Model-Free Q-Learning Controller with Projection Function and Dead Time Compensation for Industrial Process Control

Jakub Musiał, Krzysztof Stebel, and Jacek Czeczot, *Member, IEEE*

***Abstract*—The performance of a significant majority of industrial proportional-integral-derivative (PID) controllers remains suboptimal due to inadequate tuning. Manual retuning requires expert knowledge and experimental data, making it impractical for large-scale industrial facilities operating hundreds of control loops simultaneously. This challenge is further compounded when processes exhibit significant dead time, which degrades closed-loop performance and complicates traditional compensation methods such as Smith Predictor or Internal Model Control, both of which require accurate process models.**

This paper presents a model-free Q-learning controller with two key extensions that address these industrial challenges. First, a projection function enables operation with desired time constants differing from the baseline PI tuning, compensating for the mismatch through an additional control term derived from first-order trajectory requirements. Second, a delayed credit assignment mechanism handles dead time compensation by decoupling the physical plant delay from the controller's compensation strategy, implemented through FIFO buffers that defer Q-value updates until action consequences become observable.

The proposed Q2d controller bumplessly replaces existing PI controllers, starting with equivalent initial performance and gradually improving online through reinforcement learning without requiring a process model or offline training. Validation on first- and second-order inertia processes with dead times up to 4 seconds demonstrates 12–22% improvement in integral absolute error compared to PI baseline. Notably, the method exhibits graceful degradation: even 50% undercompensation of dead time (setting $T_{0,\text{controller}} = T_0/2$) provides substantially better performance than no compensation, offering robustness to uncertain dead time estimates common in industrial practice.

***Index Terms*—Q-learning, dead time compensation, projection function, industrial control, reinforcement learning, bumpless switching, model-free control, delayed credit assignment**

I. Introduction

A. Industrial Motivation

Proportional-integral-derivative (PID) controllers remain the dominant control solution in industrial process automation, with estimates suggesting their deployment in over 90% of control loops across chemical plants, refineries, manufacturing facilities, and other process industries [**?**], [**?**]. Despite this ubiquity, numerous studies indicate that approximately 60% of industrial PID loops perform poorly due to inadequate tuning [**?**], [**?**]. The consequences include increased energy consumption, reduced product quality, higher raw material waste, and suboptimal process economics.

Manual retuning of PID controllers requires expert knowledge of control theory, detailed understanding of process dynamics, and access to experimental data for parameter identification. In large industrial facilities operating hundreds or thousands of simultaneous control loops, systematic retuning becomes impractical due to economic constraints, operational risks associated with experimentation, and limited availability of qualified personnel. Existing automated tuning solutions, such as relay-based autotuning [**?**] or model-based optimization [**?**], either require process disruption for identification experiments or depend on accurate process models that are difficult to obtain and maintain for complex, time-varying industrial systems.

The challenge is further compounded when processes exhibit significant dead time (transport delay), which is ubiquitous in industrial applications due to material transport through pipes, measurement delays, actuator dynamics, and communication latencies. Dead time severely degrades closed-loop performance and stability margins, necessitating specialized compensation techniques. Traditional approaches such as the Smith Predictor [**?**] and Internal Model Control (IMC) [**?**] require accurate process models including precise dead time estimation. Model mismatch, particularly in dead time, can lead to poor performance or instability, limiting the practical applicability of these methods in industrial environments where process parameters drift over time and exact models are rarely available.

These industrial realities motivate the search for self-improving controllers that can be deployed without process models, require minimal commissioning effort, maintain acceptable performance during the learning phase, and exhibit robustness to parameter uncertainty including dead time estimation errors.

B. Q-Learning for Process Control

Reinforcement learning (RL), and Q-learning in particular, offers a promising framework for developing controllers that improve autonomously through interaction with the process [**?**], [**?**]. Unlike supervised learning approaches that require labeled training data, or model-based control methods that require process identification, Q-learning discovers optimal control policies through trial-and-error interaction guided by a scalar reward signal. The fundamental concept involves

J. Musiał, K. Stebel, and J. Czeczot are with the Department of Automatic Control and Robotics, Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology, 44-100 Gliwice, Poland (e-mail: jakub.musial@polsl.pl; krzysztof.stebel@polsl.pl; jacek.czeczot@polsl.pl).

