN parameter vector $\mathbf{w}$ will be synchronized with the target NN parameter vector $\boldsymbol{\rho}$. The iteration index is set to $t_d = 0$.

Step D2. 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 [22]

$$a(t_d) = \pi(\mathbf{s}(t_d)) = \arg\max_{a \in A} Q_\mathbf{w}(\mathbf{s}(t_d), a, \mathbf{w}(t_d)). \quad (7)$$

Step D3. 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.

Step D4. Update the sequence of control actions (vector) $\mathbf{a}$ defined in (3). Update the element placed in the position t_d with $a(t_d)$ generated in *step D2*.

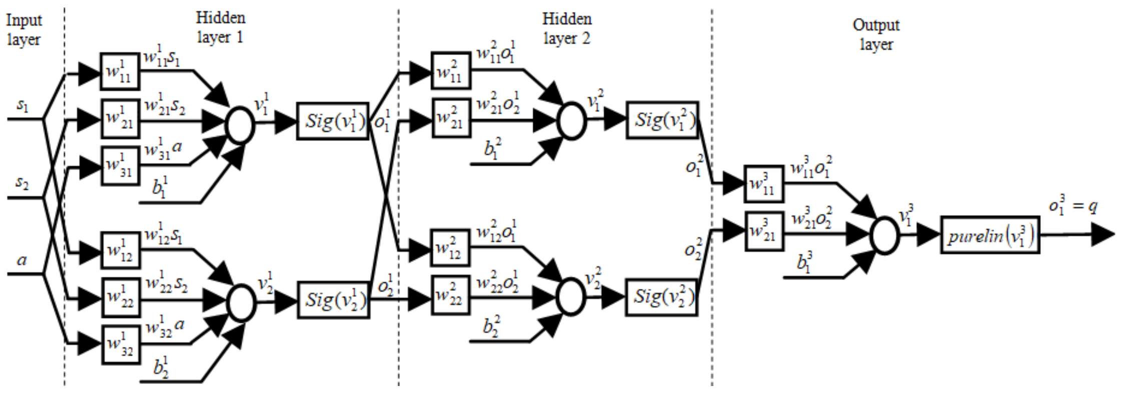

Fig. 3. Q-function NN architecture with two hidden layers [22].

Step D5. 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).

Step D6. Use $\mathbf{s}(t_d)$ and $a(t_d)$ obtained in *step D2*, and the current Q-function NN $Q_\mathbf{w}$ to determine the value $q(t_d)$ of the state-action pair [22]

$$q(t_d) = Q_\mathbf{w}(\mathbf{s}(t_d), a(t_d), \mathbf{w}(t_d)). \tag{8}$$

Step D7. Apply $a(t_d)$ to obtain $\mathbf{s}(t_d+1)$ and the target NN $\hat{Q}_\boldsymbol{\rho}$, estimate the value of $\mathbf{s}(t_d+1)$ and use $\hat{Q}_\boldsymbol{\rho}$ to obtain the Temporal Difference (TD) target value $\delta(t_d)$ [22]

$$\delta(t_d) = r(t_d) + \gamma\,\hat{Q}_\boldsymbol{\rho}(\mathbf{s}(t_d+1), a(t_d), \boldsymbol{\rho}(t_d)), \tag{9}$$

where $\hat{Q}_\boldsymbol{\rho}(\mathbf{s}(t_d+1), a(t_d), \boldsymbol{\rho}(t_d))$ is the output of $\hat{Q}_\boldsymbol{\rho}$.

Step D8. Build the experience tuple $\varphi(t_d) = \{\mathbf{s}(t_d), a(t_d), e(t_d), \mathbf{s}(t_d+1), q(t_d), \delta(t_d)\}$.

Step D9. Add the new experience $\varphi(t_d)$ to Φ.

Step D10. Add the new experience $\varphi(t_d)$ to K.

Step D11. Verify if the time to train the Q-learning NN occurs, i.e., *z* steps passed in the DRL process. If $t \bmod z = 0$, the batch will be next built using experiences chosen arbitrarily in Φ, and the Q-function NN will be trained accordingly. Else, continue with *step D12*.

Sub-step D11.1. Initialize the iteration index of the stochastic GD training loop to $t_{SGD} = 0$.

Sub-step D11.2. The values involved in the training process are $q(t_d)$ defined in (8) and $\delta(t_d)$ defined in (9). Extract the elements $q_{t_{SGD}}(t_d)$ and $\delta_{t_{SGD}}(t_d)$ in the experience tuple $\varphi_{t_{SGD}}(t_d)$.

