10.24425/acs.2021.138691

Archives of Control Sciences
Volume 31(LXVII), 2021
No. 3*, pages 527–551*

Self-improving Q-learning based controller for a class of dynamical processes

Jakub MUSIAL, Krzysztof STEBEL and Jacek CZECZOT

This paper presents how Q-learning algorithm can be applied as a general-purpose self-improving controller for use in industrial automation as a substitute for conventional PI controller implemented without proper tuning. Traditional Q-learning approach is redefined to better fit the applications in practical control loops, including new definition of the goal state by the closed loop reference trajectory and discretization of state space and accessible actions (manipulating variables). Properties of Q-learning algorithm are investigated in terms of practical applicability with a special emphasis on initializing of Q-matrix based only on preliminary PI tunings to ensure bumpless switching between existing controller and replacing Q-learning algorithm. A general approach for design of Q-matrix and learning policy is suggested and the concept is systematically validated by simulation in the application to control two examples of processes exhibiting first order dynamics and oscillatory second order dynamics. Results show that on-line learning using interaction with controlled process is possible and it ensures significant improvement in control performance compared to arbitrarily tuned PI controller.

Key words: process control, Q-learning algorithm, reinforcement learning, intelligent control, on-line learning

1. Introduction

In industrial control systems, over 90% still use conventional PI(D) controllers [16]. Potentially, in majority of cases, they can ensure satisfying control performance, however, in 80% their tuning is far from optimal [19]. Moreover, almost 30% have not been tuned at all and operate with default tunings. Consequently, it results in deteriorated performance which leads to increased energy

All Authors are with Silesian University of Technology, Faculty of Automatic Control, Electronics and Computer Science, Department of Automatic Control and Robotics, 44-100 Gliwice, ul. Akademicka 16, Poland. J. Czeczot is the corresponding author, e-mail: jacek.czeczot@polsl.pl

This work was financed by the grant from SUT – subsidy for maintaining and developing the research potential in 2020: K. Stebel, J. Czeczot (grant 02/060/BK_20/0007, BK-276/RAU3/2020) and J. Musial (grant BKM-724/RAU3/2020).

Received 19.03.2021. Revised 22.08.2021.

consumption, medium wasting (chilled water or steam) and faster degradation of control equipment [4]. Obviously, performance improvement increases overall efficiency of manufacturing processes, especially if account is taken of the ever increasing economic and environmental requirements.

Many advanced controller design methods have been developed that promise to improve control performance but at the cost of increasing complexity. At the same time, the most significant bottlenecks in practical application of such advanced control: difficulties in implementation in the form of general-purpose function blocks and lack of clear tuning rules, not to mention autotuning. Consequently, even if bad performance of currently operating PID-based control loop is observed, its improvement is difficult because it requires time consuming retuning of the controller or even more challenging substituting it by more advanced controller. Thus, probably the most awaited solution for this difficulty is self-improving controller that should be able to substitute existing controller exhibiting poor performance and gradually improve the performance by learning from interaction with controlled dynamical process. From practical viewpoint, such a solution is still an unrealistic promise but some initial attempts using machine learning techniques which make it implementable can be found in literature.

In this paper, Q-learning technique is suggested to design self-improving controller as the most promising attempt with relatively many reported initial studies. In a sense, this approach can be considered as similar to Iterative Learning Control (ILC) that is widely used in automatic and robotics for batch processes [3, 20]. Different applications of Q-learning in automation have been already reported. Boubertakh et al. [1] propose to use Q-learning for optimization of the performance of fuzzy PID controllers. Q-learning algorithm was compared with conventional PID controller for fermentation process [7], and nonlinear liquid level stabilization [8]. Quality control of chemical production line based on application of Q-learning technique in fuzzy multi-agent system is reported in [12]. In [16,17], application of Q-learning algorithm for pH control are reported while Syafiie et al. [18] present its implementation for wastewater treatment control. Examples of applications of Q-learning can be also found in robotics [2,11] and automotive [10]. In [9], it is shown how to use Q-learning approach to generate assembler encoding program that supports implementation of artificial neural networks. Preliminary results on general aspects of application of Q-learning algorithm in industrial automation are discussed by Stebel [13].

Even if applications of Q-learning algorithm for design or performance improvement of control systems have been reported relatively often, proposed solutions were dedicated to very specific cases and they suffer from lack of generality. Thus, there are still significant difficulties that have to be managed when it comes to general-purpose application in practical control systems. Firstly, definition and discretization of state space describing current state of control system is not clearly established at the acceptable level of generality. Secondly, definition of set of acceptable actions in relation to limitations of real control system is not determined. Thirdly, even if some practical applications were reported, these cases were limited to processes of exhibiting simple (first order) dynamics. Thus, extension to processes with more complex dynamics remains still a challenge. Finally, stage of initial learning (based on model or worsely, during normal operation) is not acceptable for practitioners. This paper concentrates on systematic study and settle of most important difficulties that limit practical implementation of Q-learning algorithm as a self-improving control strategy. The major novelty of this paper results from the following contributions:

- redefining state space for Q-learning algorithm based on definition of reference trajectory of control system as a *goal state*,
- proposing reliable method for initialization of Q-learning algorithm based on tunings of existing PI controller,
- systematic validation of the suggested approach not only for first order but also for oscillatory second order processes.

2. Short introduction to q-learning algorithm

Q-learning algorithm [21] belongs to a class of *reinforcement learning* methods [14] that combine machine learning with optimisation. More precisely, based on trial and error learning inspired by psychology, Q-learning algorithm applies the reward/punishment policy that provides a suboptimal solution (reaching the *goal state S*) of even very complex dynamical problems, for which the analytical solution is not available, e.g. due to the lack of accurate model. Based on this policy, Q-learning algorithm learns directly from interaction between active decision-making *Agent* and its dynamical target system. Learning process allows for modification of *Agent* behavior to adapt to varying properties of a target system. At the same time, when these properties are constant, Q-learning algorithm is able to improve its actions if they are not optimal at the beginning of learning.

Q-learning algorithm is based on learning policy that in general can be described by the following iterative formula:

$$Q\left(s_t, a_t\right) \leftarrow Q_p\left(s_t, a_t\right) + \alpha\left[R + \gamma \cdot \max_a Q\left(s_{t+1}, a_{t+1}\right) - Q_p\left(s_t, a_t\right)\right], \tag{1}$$

where t is discrete time instant, s and a respectively denote state and action that should be taken at this state, $Q\left(s_t, a_t\right)$ denotes the element (Q-value) of so-called Q-matrix that represents reward for applying action a when the system is at the state s. Additionally, in Eq. (1), $Q_p\left(s_t, a_t\right)$ is the value of $Q\left(s_t, a_t\right)$ before update, $Q\left(s_{t+1}, a_{t+1}\right)$ indicates the state to which the system will move from $Q\left(s_t, a_t\right)$,

$\alpha \in [0,1]$ is the learning rate, R is the reward assigned to state s, $\gamma \in [0,1]$ is the discount factor and $\max_a$ selects the optimal action for taking which the maximum reward was encountered. In general, parameters α and γ must be considered as turning parameters of Q-learning algorithm. However, even if there are some tuning tips, tuning of Eq. (1) is still not deterministic and it requires trial and error approach.

Reward R is assigned depending if the system is at the *goal state* ($s = S$) or not. Precisely, if $s = S$, then $R = 1$ is applied, otherwise $R = 0$. During continuous learning, *Agent* must take actions other than optimal one that was previously assigned to current state. For simultaneous learning and normal operation, exploration/exploitation problem must be solved depending on the ratio between exploration and exploitation. For normal operation (exploitation), previously learned optimal actions are taken while for further learning (exploration), in general, random actions are taken to force different behavior and consequently to check if action so far considered as optimal should be updated. When Q-learning algorithm is to operate real system, exploitation should have higher priority over exploration, which impacts on performance but extend the learning time.

3. Application for process control

3.1. Statement of the problem

In this paper, it is assumed that proposed Q-learning algorithm should be able to substitute PI controller applied in conventional industrial single-loop control system as shown in Fig. 1. The control goal is defined to keep *Process* output Y equal to its desired setpoint Y_{sp} (tracking) by adjusting manipulating variable U in the presence of load disturbance d. PI controller is pre-tuned so it provides more or less acceptable control performance but this performance can be far from optimal that is defined by desired tracking trajectory. Therefore, replacing the PI controller with the Q-learning algorithm is intended to allow for a continuous improvement of the control performance thanks to the potential learning from the behavior of the *Process*. In this work, considerations are limited to first order (FO) and second order (SO) processes.

In order to ensure potential practical applicability of the suggested solution, switching between PI controler and Q-learning algorithm has to be bumpless. In other words, after the switching, Q-learning algorithm should be able to provide exactly the same control performance as substituted PI controller. Because it is assumed that *Process* model is unknown, initial learning by simulation can not be used so this feature must be ensured by proper preliminary tuning of Q-learning algorithm based on tunings of substituted PI controller. At the same time, on-line further learning is to be ensured while normal operation of the system,

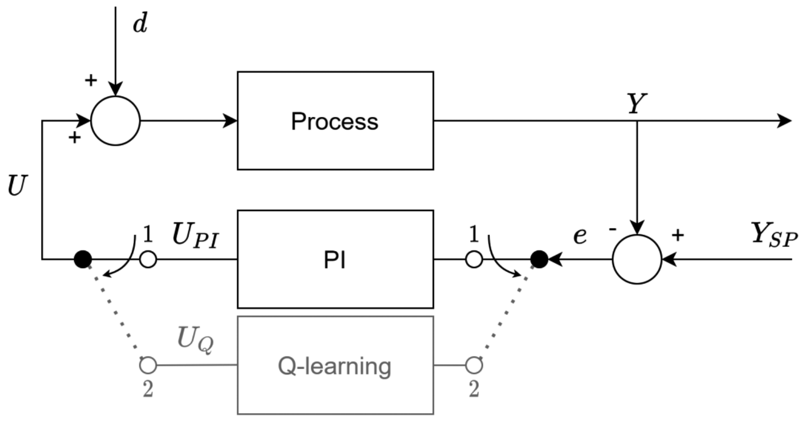


Figure 1: Schematic diagram of the considered control system. Initially, the system is operated by conventional PI controller. Q-learning algorithm should be designed to ensure not only bumpless control after switching from *1* to *2* but also self-improving properties by on-line learning from interaction with a process

so reasonable balance between exploration and exploitation must be provided, especially if learning requires additional excitation of the process.