Manuscript received Month DD, 2025; revised Month DD, 2025.

learning a Q-function $Q(s, a)$ that estimates the expected cumulative discounted reward for taking action a in state s and following an optimal policy thereafter.

Recent years have witnessed growing interest in applying Q-learning to process control problems. Applications include pH neutralization [**?**], batch reactor control [**?**], temperature regulation [**?**], and multi-input multi-output systems [**?**]. However, most existing Q-learning controllers share common limitations that hinder industrial deployment: (1) they typically require extensive offline training in simulation before deployment, (2) initial performance can be poor until sufficient learning occurs, (3) exploration during learning can disturb normal process operation, and (4) dead time compensation has received limited attention in the Q-learning literature.

A critical requirement for industrial acceptance is *bumpless switching*—the controller must exhibit predictable, acceptable performance from the moment of deployment. Our previous work addressed this challenge by initializing the Q-learning controller from existing PID tunings [**?**], [**?**]. Specifically, we developed a two-dimensional Q-learning approach (Q2d) that merges error and error derivative into a single state variable based on first-order closed-loop trajectory requirements: $s = \dot{e}+(1/T_e)\cdot e$, where T_e is the desired closed-loop time constant. By setting T_e equal to the integral time T_I of the existing PI controller and initializing the Q-matrix as an identity matrix, the Q2d controller starts with PI-equivalent performance and gradually improves through online learning.

The Q2d approach offers significant advantages over the earlier three-dimensional formulation (Q3d) [**?**]: the Q-matrix size reduces from N^3 to N^2 elements, learning acceleration improves by approximately threefold, and convergence becomes more consistent. Validation on second-order processes and experimental verification on an asynchronous motor demonstrated successful bumpless initialization and gradual performance improvement without offline training [**?**].

C. Research Gap and Contributions

Despite the progress achieved with the Q2d approach, two important challenges remain unresolved for industrial deployment:

Challenge 1: Time Constant Mismatch. When the desired closed-loop time constant T_e differs significantly from the baseline integral time T_I (e.g., when faster response is required: $T_e = 2$ s vs. $T_I = 20$ s), the initial performance may deviate from the PI baseline. Large mismatches require either gradual adaptation or direct compensation. While our subsequent work addresses this through staged learning with gradual T_e reduction [**?**], the present paper investigates an alternative approach using a projection function that directly compensates for the mismatch.

Challenge 2: Dead Time Compensation. Standard Q-learning assumes that action consequences are observable in the next time step, which fails when significant dead time exists between control action and measured output. The credit assignment problem—determining which past action caused the currently observed state—becomes critical for learning convergence. Existing Q-learning literature provides limited guidance for systematic dead time compensation in continuous control applications.

This paper addresses both challenges through extensions to the Q2d framework:

1) **Projection Function for Time Constant Mismatch:** We derive a compensation term $\Delta u_{\text{proj}} = -e \cdot (1/T_e - 1/T_I)$ that enables direct operation with $T_e \neq T_I$ from the start of learning. The projection function maintains the trajectory-following interpretation while allowing flexibility in the desired time constant.
2) **Delayed Credit Assignment for Dead Time:** We introduce decoupled dead time parameters—T_0 representing the physical plant delay and $T_{0,\text{controller}}$ representing the compensation strategy. State-action pairs are buffered in FIFO structures with size $T_{0,\text{controller}}/\Delta t$, deferring Q-value updates until action consequences become observable. This enables systematic study of matched compensation ($T_{0,\text{controller}} = T_0$), undercompensation ($T_{0,\text{controller}} < T_0$), and no compensation ($T_{0,\text{controller}} = 0$).
3) **Robustness Validation:** We demonstrate that the method exhibits graceful degradation—even 50% undercompensation ($T_{0,\text{controller}} = T_0/2$) provides substantially better performance than no compensation. This robustness to partial dead time knowledge is particularly valuable in industrial practice where exact dead time estimates are difficult to obtain.