Sub-step D11.3. Compute the TD error $e_{TD}(t_d)$

$$e_{TD}(t_d) = \delta(t_d) - q(t_d) \tag{10}$$

and the cost function $J_2(\mathbf{w}_{t_{SGD}}(t_d))$ [22]

$$J_2(\mathbf{w}(t_d)) = \sum_{t_Q = t_d}^{t_d+H-1} (e_{TD}(t_Q))^2 \tag{11}$$

on the time horizon of length H of discrete time steps t_Q involved in the Q-function NN training.

Sub-step D11.4. Update $\mathbf{w}_{t_{SGD}}(t_d)$ to minimize the cost function $J_2(\mathbf{w}_{t_{SGD}}(t_d))$ using the gradient of $J_2(\mathbf{w}_{t_{SGD}-1}(t_d))$ [22]

$$\mathbf{w}_{t_{SGD}}(t_d) = \mathbf{w}_{t_{SGD}-1}(t_d) + \alpha\,\nabla_\mathbf{w} J_2(\mathbf{w}_{t_{SGD}-1}(t_d)), \tag{12}$$

where $\nabla_\mathbf{w} \bullet$ indicates the gradient of the scalar function $\bullet$ calculated with respect to the vector variable $\mathbf{w}$.

Sub-step D11.5. Replace t_{SGD} with $t_{SGD}+1$. Continue with the next experience $\varphi_{t_{SGD}+1}(t_d)$ in the batch.

Sub-step D11.6. Check the stochastic GD training stopping criterion. If $t_{SGD} = B_H$, this indicates that all experiences in the batch were processed and the training process needs to stop, therefore go to *step D12*. Else, continue training by going to *sub-step D11.2*.

Step D12. Verify if it is the proper moment to synchronize the parameter vector $\mathbf{w}$ of the Q-function NN with the parameter vector $\boldsymbol{\rho}$ of the target NN. If $t \bmod g = 0$, then $\boldsymbol{\rho}(t_d) = \mathbf{w}(t_d)$; otherwise, continue with *step D13*.

Step D13. Determine if $t_{\max}$ is reached. If $t = t_{\max}$ after that 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 $J_1(\mathbf{w}(t_d))$. The algorithm calculates $\mathbf{a}^*$ incrementally calculated in *step D4* as solution to the optimization problem (1) and K, and iteratively updated in *step D10*. The dataset K will next be employed to train the NN-based controller used as a nonlinear controller in the experiments in the next section. Else, continue with *step D14*.

Using the presentation in [22] focusing on GWO, *the steps of the PI RL-based control approach combined with MAs* are formulated as follows using the description given in [23] from a control perspective:

Step P1. Set the policy NN $\pi_{\boldsymbol{\theta}}$ architecture. Set the dynamic regime considered to solve the optimization problem defined in (5). 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 in *step P4*. Randomly initialize the values of the elements of the initial state vector $\mathbf{s}_0$. Randomly initialize the parameter vector $\boldsymbol{\theta}_0$ for $\pi_{\boldsymbol{\theta}}$. Set the learning rate $\alpha > 0$ involved in the stochastic GD algorithm, $t_{\max}$ and μ_o. Randomly generate the initial population of agents of the MA and initialize their fitness values, i.e., cost function values. The n+1– dimensional state space S (n is the number, assumed to be known, of the state variables of the controlled process) and the one-dimensional action space A do not require initialization because they are continuous. Initialize the iteration index to $t_d = 0$.

Step P2. Considering each search agent $\boldsymbol{\theta}_i(t_d)$, i.e., solution, i = 1…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.

Step P3. For each search agent $\boldsymbol{\theta}_i(t_d)$, i.e., solution, i = 1…N, conduct:

Sub-step P3.1. 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 $\boldsymbol{\theta}_i(t_d)$.

Sub-step P3.2. Compute the fitness value (i.e., cost function value) of the solution in terms of the next sub-sub-steps:

P3.2.1. On the basis of $s(t_d)$, compute a new control action $a(t_d)$ [23]

$$a(t_d) = \pi_{\boldsymbol{\theta}}(\mathbf{s}(t_d), \boldsymbol{\theta}_i(t_d)). \tag{13}$$

P3.2.2. 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.

P3.2.3. 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).

Sub-step P3.3. Update the best solutions so far $\boldsymbol{\theta}_b(t_d)$ using the operating mechanism of the MA by comparing them with $\boldsymbol{\theta}_i(t_d)$.

Step P4. Verify if for the last λ time steps, the fitness value of the best solution obtained so far was not updated, i.e., $J_1(\boldsymbol{\theta}_b(t_d))$ did not decrease, and it was below τ, i.e., $J_1(\boldsymbol{\theta}_b(t_d)) < \tau$. 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, then 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). Else, continue with *step P5*.