In control theory, control performance of the closed loop system is quantified by control error $e = Y_{sp} - Y$ and by its time evolution. Thus, the simplest and most intuitive definition of desired goal state for considered problem $S = (e = 0)$. Based on this definition, application of Q-learning for process control was discussed in [16]. Authors suggest to define a number of states of closed-loop system based on the current value of control error e. These states are uniformly distributed around the *goal state* $S = (|e| \leqslant e_{\min})$ and for each state, *reward/punishment* policy is proposed to keep control error in this goal state S. Desired threshold $e_{\min}$ is a positive value complying impact of measurement noise that occurs in particular closed-loop control system. It was reported that this approach provides acceptable control performance for certain cases but from practical viewpoint, it has a significant drawback. Defining the *goal state* as $S = (|e| \leqslant e_{\min})$ does not ensure good control performance without significant overregulations and oscillations in transients. Figure 2 shows that when the *goal state* of control system is defined only as $S = (|e| \leqslant e_{\min})$, for both oscillatory and overregulated cases this state can be reached many times before error decays, which is potentially misleading for Q-learning algorithm.

In practical cases, every closed-loop control system should be designed based on the predefined trajectory that describes desired closed-loop dynamics. Thus, in this paper, the *goal state* of control system is redefined based on required reference trajectory. Namely, suggested *goal state* is defined as follows: for each value $|e| > e_{\min}$, controller should take the actions and change the manipulating

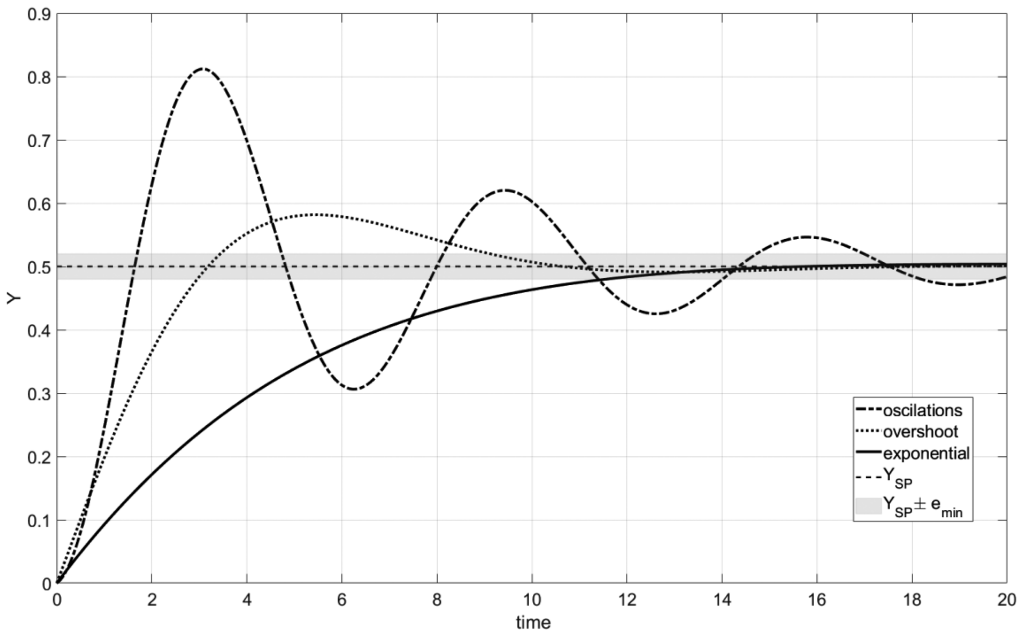


Figure 2: Three different trajectories representing potential tracking properties in real control systems. Note that only for exponential one, definition of goal state $S = (|e| \leqslant e_{\min})$ directly represents desired steady state of control system. For two other cases, control system goes through this state twice (overshoot) or even more times (oscillatory), still being in transient

variable U to ensure not only that the control error is regulated to $|e| \leqslant e_{\min}$ but also that this regulation takes place with dynamics defined by reference trajectory.

Reference trajectory is defined as dynamical equation describing decay of control error e. The most desired trajectory is exponential decay with adjusted time constant $T_e > 0$ that is considered as the design parameter for closed-loop system. This exponential trajectory is described as follows:

$$F(e, \dot{e}) = \dot{e} + \frac{1}{T_e} e = 0, \tag{2}$$

where $\dot{e}$ denotes time derivative of control error e and exemplary shape of this trajectory is shown in Fig. 2, jointly with undesirable oscillatory or overregulated error decay.

If the state of control system is defined as a deviation from reference trajectory (2), the *goal state* can be defined as $S = (|F(e, \dot{e})| \leqslant \sigma(e, \dot{e}))$ where $\sigma(e, \dot{e}) > 0$. Consequently, states of the system for Q-learning algorithm can be defined from Eq. (2), based on discretization of both e and $\dot{e}$ values for determining discretized state space.

For considered case, action $u_{Q,t}$ taken by Q-learning algorithm at sampling instant (t) can be converted to manipulating variable applied to a process $U_{Q,t}$ by the following formula:

$$U_{Q,t} = \begin{cases} U_{Q,t-1} + u_{Q,t} & \text{if } s \neq S, \\ U_{Q,t-1} & \text{otherwise,} \end{cases} \tag{3}$$

where $U_{Q,t-1}$ denotes manipulating variable applied to a process at previous sampling instant $(t-1)$. One can note that Eq. (3) is equivalent to action taken by conventional PI controller depending on control error and written in velocity form:

$$U_{\mathrm{PI},t} = U_{\mathrm{PI},t-1} + K_{\mathrm{PI}} T_s \left(\frac{1}{T_I} e_t + \dot{e}_t \right), \tag{4}$$

where K_{PI} and T_I respectively denote gain and integral time of PI controller and T_s is a sampling time. Thus, by combining Eqs. (3) and (4), one can obtain:

$$u_{Q,t} = K_{\mathrm{PI}} T_s \left(\frac{1}{T_I} e_t + \dot{e}_t \right). \tag{5}$$

This property leads to the conclusion that has very strong impact on practical applicability of Q-learning algorithm in industrial control loops. Namely, it allows for deterministic definition of the values of actions accessible for Q-learning algorithm and suggesting initialization of Q-matrix to ensure bumpless switching.

3.2. Definition of states space and action values

Application of Q-learning algorithm for use in control loops requires defining states and actions and in this paper, the definition is suggested to be consistent with the definition of *goal state* given by reference trajectory (2) according to conventional PI controller shown by Eq. (5).

Based on reference trajectory (2), a state of control system is defined by actual values of control error e and its time derivative $\dot{e}$. Thus, to ensure finite number of states, space $(e, \dot{e})$ must be discretized. Not uniform discretization should be applied because for small values of e and $\dot{e}$, more dense discretization is required to ensure ability of precise control when the system is close to the desired steady state. If e and $\dot{e}$ is higher, then precision of the discretization may be lower. Thus, for discretization of e, three design parameters must be determined: minimal control error $e_{\min}$ representing dead zone, maximal expected control error $e_{\max}$ and time constant of the desired reference trajectory T_e. Then, the following formulas are suggested for determining boundaries of consecutive e intervals:

$$e_1 = e_{\min}, \tag{6a}$$

$$e_{i+1} = e_i \frac{T_e + T_s}{T_e}, \quad \text{for } i \text{ starting from 1, while } e_{i+1} \leqslant e_{\max}, \tag{6b}$$

where i denotes consecutive e interval number.

Suggested discretization of $\dot{e}$ requires defining of two additional design parameters: k denoting a number of $\dot{e}$ intervals between $\dot{e} = 0$ and $\dot{e} < \frac{1}{T_e}e$ ($\frac{1}{T_e}e$ represents reference trajectory) and w denoting number of the same k intervals for $\dot{e} > 0$. Then, the boundaries of consecutive $\dot{e}$ intervals can be calculated as:

$$\dot{e}_{i,j=1} = \frac{e_i}{k \cdot T_e}, \tag{7a}$$

$$\dot{e}_{i,j} = \dot{e}_{i,j-1} + \frac{e_i}{k \cdot T_e}, \quad \text{for } j \leqslant k \cdot w, \tag{7b}$$

where j denotes consecutive $\dot{e}$ interval number. Note that boundaries of $\dot{e}$ intervals are calculated separately for each e boundary calculated by Eqs.(6). It ensures that for bigger values of e and $\dot{e}$, precision of discretization decreases as it was suggested before.