The proposed extensions maintain the core advantages of Q2d: model-free operation, bumpless switching, and online learning without performance degradation. Validation on first- and second-order inertia processes with dead times up to 4 seconds demonstrates 12–22% improvement in integral absolute error (IAE) compared to PI baseline, with dead time compensation providing 2–3$\times$ better improvement than operation without compensation.

The remainder of this paper is organized as follows. Section **??** provides background on Q-learning and formulates the control objective based on first-order trajectory requirements. Section **??** details the Q2d controller with projection function, including state/action generation and the complete control algorithm. Section **??** presents the delayed credit assignment mechanism for dead time compensation. Section **??** provides validation results on two plant models with multiple dead time scenarios. Section **??** discusses practical considerations and limitations. Section **??** concludes with directions for future work.

II. Problem Statement

A. Q-Learning Fundamentals

Q-learning is a model-free reinforcement learning algorithm that learns optimal control policies through interaction with an environment [**?**], [**?**]. The agent observes the current state $s \in \mathcal{S}$, selects an action $a \in \mathcal{A}$, receives a scalar reward $R \in \mathbb{R}$, and transitions to a new state $s' \in \mathcal{S}$. The objective

is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative discounted reward:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor that determines the relative importance of immediate versus future rewards.

The Q-function $Q(s, a)$ represents the expected cumulative reward for taking action a in state s and following an optimal policy thereafter:

$$Q^*(s, a) = \mathbb{E}\left[R + \gamma \max_{a'} Q^*(s', a') \mid s, a\right] \tag{2}$$

The Q-learning algorithm iteratively updates Q-value estimates based on observed transitions using the temporal difference (TD) error:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[R + \gamma \max_{a'} Q(s', a') - Q(s, a)\right] \tag{3}$$

where $\alpha \in (0, 1]$ is the learning rate. Under appropriate conditions (all state-action pairs visited infinitely often, learning rate decay), Q-learning converges to the optimal Q-function Q^* [**?**].

Action selection balances exploration (trying new actions to discover potentially better policies) and exploitation (selecting actions with highest current Q-values). The ε-greedy strategy provides a simple yet effective approach:

$$a = \begin{cases} \arg\max_{a'} Q(s, a') & \text{with probability } 1 - \varepsilon \\ \text{random action} & \text{with probability } \varepsilon \end{cases} \tag{4}$$

where $\varepsilon \in [0, 1]$ controls the exploration rate.

B. Control Objective and Target Trajectory

Consider a single-input single-output (SISO) dynamical process with control signal $u(t)$, process output $y(t)$, setpoint $y_{\text{sp}}(t)$, and control error $e(t) = y_{\text{sp}}(t) - y(t)$. The control objective is to drive the error to zero while satisfying constraints on control effort and avoiding excessive oscillations.

We define the desired closed-loop behavior through a first-order trajectory requirement:

$$\dot{e}(t) + \frac{1}{T_e} e(t) = 0 \tag{5}$$

where $T_e > 0$ is the desired closed-loop time constant. The analytical solution to (**??**) is:

$$e(t) = e_0 \exp\left(-\frac{t}{T_e}\right) \tag{6}$$

which represents exponential decay of the error with time constant T_e. Smaller T_e values correspond to faster response but may require more aggressive control action.

This trajectory formulation provides an intuitive link to existing PI controller tunings. A standard PI controller in velocity form can be expressed as:

$$\Delta u_{\text{PI}}(t) = K_P \Delta t \left[\dot{e}(t) + \frac{1}{T_I} e(t)\right] \tag{7}$$

where K_P is the proportional gain, T_I is the integral time, and Δt is the sampling time. Comparing (**??**) with (**??**) reveals the structural similarity: the PI controller attempts to drive the error along a trajectory defined by T_I.