Step P5. 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 $t_d + 1$. Go to the next state. 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 $\mathbf{s}(t_d)$ based on *step P3*, to the current value of the state $\mathbf{s}(t_d)$, and go to *step P2*.

Considering two state variables of the controlled process with the notations x_1 and x_2 for the state variables of the process and x_3 for the integrator state variable in the AC formulation, a policy NN architecture with two hidden layers is exemplified in Fig. 4. This architecture represents actually the structure of the NN-based controller used as a state feedback controller in the experiments conducted in the next section.

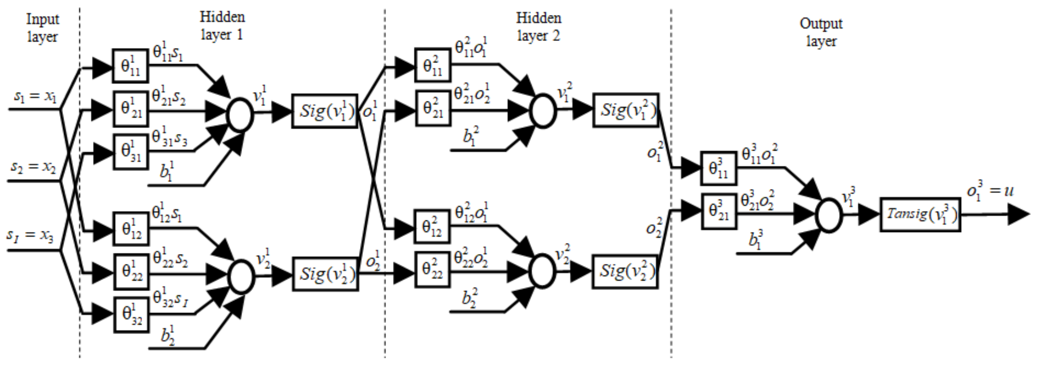

Fig. 4. Policy NN architecture with two hidden layers [23].

The stopping criteria are defined and explained in both algorithms presented in this section. 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.

IV. 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. Details on the design of the metaheuristic algorithms are given in [22] and [23].

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. 5. The process model is introduced in [24] as

$$\mu(t)=\begin{cases}-1, & \text{if } u(t)\le -u_b,\\ (u(t)+u_c)/(u_b-u_c), & \text{if } -u_b<u(t)<-u_c,\\ 0, & \text{if } -u_c\le |u(t)|\le u_a,\\ (u(t)-u_a)/(u_b-u_a), & \text{if } u_a<u(t)<u_b,\\ 1, & \text{if } u(t)\ge u_b,\end{cases}$$
$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix}=\begin{bmatrix}0 & 1\\ 0 & -1/T_\Sigma\end{bmatrix}\mathbf{x}(t)+\begin{bmatrix}0\\ k_P/T_\Sigma\end{bmatrix}\mu(t), \qquad (14)$$
$$y(t)=[1 \quad 0]\,\mathbf{x}(t),$$

where the significance of variables and parameters is [23]: $t\ge 0$ 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 $-1\le u(t)\le 1$, 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 equation in (14). The parameter values related to (14), obtained by least squares identification, are $u_a=0.15$, $u_b=1$, $u_c=0.15$, $k_P=140$ and $T_\Sigma=0.92\ \text{s}$ [24].

Each algorithm is assessed on how well it solves an ORTCP – defined in (1) or (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 σ_1 (%).

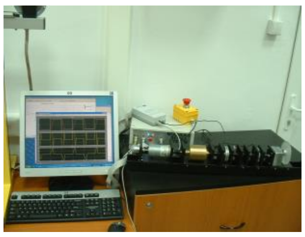
Fig. 5. Photo of nonlinear servo system experimental setup in authors' laboratories [22].

The dynamic regime related to the optimization problems defined in (1) and (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] and [23], 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] and [23], 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 (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] and [23]. 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_ECC.m* Matlab file [25]. The readers are invited to examine several samples of CS responses in [22] and [23].

The results obtained after conducting the variance (ANOVA) test of the minimum cost function $J_{1\min}$ evaluated after running these seven algorithms are presented in Fig. 6. Fig. 6 shows that the best performance is achieved by the GSA-DQL algorithm, followed by the GWO-DQL algorithm and the PSO-DQL algorithm.

The ranks, the statistics and the probability p-values corresponding to the Friedman, Friedman aligned and Quade tests for the minimum cost function $J_{1\min}$ obtained after running the seven algorithms are presented in Table I in terms of average ranks. Table I shows that the best performance is exhibited by the GSA-DQL algorithm, followed by the GWO-DQL and PSO-DQL algorithms.