Once the continuous state space is discretized, a discrete number of acceptable actions must generated for each state. These actions are generated using middle points of e, $\dot{e}$ intervals that are defined for each state as:

$$e_1^m = 0, \tag{8a}$$

$$e_{l+1}^m = \left(e_l + e_l \frac{T_e + T_s}{T_e}\right)/2, \quad \text{for } l \leqslant i + 1, \tag{8b}$$

for e intervals, and:

$$\dot{e}_{i,j=1}^m = 0, \tag{9a}$$

$$\dot{e}_{i,j}^m = \dot{e}_{i,j-1}^m + \frac{e_i^m}{k \cdot T_e}, \quad \text{for } j \leqslant k \cdot w + 1, \tag{9b}$$

for $\dot{e}$ intervals, where e_i^m and $\dot{e}_{i,j}^m$ denote middle points of e, $\dot{e}$ intervals, respectively.

Then, using previously calculated middle points discretized actions for each state are generated similarly to PI controller given by Eq. (5) as:

$$u_{Q\,i,j} = K_{\mathrm{PI}} T_s \left(\frac{1}{T_e} e_i^m + \dot{e}_{i,j}^m\right), \quad \text{for } j \leqslant k \cdot w + 1. \tag{10}$$

All calculations defined by Eqs. (6)–(10) are given for positive values of e, $\dot{e}$. Extension for other e, $\dot{e}$ space quadrants is straightforward due to symmetry of both e, $\dot{e}$ axes.

Figure 3 shows the suggested concept of discretization of the space (e, $\dot{e}$) presented only for the first quadrant. Other quadrants are discretized by symmetry in the same way.

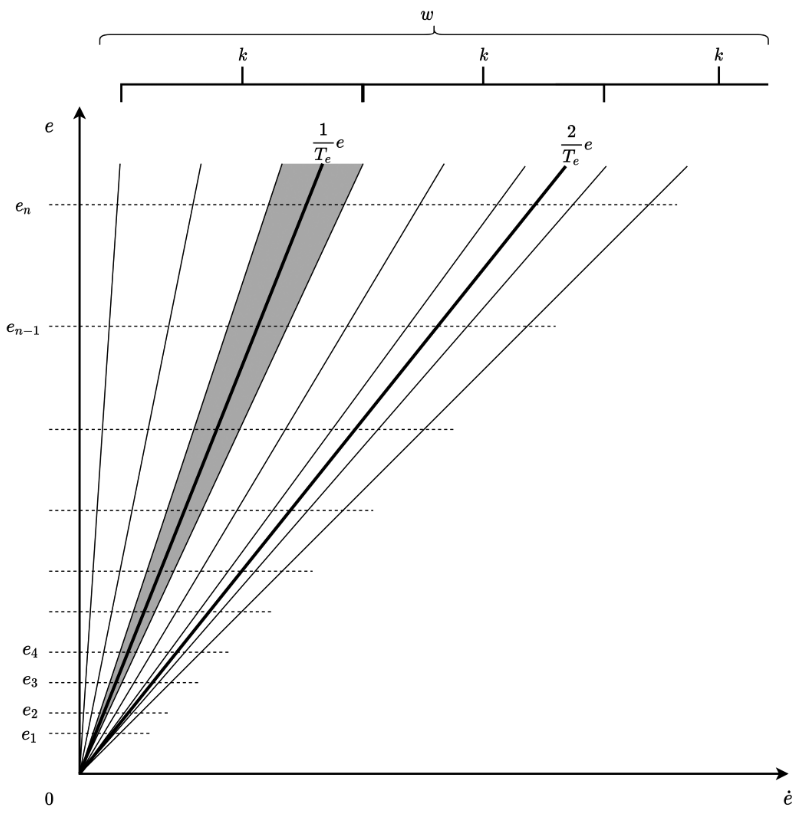


Figure 3: Definition of system states based on discretization of e and $\dot{e}$ values. Straight line $\frac{1}{T_e}e$ represents reference trajectory. Gray area represents *goal state* of the system

Note that the suggested method of discretization of e and $\dot{e}$ values into the state space of Q-learning algorithm results in the fact that Q-values are stored in cuboid array. At the same time, values of actions calculated for each e_i by Eq. (10) are stored in rectangular array.

3.3. Initialization of Q-learning algorithm and self-improving by on-line learning

In conventional Q-learning approach, Q-matrix does not contain any knowledge about process behavior at the initial stage thus an initial learning is required. This stage can be performed by simulation but only if relatively precise model of a process is known. However, it is rare case in practice. Moreover, even if it is known, there are many well established methods that can be directly applied for deriving model-based advanced controllers that provide very good control performance with no learning required. The other possibility is direct learning from interaction with real process but in practice, this approach is absolutely unacceptable due to potential very large perturbations in the process operation required at the initial stage of learning.

In this paper, using similarity with PI controller shown by Eq. (5), a method for deterministic initialization of Q-matrix is proposed based only on tunings of conventional PI controller currently operating real process. This approach is partially uses the method suggested in [13] and it allows to avoid initial learning. Moreover, it ensures that after such initialization of Q-matrix, Q-learning algorithm provides exactly the same control performance as PI controller which tunings were used for Q-matrix initialization.

Initialization method uses actions calculated by Eq. (10) and stored in actions array. After initial reset of all Q-values stored in cuboid Q-matrix, for each state, unitary weight is adjusted in the cell representing the action which value is closest to the action that would be applied by PI controller with the same settings that were used for generating actions by Eq. (10). This method provides very similar control performance as PI controller does, and eventual small differences result only from discretization accuracy.

After the suggested initialization, Q-learning algorithm can substitute PI controller without losses in control performance. Now continuous improvement is expected during operation the real process. Thus, the exploration/exploitation problem must be solved to ensure balance between normal operation and on-line learning. It is suggested to solve this problem using ε-greedy method [6, 15] with additional limitation on the range, from which the actions are randomized for on-line learning. In standard Q-learning approach, actions are randomized with no limitation and it allows for relatively fast learning but in practice, such an approach could result in unacceptable disturbance of the process introduced intestinally by Q-learning algorithm only for learning. Limitation on range of draws (RD) extends the learning process but allows for safe operation of the process. This concept is shown in Fig. 4 for an example of $RD = 2$. If gray box represents the action so far considered optimal for current state s_t by adjusting maximal weight in Q-matrix ($\max_a Q(s_t, a_t) = n$), for learning, this optimal action is substituted by the other action drawn from the range $[n-RD,\ n+RD]$. According to ε-greedy method, this substitution takes place only if randomly drawn $\xi \leqslant \varepsilon$, providing ξ and ε are both within the range $(0, 1)$ and ε is arbitrarily adjusted by the user. Otherwise, optimal action is taken.

In practical application, RD must be adjusted according to acceptable fluctuations of the process output but if RD is too small, the control error cannot decay with the assumed reference trajectory because none of the actions that can be drawn to provide such property.

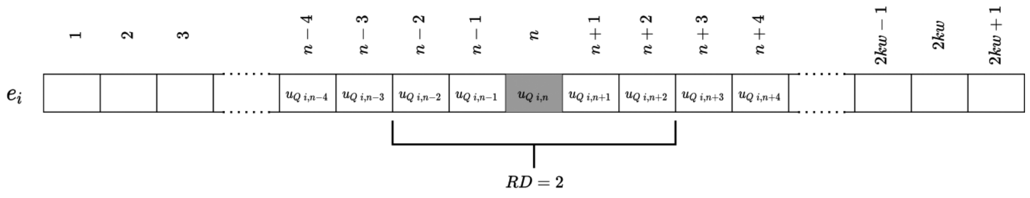


Figure 4: An example of one raw from actions rectangular table for a single discretised value of e_i. Gray box represents the action so far considered optimal by adjusting maximal weight in Q-matrix. $RD = 2$ shows an example of range of draw for actions adjusted for on-line learning

3.4. Adjustment of discount factor γ for practical applications

An important parameter of Q-learning algorithm is the discount factor γ which determines reward policy described by Eq. (1). Its value influences reward R distribution to the states that belong to a single transition path. At the same time, it determines the differences between Q-values assigned to each state. This property is very important if Q-learning algorithm is to be implemented in the platforms with limited decimal precision of calculations. Due to numerical rounding, too small difference between Q-values may cause the same Q-values assignment to different states. Then, the choice of optimal transition path is not ambiguous. Consequently, proper choice of the discount factor γ is very critical for practical applications of Q-learning algorithm in industrial control loops.