For bumpless switching from an existing PI controller to Q-learning, we can initialize with $T_e = T_I$ and controller gain $K_Q = K_P$, ensuring equivalent initial behavior. However, when improved performance is desired (e.g., faster response with $T_e < T_I$), the mismatch between T_e and T_I must be compensated. This motivates the projection function approach developed in Section **??**.

C. Dead Time Challenge

Many industrial processes exhibit significant dead time (also called transport delay or time delay) between control action and measured output:

$$y(t) = G(s) \cdot e^{-T_0 s} \cdot u(t) \tag{8}$$

where $G(s)$ is the process transfer function without delay and $T_0 \geq 0$ is the dead time. Physical sources of dead time include material transport through pipes, distance-velocity lags in continuous processes, measurement sensor delays, actuator response times, and communication latencies in distributed control systems.

Dead time poses a fundamental challenge for Q-learning: the standard update rule (**??**) assumes that the consequence of action $a(t)$ is observable in the next state $s(t + \Delta t)$. When dead time T_0 spans multiple sampling intervals ($T_0 \gg \Delta t$), the control action $u(t)$ only affects the measured output at time $t + T_0$. This creates a *credit assignment problem*—when the controller observes state $s(t)$ at time t, which past action is responsible?

Without proper credit assignment, Q-learning may update the wrong state-action pairs or converge slowly. For example, if $T_0 = 4$ s and $\Delta t = 0.1$ s, the action taken 40 time steps ago influences the current state, but naive application of (**??**) would credit the most recent action. Section **??** addresses this through delayed credit assignment using FIFO buffers that defer Q-value updates until action consequences become observable.

III. Q2d Controller with Projection Function

A. State Space Merging

Traditional Q-learning formulations for control applications discretize error e and error derivative $\dot{e}$ independently, creating a two-dimensional state space $(e, \dot{e})$ and a three-dimensional Q-matrix when considering actions [**?**]. For example, with N discrete values for each dimension, the state space contains $N \times N$ states and the Q-matrix requires $N \times N \times N = N^3$ elements. This cubic growth in memory becomes prohibitive for fine discretization or resource-constrained implementations such as programmable logic controllers (PLCs).

The Q2d approach overcomes this limitation by merging error and error derivative into a single state variable based on the target trajectory (**??**):

$$s = \dot{e} + \frac{1}{T_e} e \tag{9}$$

This merged state has a clear physical interpretation: s measures the deviation from the desired first-order trajectory.

When $s = 0$, the error evolves according to $\dot{e} = -(1/T_e)e$, meaning the system follows the target trajectory exactly. Positive values $s > 0$ indicate the error is decreasing slower than desired or increasing, while negative values $s < 0$ indicate faster-than-desired error reduction.

The dimensionality reduction from $(e, \dot{e})$ to s reduces the Q-matrix from N^3 to N^2 elements, providing substantial memory savings. For example, with $N = 100$, the reduction is from 10^6 to 10^4 elements—a 100-fold decrease. Beyond memory efficiency, our previous work demonstrated that Q2d exhibits faster learning convergence and more consistent behavior than Q3d [**?**].

The merged state naturally accommodates both positive and negative values. We denote the discrete state space as:

$$\mathcal{S} = \{s_1, s_2, \ldots, s_{N_s}\} \tag{10}$$

where N_s is the total number of states, typically an odd number (e.g., $N_s = 2N - 1$) to include a goal state at $s = 0$ and symmetric positive/negative states.

B. Projection Function Derivation

When the desired time constant T_e differs from the baseline integral time T_I (e.g., $T_e = 10$ s vs. $T_I = 20$ s), direct application of the merged state (**??**) with the PI control structure (**??**) creates a mismatch. The PI controller was tuned for trajectory dynamics characterized by T_I, but we desire dynamics characterized by T_e. This section derives a projection function that compensates for this mismatch.