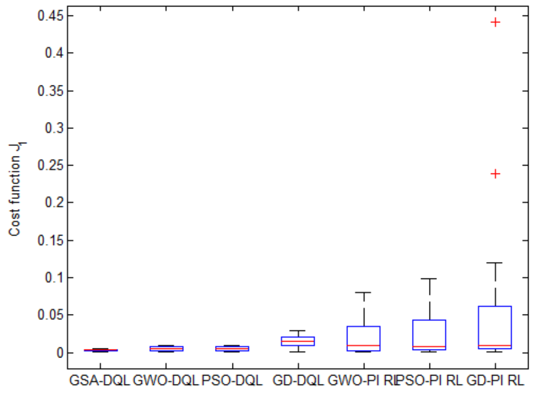

Fig. 6. ANOVA test of minimum cost function J_1 for all algorithms.

TABLE I
RANKS, STATISTICS AND RELATED P-VALUES OF SEVEN 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
Statistic	49.91	58.47	12.17
p-value	$5.89\cdot10^{-9}$	$9.1878\cdot10^{-11}$	$2.077\cdot10^{-11}$

Table II 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.

TABLE II
COMPARISON OF CONTROL SYSTEM PERFORMANCE INDICES WITH CONTROLLERS TUNED BY SEVEN 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

The best algorithm as far as the settling time is concerned is GWO-PI RL, followed by GD-PI RL and GSA-DQL. 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-PI RL, followed by GSA-DQL and GWO-DQL. These conclusions obtained from Table II 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.

The integrator is introduced to ensure zero steady-state error; experimental data to confirm this conclusion is also included in [22] and [23]. A sample of control system response is illustrated in Fig. 7, 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 (14).

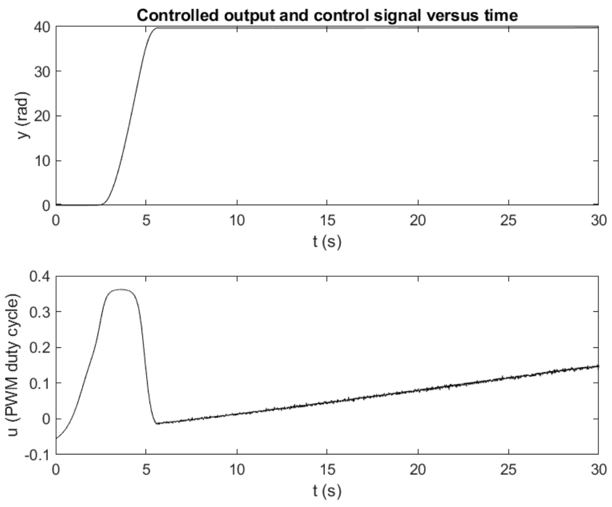

Fig. 7. Real-time experimental system responses y and u when using the PSO-PI RL algorithm [23].

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. 7.

V. CONCLUSION

This paper carried out the performance analysis of two 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 were implemented as Neural Network (NN)-based state feedback controllers and applied to the position control of a nonlinear servo system.

The algorithms were suggested in the recent papers [22] and [23] of the first four authors. The real-time experiments and the non-parametric statistical analysis were carried out with the help of the fifth author.

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 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.

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.

ACKNOWLEDGMENT

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.

REFERENCES

[1] R.-E. Precup, R.-C. Roman, and A. Safaei, *Data-Driven Model-Free Controllers, 1st Ed*. Boca Raton, FL: CRC Press, Taylor & Francis, 2022.

[2] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction, 2nd Ed*. Cambridge, MA, London: MIT Press, 2017.

[3] R. S. Sutton, A. G. Barto, and R. J. Williams, "Reinforcement learning is direct adaptive optimal control," *IEEE Control Syst. Mag.*, vol. 12, no. 2, pp. 19–22, Apr. 1992.

[4] L. Busoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: performance, stability, and deep approximators," *Annu. Rev. Control*, vol. 46, pp. 8–28, Dec. 2018.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, and D. Wierstra, M. Riedmiller, "Playing Atari with deep reinforcement learning," arXiv:1312.5602, Dec. 2013.

[6] R.-E. Precup and R.-C. David, *Nature-inspired Optimization Algorithms for Fuzzy Controlled Servo Systems*. Oxford: Butterworth-Heinemann, Elsevier, 2019.

[7] R.-E. Precup, P. Angelov, B. S. J. Costa, and M. Sayed-Mouchaweh, "An overview on fault diagnosis and nature-inspired optimal control of industrial process applications," *Comput. Ind.*, vol. 74, pp. 75–94, Dec. 2015.