This choice can be suggested based on example of a single transition path following from the state s^N through $s^{N-1} \ldots s^1$, $s^0 = S$ (goal state) with $n = N \ldots 0$ denoting consecutive numbers of states in the transition path. If the system continuously remains at the *goal state* S, for Q-values $Q(s_{t+1}, a_{t+1}) = Q(s_t, a_t) = Q(S, a_t)$, so Eq. (1) can be written as:

$$\frac{\Delta t}{\alpha} \frac{Q(S, a_t) - Q_p(S, a_t)}{\Delta t} = R + (\gamma - 1)\, Q_p(S, a_t), \tag{11}$$

where Δt denotes sampling time. After assuming that Δt is small enough comparing to dynamics of closed loop system with Q-learning algorithm, Eq. (11) can be written in the continuous form:

$$\tau \frac{\mathrm{d}Q(S, a_t)}{\mathrm{d}t} = R + (\gamma - 1)\, Q_p(S, a_t), \tag{12}$$

where $\tau = \dfrac{\Delta t}{\alpha}$ is a time constant representing dynamics of changes of each single Q-value. Therefore, if the system remains in the *goal state* S, $\dfrac{\mathrm{d}Q(S, a_t)}{\mathrm{d}t} = 0$ then

Q-value assigned to this *goal state* has the maximal value:

$$Q(S, a_t) = \frac{R}{1-\gamma}\,. \tag{13}$$

Thus, after rearrangement, Eq. (12) describes distribution of this value to subsequent states s^n $(n = N \ldots 0)$ without any additional reward in transition states:

$$\tau \frac{\mathrm{d}Q(s_t^n, a_t)}{\mathrm{d}t} = -Q\left(s_t^n, a_t\right) + \gamma Q(s_{t+1}^{n-1} a_{t+1}), \tag{14}$$

and because $Q\left(s_t^0, a_t\right) = Q\left(S, a_t\right)$, $Q\left(s_t^n, a_t\right) = \gamma Q\left(s_{t+1}^{n-1}, a_{t+1}\right) = \gamma^n Q(S, a_t)$ so it can be written that:

$$Q(s_t^n, a_t) = \frac{R\gamma^n}{1-\gamma}\,. \tag{15}$$

Eq. (15) can be used to determine minimal value of γ that ensures discrimination between Q-values assigned to different states for assumed $\delta_Q > 0$ considered as a design parameter. Such a discrimination feature is defined as:

$$Q\left(s_t^N, a_t\right) - Q\left(s_{t+1}^{N-1}, a_{t+1}\right) > \delta_Q\,, \tag{16}$$

and after combining Eqs. (15) and (16), it can be obtained that:

$$\gamma^N Q\left(S, a_t\right) - \gamma^{N-1} Q\left(S, a_t\right) > \delta_Q\,, \tag{17}$$

which leads to the final condition for adjusting the value of γ:

$$\gamma > \sqrt[N-1]{\frac{\delta_Q}{R}}\,. \tag{18}$$

In Eq. (18), R is the reward value adjusted by the user. N should be adjusted based on the number of states in the longest reasonable transition path. In the considered case, it is suggested to adjust $N = k \cdot w + 1$. The value of δ_Q should be adjusted based on the decimal precision of calculations accessible for certain platform, on which Q-learning algorithm is to be implemented. For instance, if this platform is Programmable Logic Controller (PLC), its precision may be limited to four decimal places. Thus, in such a case, it is suggested to adjust $\delta_Q > 0.001$.

4. Simulation results

Potential applicability of the suggested approach was tested by simulation for two examples of processes represented by models: FO model $K_1(s) = \dfrac{1}{5s+1}$

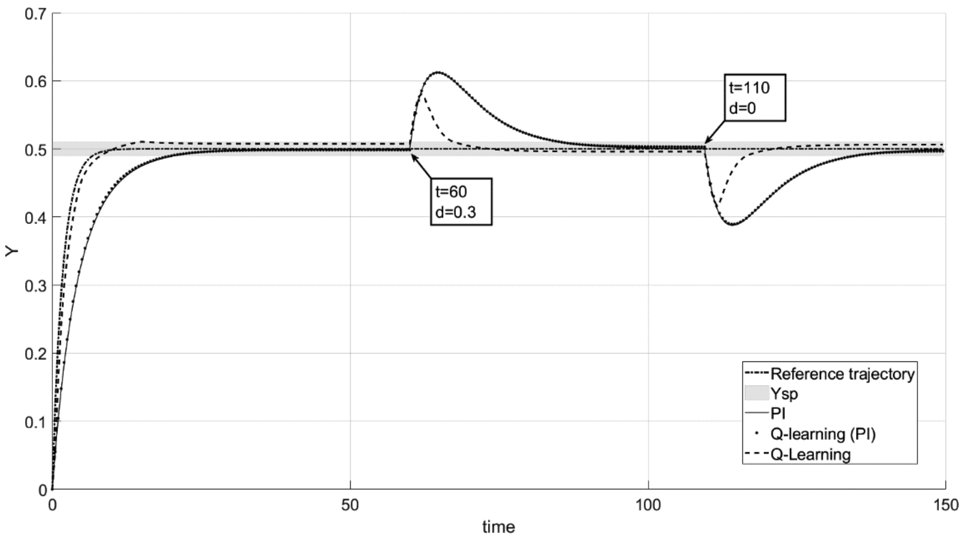
and SO model $K_2(s) = \dfrac{1}{5s^2 + 2s + 1}$. Note that the latter represent more complex dynamics that exhibits oscillatory behaviour. Both models were initially operated in control loop presented in Fig. 1 with PI controllers tuned as follows: $K_{\mathrm{PI}} = 1$, $T_I = 5$ for FO model and $K_{\mathrm{PI}} = 0.96$, $T_I = 10$ for SO model.

At the first stage of validation, for both models, Q-matrix was initialized by the suggested method using corresponding PI tunings to ensure the same initial control performance both for PI and Q-learning algorithm. Then, after adjusting $Y_{sp} = 0.5$ and replacing PI to Q-learning algorithm, learning stage was simulated by consecutive randomizing initial conditions for $e(0)$ and $\dot{e}(0)$. After each draw, control system was brought to a steady state, which represents a single learning epoch. After a number of epochs, Q-matrix indicates control actions that ensure more precise tracking of assumed reference trajectory.

Figure 5 shows comparative results for FO model between: PI controller, Q-learning algorithm which was initialized using on PI tunings without any additional learning and Q-learning after intensive learning procedure covering 10 000 epochs. Control performance of PI and Q-learning (PI) is the same due to affective initialization procedure. Then, after learning period, Q-learning algorithm ensures very precise tracking the reference trajectory defined by $T_e = 2$ and a significant improvement in disturbance rejection. Very similar results can be seen in Fig. 6 for SO oscillatory model. This time, control performances of PI controller and Q-learning (PI) algorithm are not the same but they are very similar, which ensures bumpless switching between both controllers. The difference results from the fact that when Q-learning algorithm ensures tracking the reference trajectory, it cannot take any action and the process output temporarily remains unchanged. After learning period of 10 000 epochs, Q-learning algorithm also ensures very significant improvement in control performance. Control system tracks the desired reference trajectory defined by $T_e = 6$ and disturbance rejection is more efficient. Oscillatory behaviour of manipulating variable U results from the fact that the model itself exhibits oscillatory SO dynamics. Thus, if reference trajectory is defined as exponential, it is a big challenge for any controller to ensure desired closed loop behaviour. Summarizing, the results presented in Figs. 5 and 6 prove potential ability of self-improvement feature of Q-learning algorithm.

Figure 7 shows how the control performance of Q-learning algorithm improves over time due to continuous learning after different number of epochs for both FO and oscillatory SO models. In both cases, Q-learning algorithms were initialized by corresponding PI tunings and the improvement is shown after 1000, 2000 and 3000 epochs. For both processes, improvement occurs during learning and it is visible even after 1000 epochs.

Learning abilities are strongly influenced by RD value that limits the range from which actions are drawn during on-line learning. Figure 8 shows how RD influences improvement in control performance from on-line learning for



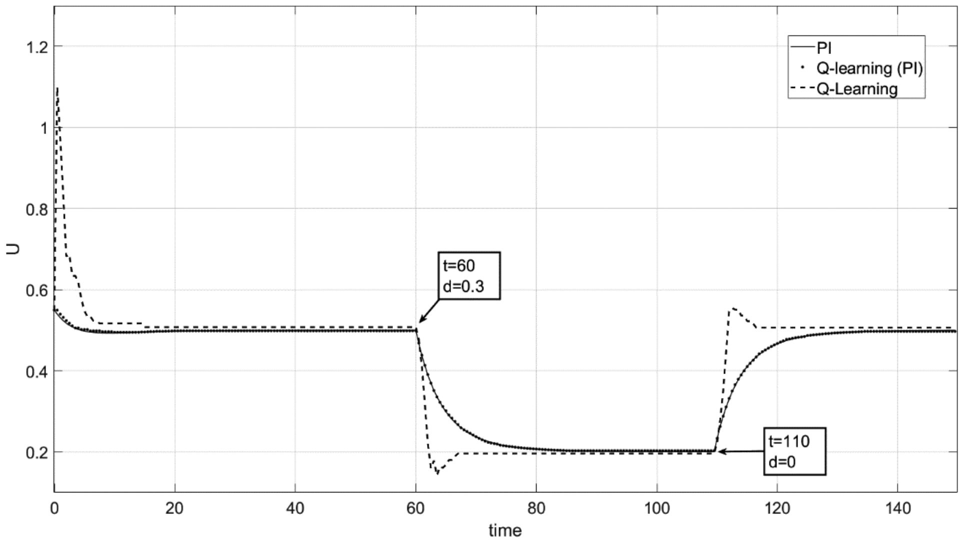


Figure 5: Tracking and disturbance rejection for Q-learning algorithm operating FO model after 10 000 learning epochs, for $w = 4$, $k = 10$. Upper diagram shows process output Y and lower diagram shows manipulating variable U. Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings

both considered models. In both cases, higher value of *RD* results in greater improvement after 5000 epochs but it was encountered that for oscillatory SO model, *RD* value requires stricter limitation because for higher values of *RD*, instead of improvement, learning process can lead to deterioration in control

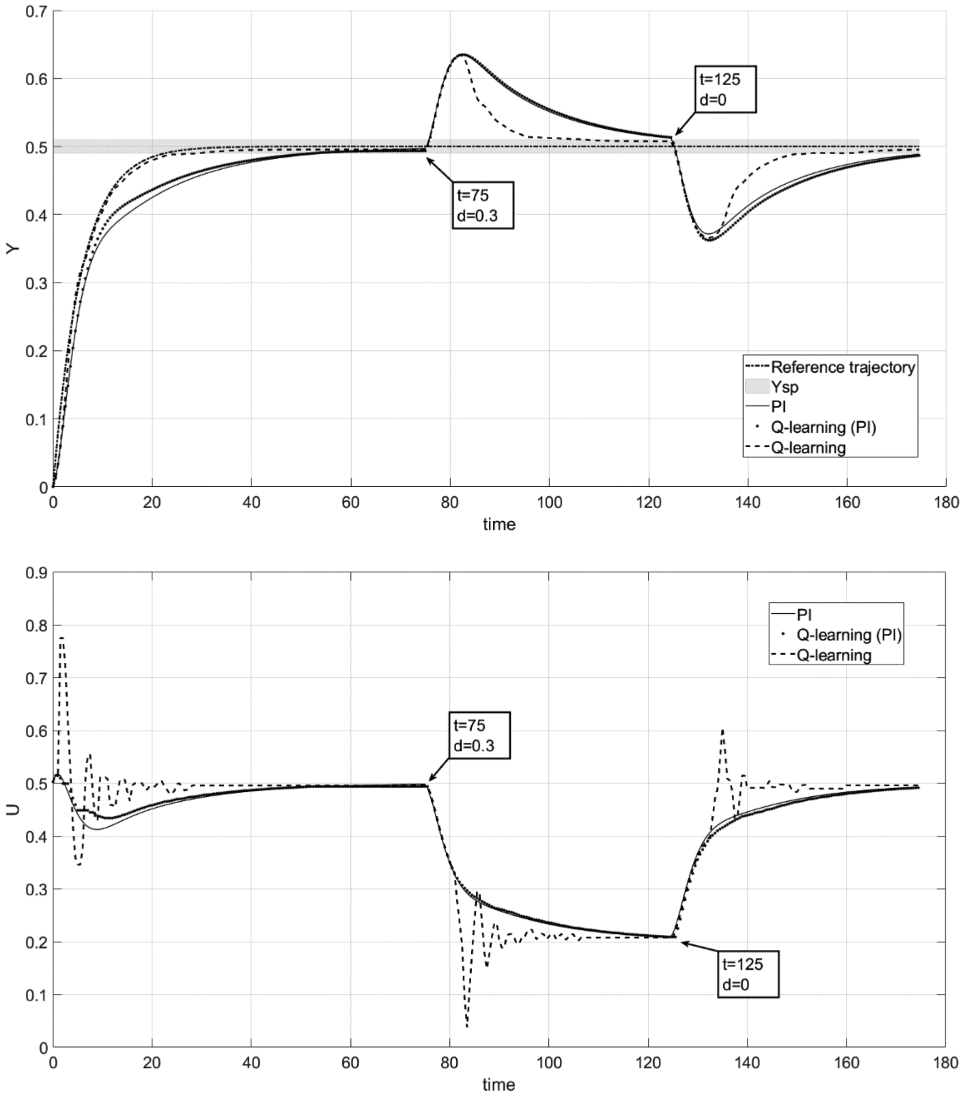
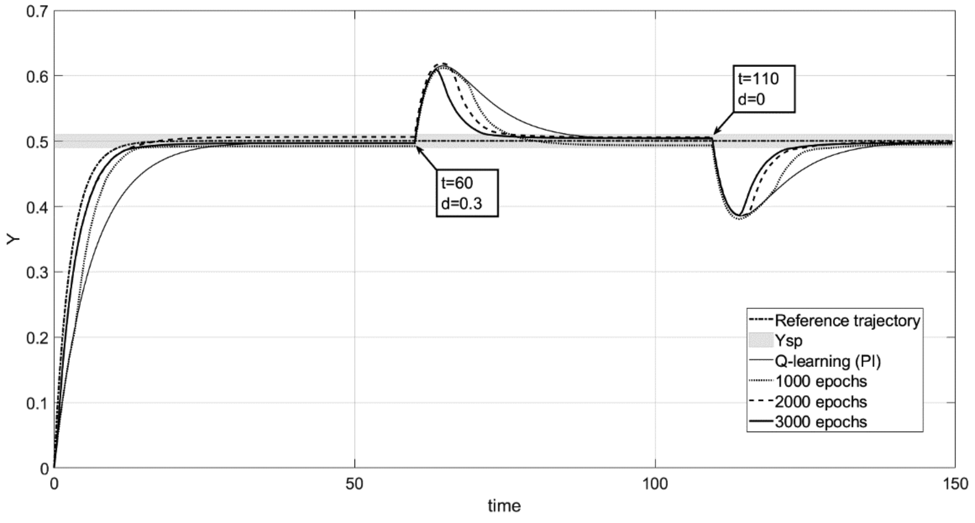


Figure 6: Tracking and disturbance rejection for Q-learning algorithm operating oscillatory SO model after 10 000 learning epochs, for $w = 6$, $k = 14$, $RD = 40$. Upper diagram shows process output Y and lower diagram shows manipulating variable U. Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings

performance comparing to Q-learning algorithm initialized by PI tunings. On the other hand, too small value of RD results in very small improvement in control performance even for a huge number of learning epochs.

Influence of the value of ε on control performance of Q-learning algorithm after on-line learning is shown in Fig. 9 for FO and oscillatory SO models. This



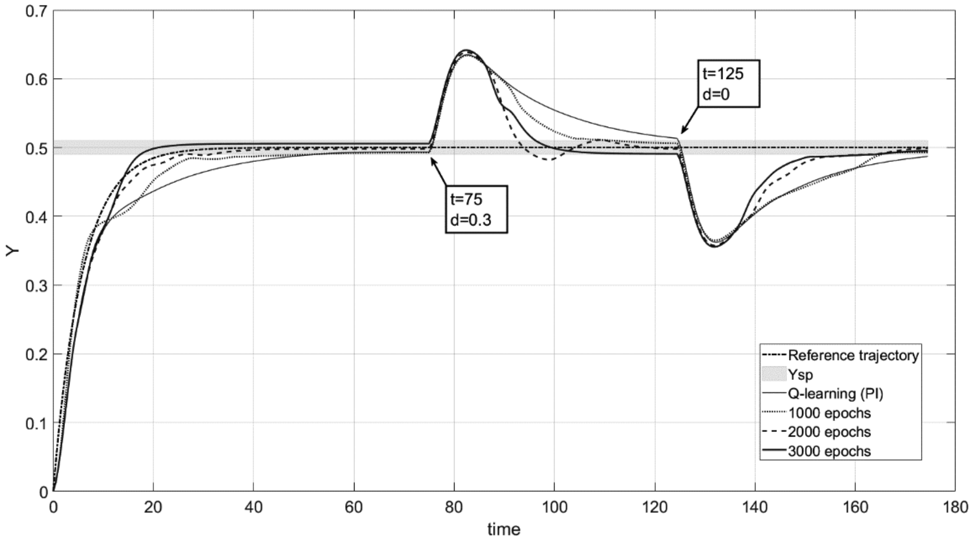
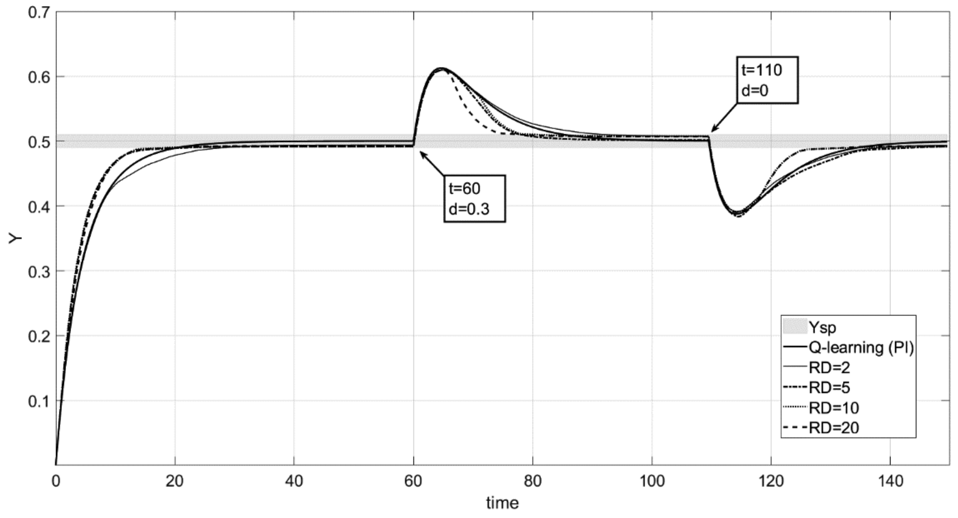


Figure 7: Improvement in control performance for Q-learning algorithm operating FO model (upper diagram) and oscillatory SO model (lower diagram). Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings

value determines a tradeoff between exploitation and exploration and once again, the results are as expected. For both cases, higher value of ε results in more significant improvement after the same number of learning epochs.