[8] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," arXiv:1712.06567, Dec. 2017.

[9] A. Sehgal, H. M. La, S. J. Louis, and H. Nguyen, "Deep reinforcement learning using genetic algorithm for parameter optimization," in *Proc. 2019 3rd IEEE International Conference on Robotic Computing*, Naples, Italy, 2019, pp. 596–601.

[10] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," arXiv:1703.03864, Mar. 2017.

[11] D. A. Goulart and R. D. Pereira, "Autonomous pH control by reinforcement learning for electroplating industry wastewater," *Comput. Chem. Eng.*, vol. 140, paper 106909, Sep. 2020.

[12] H.-W. Lin, Q.-Y. Wu, D.-R. Liu, B. Zhao, and Q.-M. Yang, "Fault tolerant control for nonlinear systems based on adaptive dynamic programming with particle swarm optimization," in *Proc 2019 10th International Conference on Intelligent Control and Information Processing*, Marrakesh, Morocco, 2019, pp. 322–326.

[13] X. Liu, B. Zhao, and D. Liu, "Fault tolerant tracking control for nonlinear systems with actuator failures through particle swarm optimization-based adaptive dynamic programming," *Appl. Soft Comput.*, vol. 97, part A, paper 106766, Dec. 2020.

[14] D. Hein, A. Hentschel, T. Runkler, and S. Udluft, "Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies," *Eng. Appl. Artif. Intell.*, vol. 65, pp. 87–98, Oct. 2017.

[15] G. S. Piperagkas, G. Georgoulas, K. E. Parsopoulos, C. D. Stylios, and A. C. Likas, "Integrating particle swarm optimization with reinforcement learning in noisy problems," in *Proc. 14th Annual Conference on Genetic and Evolutionary Computation*, Philadelphia, PA, USA, 2012, pp. 65–72.

[16] H. Iima and Y. Kuroe, "Swarm reinforcement learning algorithms based on particle swarm optimization," in *Proc. 2008 IEEE International Conference on Systems, Man and Cybernetics*, Singapore, Singapore, 2008, pp. 1110–1115.

[17] D. Hein, A. Hentschel, T. Runkler, and S. Udluft, "Reinforcement learning with Particle Swarm Optimization Policy (PSO-P) in continuous state and action spaces," *Int. J. Swarm Intell. Res.*, vol. 7, no. 3, pp. 23–42, Sep. 2016.

[18] S. I. Meerza, M. Islam, and M. M. Uzzal, "Q-learning based particle swarm optimization algorithm for optimal path planning of swarm of mobile robots," in *Proc. 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology*, Dhaka, Bangladesh, 2019, pp. 1–5.

[19] Y.-Z. Gao, J.-W. Ye, Y.-M. Chen, and F.-L. Liang, "Q-learning based on particle swarm optimization for positioning system of underwater vehicles," in *Proc. 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, Shanghai, China, 2009, vol. 2, pp. 68–71.

[20] P. Zhang, H. Li, Q. P. Ha, Z.-Y. Yin, and R.-P. Chen, "Reinforcement learning based optimizer for improvement of predicting tunneling-induced ground responses," *Adv. Eng. Informat.*, vol. 45, paper 101097, Aug. 2020.

[21] S. Mirjalili, "How effective is the grey wolf optimizer in training multi-layer perceptrons," *Appl. Intell.*, vol. 43, no. 1, pp. 150–161, Jul. 2015.

[22] I. A. Zamfirache, R.-E. Precup, R.-C. Roman, and E. M. Petriu, "Reinforcement learning-based control using Q-learning and gravitational search algorithm with experimental validation on a nonlinear servo system," *Inf. Sci.*, vol. 583, pp. 99–120, Jan. 2022.

[23] I. A. Zamfirache, R.-E. Precup, R.-C. Roman, and E. M. Petriu, "Policy iteration reinforcement learning-based control using a grey wolf optimizer algorithm," *Inf. Sci.*, vol. 585, pp. 162–175, Mar. 2022.

[24] R.-E. Precup, R.-C. David, and E. M. Petriu, "Grey wolf optimizer algorithm-based tuning of fuzzy control systems with reduced parametric sensitivity," *IEEE Trans. Ind. Electron.*, vol. 64, no. 1, pp. 527–534, Jan. 2017.

[25] I. A. Zamfirache, R.-E. Precup, R.-C. Roman, E. M. Petriu, and M.-M. Damian. (May 2022). *Data obtained by 30 independent runs of all algorithms*. [Online]. Available: http://www.aut.upt.ro/~rprecup/Data_ECC.m.