Self-improving property requires ability of on-line learning during normal operation. Thus, the second stage of validation consists in experiments that are to simulate realistic on-line learning when process is frequently disturbed by apply-



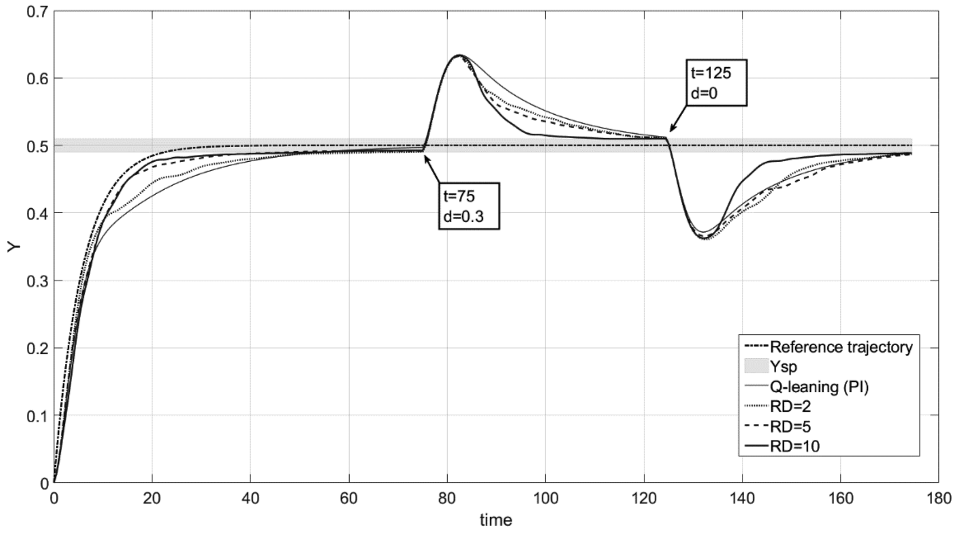
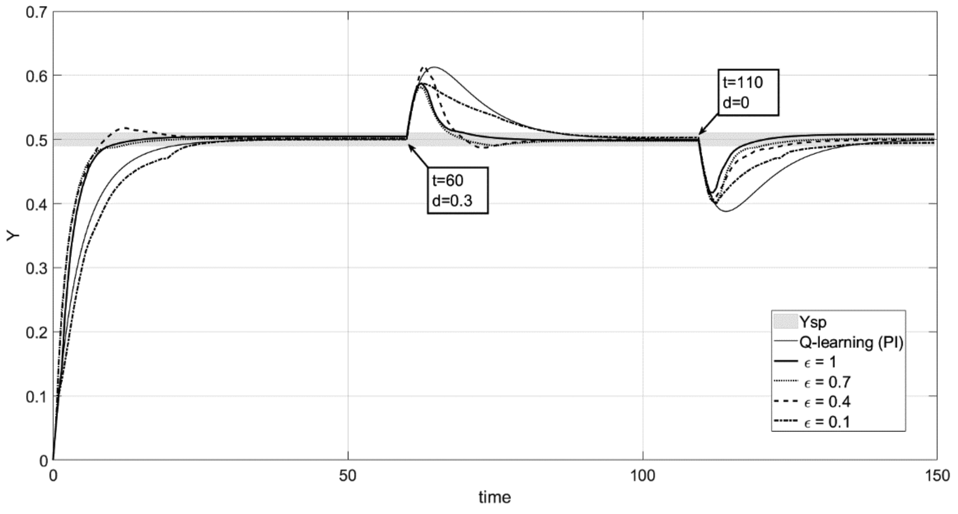


Figure 8: Influence of *RD* on improvement in control performance after 5000 epochs for Q-learning algorithm operating FO model (upper diagram) and oscillatory SO model (lower diagram). Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings

ing a huge number of consecutive step changes of load disturbance *d* of amplitude randomized within the range (0, 0.5). At the beginning, Q-matrix is initialized based on PI tunings. After applying each step change, Q-learning rejects a disturbance and at the same time, learns by taking modified actions. Consecutive step changes of load disturbance are separated by a period which is required to lead the process to steady state and a period between two load changes form a single



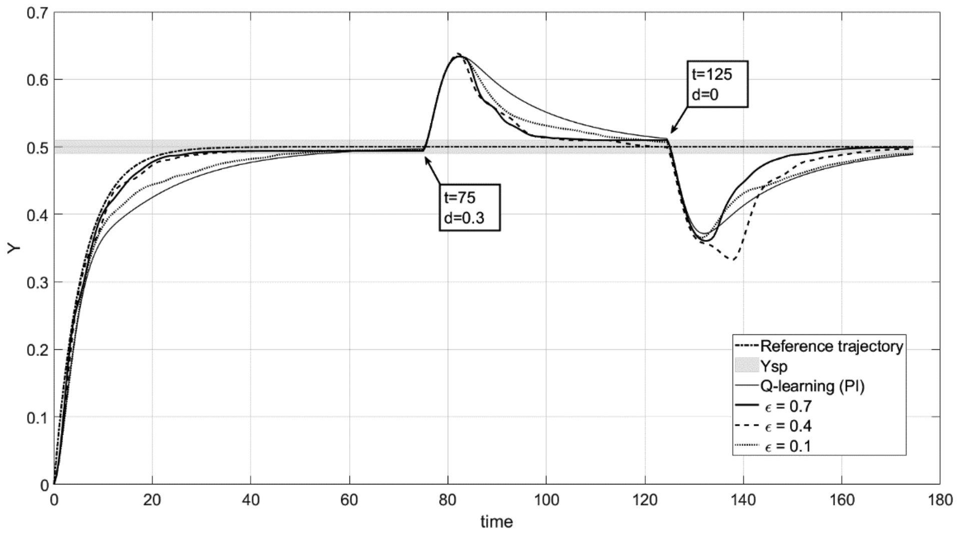
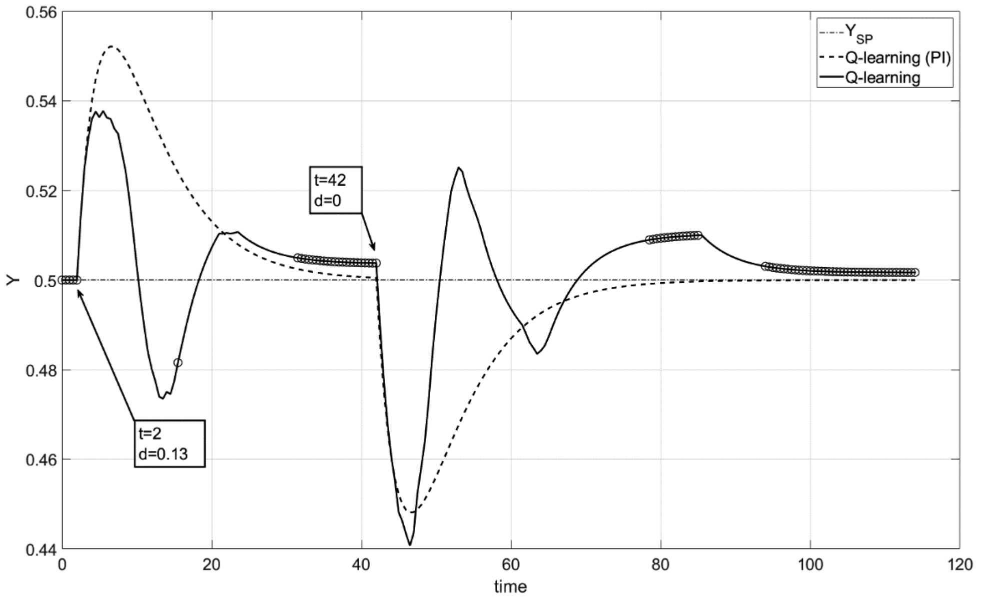


Figure 9: Influence of ε on improvement in control performance after 5000 epochs for Q-learning algorithm operating FO model (upper diagram) and oscillatory SO model (lower diagram). Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings

epoch. Figures 10 and 11 show an example of two consecutive learning epochs for FO model and for oscillatory SO model, respectively. Comparison between not learned Q-learning initialized by PI tunings and Q-learning algorithm that learns on-line show the scale of changes introduced in manipulating signal U by learning procedure. They are acceptable from practical viewpoint because they provide overregulation and settling time similar to Q-learning (PI).



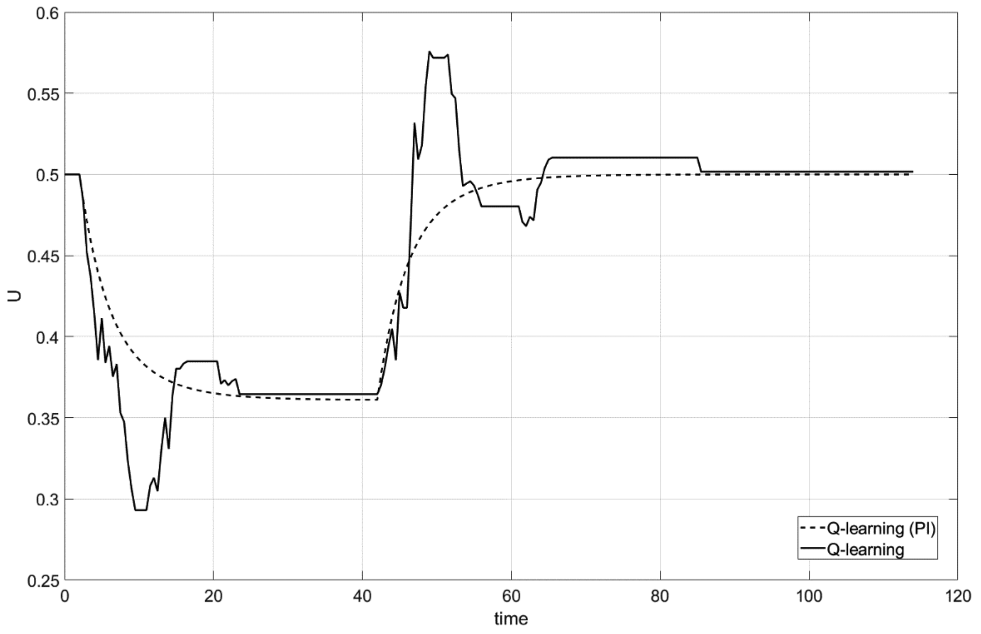
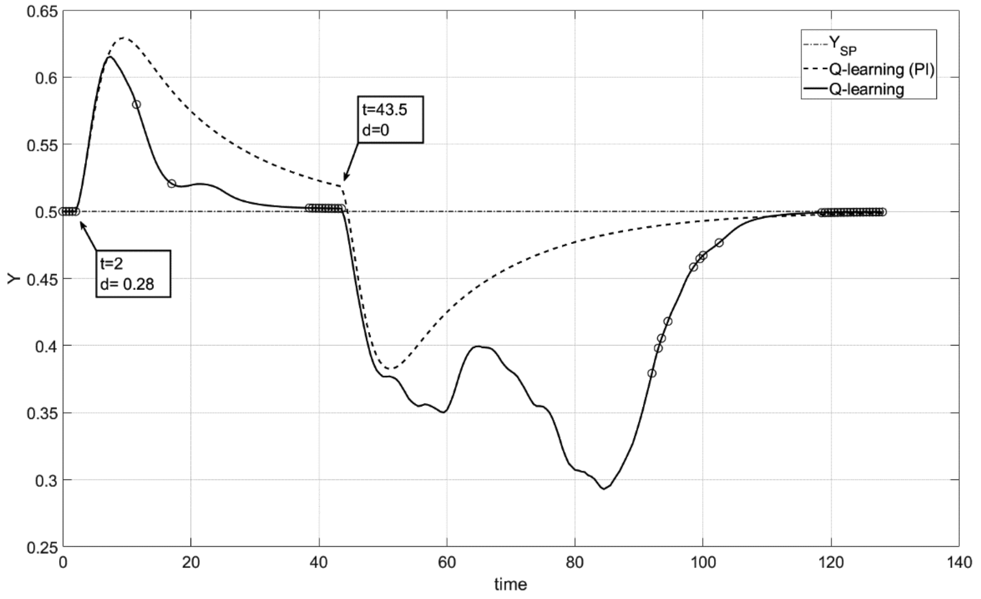


Figure 10: An example of two consecutive learning epochs for on-line learning of Q-learning algorithm from load disturbance step changes for FO model. Upper diagram shows process output Y and lower diagram shows manipulating variable U. Dots indicate moments of rewarding for preserving of the reference trajectory. Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings



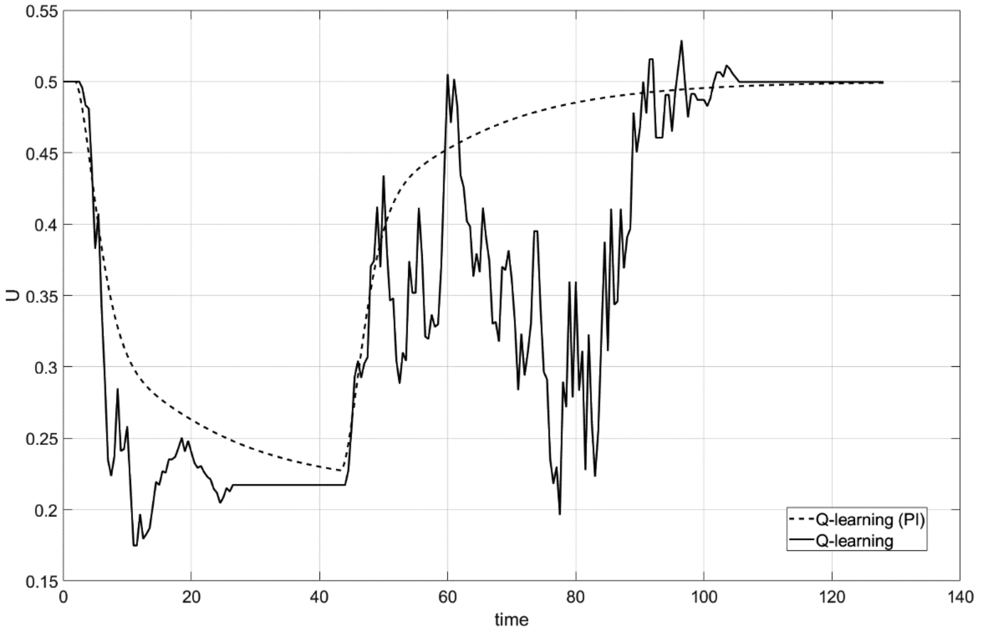
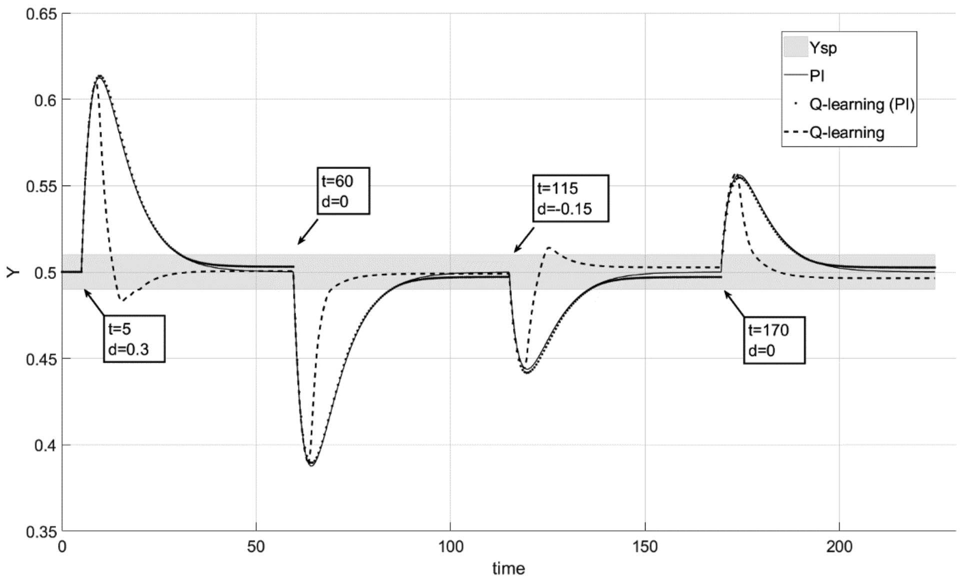


Figure 11: An example of two consecutive learning epochs for on-line learning of Q-learning algorithm from load disturbance step changes for oscillatory SO model. Upper diagram shows process output Y and lower diagram shows manipulating variable U. Dots indicate moments of rewarding for preserving of the reference trajectory. Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings

Improvement in control performance after such learning procedure is presented for both models in Figs. 12 and 13, for two different numbers of learning epochs. These results show that control performance improves with successive



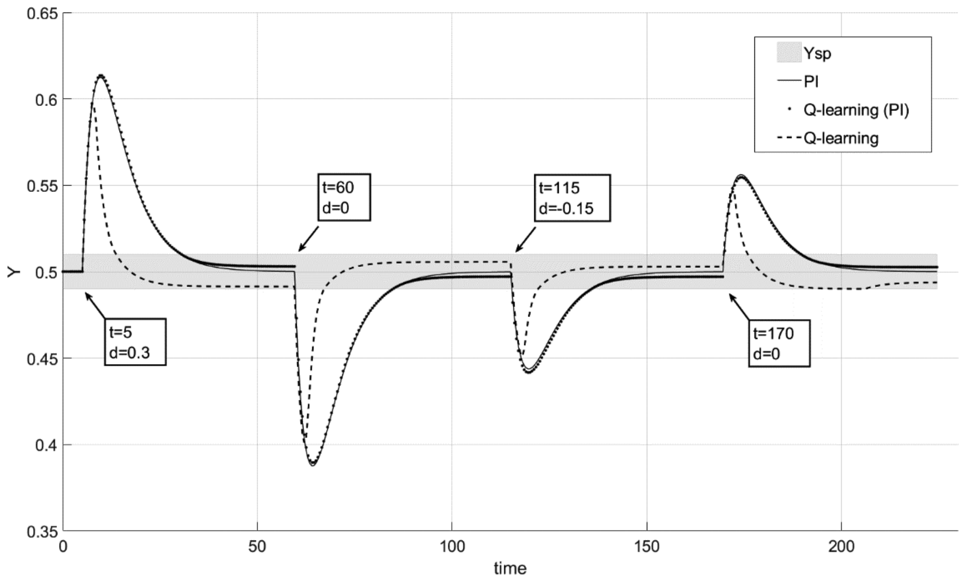


Figure 12: Disturbance rejection for Q-learning algorithm operating FO model for online learning from consecutive load disturbance step changes, after 1000 learning epochs (upper diagram) and 2000 learning epochs (lower diagram). Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings

learning epochs and that on-line learning is possible only from process disturbances, without any preliminary learning.

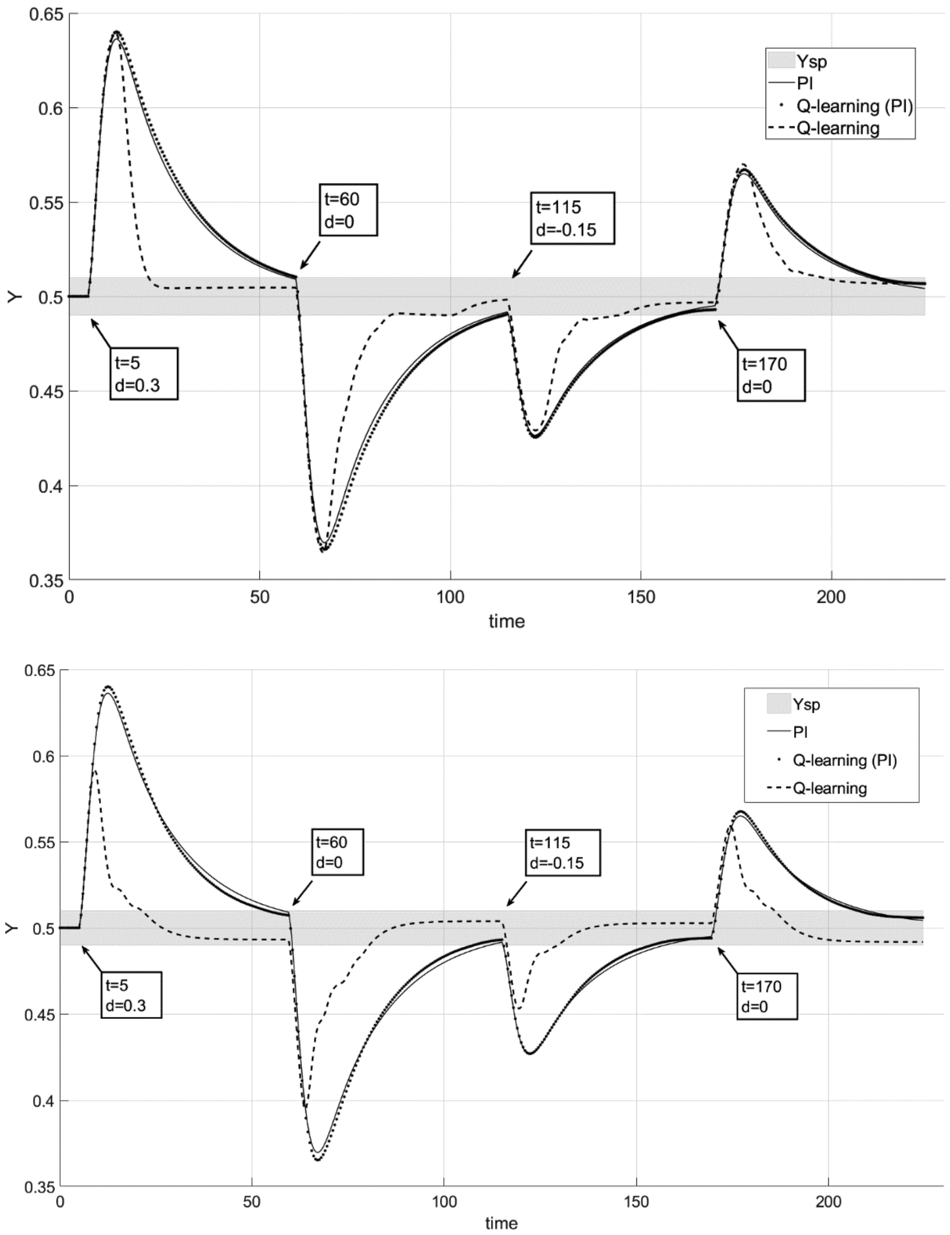


Figure 13: Disturbance rejection for Q-learning algorithm operating FO model for on-line learning from consecutive load disturbance step changes, after 2500 learning epochs (upper diagram) and 5000 learning epochs (lower diagram). Q-learning (PI) represents Q-learning algorithm initialized based on PI tunings

5. Conclusions

In this paper, it was shown that Q-learning algorithm can be potentially used as a self-improving controller to be applied in industrial control loops and it can combine normal process operating with on-line learning. It was suggested to define the *goal state* as an exponential closed loop reference trajectory, which allows for design the closed loop dynamics of the control system with Q-learning algorithm. The method to ensure bumpless switching between existing (poorly tuned) PI controller and the suggested Q-learning algorithm was proposed to use preliminary initialization of Q-matrix. This method ensures that after switching, Q-learning algorithm provides the same control performance as PI controller. Then, it can only improve this performance by on-line learning. This self-improvement ability was tested by simulation in the application to first order dynamical model and more challenging oscillatory second-order model.

Results presented in this paper were limited to linear time-invariant processes. Extension to nonlinear processes is straightforward but in cases of time-varying processes, applicability of Q-learning algorithm is limited due to relatively long time required for effective on-line learning stage. If process dynamics fluctuate faster, Q-learning algorithm is not able to follow this fluctuations. Additionally, there are still many issues that require further research to make this concept a realistic alternative for PID controllers operating in majority of control loops. Especially, aspects of practical implementation in PLC-based platforms should be investigated. Dynamics of learning process also requires further research because it limits applicability of Q-learning algorithm as an general-purpose industrial adaptive controller. Finally, more deterministic rules should be suggested for adjusting parameters of Q-learning algorithm (Q-learning tuning rules).

References

[1] H. Boubertakh, S. Labiod, M. Tadjine and P.Y. Glorennec: Optimization of fuzzy PID controllers using Q-learning algorithm. *Archives of Control Sciences*, **18**(4), (2008), 415–435

[2] I. Carlucho, M. De Paula, S.A. Villar and G.G. Acosta: Incremental Q-learning strategy for adaptive PID control of mobile robots. *Expert Systems With Applications*, **80**, (2017), 183–199, DOI: 10.1016/j.eswa.2017.03.002.

[3] K. Delchev: Simulation-based design of monotonically convergent iterative learning control for nonlinear systems. *Archives of Control Sciences*, **22**(4), (2012), 467–480.

[4] M. Jelali: An overview of control performance assessment technology and industrial applications. *Control Eng. Pract.*, **14**(5), (2006), 441–466, DOI: 10.1016/j.conengprac.2005.11.005.

[5] M. Jelali: *Control Performance Management in Industrial Automation: Assessment, Diagnosis and Improvement of Control Loop Performance*. Springer-Verlag London, (2013)

[6] H.-K. Lam, Q. Shi, B. Xiao, and S.-H. Tsai: Adaptive PID Controller Based on Q-learning Algorithm. *CAAI Transactions on Intelligence Technology*, **3**(4), (2018), 235–244, DOI: 10.1049/trit.2018.1007.

[7] D. Li, L. Qian, Q. Jin, and T. Tan: Reinforcement learning control with adaptive gain for a Saccharomyces cerevisiae fermentation process. *Applied Soft Computing*, **11**, (2011), 4488–4495, DOI: 10.1016/j.asoc.2011.08.022.

[8] M.M. Noel and B.J. Pandian: Control of a nonlinear liquid level system using a new artificial neural network based reinforcement learning approach. *Applied Soft Computing*, **23**, (2014), 444–451, DOI: 10.1016/j.asoc.2014.06.037.

[9] T. Praczyk: Concepts of learning in assembler encoding. *Archives of Control Sciences*, **18**(3), (2008), 323–337.

[10] M.B. Radac and R.E. Precup: Data-driven model-free slip control of anti-lock braking systems using reinforcement Q-learning. *Neurocomputing*, **275**, (2017), 317–327, DOI: 10.1016/j.neucom.2017.08.036.

[11] A.K. Sadhu and A. Konar: Improving the speed of convergence of multi-agent Q-learning for cooperative task-planning by a robot-team. *Robotics and Autonomous Systems*, **92**, (2017), 66–80, DOI: 10.1016/j.robot.2017.03.003.

[12] N. Sahebjamnia, R. Tavakkoli-Moghaddam, and N. Ghorbani: Designing a fuzzy Q-learning multi-agent quality control system for a continuous chemical production line – A case study. *Computers & Industrial Engineering*, **93**, (2016), 215–226, DOI: 10.1016/j.cie.2016.01.004.

[13] K. Stebel: Practical aspects for the model-free learning control initialization. in *Proc. of 2015 20th International Conference on Methods and Models in Automation and Robotics* (MMAR), Poland, (2015), DOI: 10.1109/MMAR.2015.7283918.

[14] R.S. Sutton and A.G. Barto: *Reinforcement learning: An Introduction*, MIT Press, (1998)

[15] S. Syafiie, F. Tadeo, and E. Martinez: Softmax and ε-greedy policies applied to process control. *IFAC Proceedings*, **37**, (2004), 729–734, DOI: 10.1016/S1474-6670(16)31556-2.

[16] S. Syafiie, F. Tadeo, and E. Martinez: Model-free learning control of neutralization process using reinforcement learning. *Engineering Applications of Artificial Intelligence*, **20**, (2007), 767–782, DOI: 10.1016/j.engappai.2006.10.009.

[17] S. Syafiie, F. Tadeo, and E. Martinez: Learning to control pH processes at multiple time scales: performance assessment in a laboratory plant. *Chemical Product and Process Modeling*, **2**(1), (2007), DOI: 10.2202/1934-2659.1024.

[18] S. Syafiie, F. Tadeo, E. Martinez, and T. Alvarez: Model-free control based on reinforcement learning for a wastewater treatment problem. *Applied Soft Computing*, **11**, (2011), 73–82, DOI: 10.1016/j.asoc.2009.10.018.

[19] P. Van Overschee and B. De Moor: RAPID: The End of Heuristic PID Tuning. *IFAC Proceedings*, **33**(4), (2000), 595–600, DOI: 10.1016/S1474-6670(16)38308-8.

[20] M. Wang, G. Bian, and H. Li: A new fuzzy iterative learning control algorithm for single joint manipulator. *Archives of Control Sciences*, **26**(3), (2016), 297–310. DOI: 10.1515/acsc-2016-0017.

[21] Ch.J.C.H. Watkins and P. Dayan: Technical Note: Q-learning. *Machine Learning*, **8**, (1992), 279–292, DOI: 10.1023/A:1022676722315.