1) Mathematical Basis: Consider the desired trajectory requirement (**??**) with time constant T_e:

$$\dot{e} + \frac{1}{T_e}e = 0 \tag{11}$$

The PI controller structure (**??**), however, naturally implements a trajectory with time constant T_I:

$$\dot{e} + \frac{1}{T_I}e = 0 \tag{12}$$

The difference between these trajectories is:

$$\left(\dot{e} + \frac{1}{T_e}e\right) - \left(\dot{e} + \frac{1}{T_I}e\right) = \left(\frac{1}{T_e} - \frac{1}{T_I}\right)e \tag{13}$$

This mismatch must be compensated in the control law to achieve the desired T_e dynamics.

2) Projection Term: To compensate for the trajectory mismatch, we introduce an additional control term proportional to the error and the difference in time constants:

$$\Delta u_{\text{proj}} = -e \cdot \left(\frac{1}{T_e} - \frac{1}{T_I}\right) \tag{14}$$

The negative sign ensures the compensation acts in the correct direction. When $T_e < T_I$ (faster response desired), the term $(1/T_e - 1/T_I) > 0$, and the projection adds a negative correction for positive errors, accelerating error reduction. Conversely, when $T_e > T_I$ (slower response), the projection becomes positive for positive errors, moderating the control action.

The physical interpretation is straightforward: the projection function compensates for the difference between the baseline PI trajectory (T_I) and the desired Q-learning trajectory (T_e). When $T_e = T_I$, the projection term vanishes, and standard Q2d control applies. The magnitude of compensation is proportional to both the current error (larger errors require larger correction) and the relative mismatch between time constants.

3) Complete Control Law: The complete Q2d control law with projection function becomes:

$$\begin{aligned} u(t) &= u(t - \Delta t) + K_Q \cdot a(s) \cdot \Delta t - e(t) \cdot \left(\frac{1}{T_e} - \frac{1}{T_I}\right) \\ &= u(t - \Delta t) + K_Q \cdot a(s) \cdot \Delta t + \Delta u_{\text{proj}}(t) \end{aligned} \tag{15}$$

where:

- $u(t)$ is the control signal at time t
- K_Q is the controller gain (typically set to K_P for bumpless switching)
- $a(s)$ is the action selected for the current merged state s
- Δt is the sampling time
- $e(t)$ is the control error
- $\Delta u_{\text{proj}}(t)$ is the projection compensation (**??**)

The control signal is saturated to respect actuator constraints:

$$u(t) = \text{sat}_{[u_{\min}, u_{\max}]}(u(t)) \tag{16}$$

where typical values are $u_{\min} = 0\%$ and $u_{\max} = 100\%$ for industrial applications.

C. State and Action Generation

The discretization of the continuous merged state space (**??**) into a finite set of states (**??**) critically influences controller performance. We employ an exponentially-distributed discretization that provides dense sampling near the goal state ($s = 0$) where precise control is required, and coarser sampling for larger deviations where aggressive corrective action suffices.

1) Generation Procedure: The state and action generation follows Algorithm **??**:

Parameters:

- τ: Exponential decay time constant for trajectory generation (independent of T_e)
- $e_{\max}$: Maximum expected error (determines range coverage)
- $\varepsilon_{\text{prec}}$: Precision parameter defining steady-state accuracy
- N_{expected}: Expected number of states (guides τ selection)
- $\varepsilon_{\text{merge}}$: Merging threshold for close boundaries

The parameter τ provides flexibility independent of the control time constant T_e. Smaller τ values create more densely distributed states near zero, improving steady-state accuracy at the cost of larger Q-matrices. The precision parameter $\varepsilon_{\text{prec}}$ directly determines the steady-state error tolerance—typical values range from 0.1 to 1.0 for percentage-based error representation.

2) Scaling with Time Constant: A critical property of the merged state (**??**) is that the steady-state error tolerance remains invariant to T_e changes. In steady state, $\dot{e} = 0$, so:

$$s_{\text{ss}} = \frac{1}{T_e}e_{\text{ss}} \tag{17}$$

Algorithm 1 State and Action Space Generation

Require: T_e, τ, $e_{\max}$, $\varepsilon_{\text{prec}}$, N_{expected}
Ensure: State boundaries $\mathcal{S}_b$, state means $\mathcal{S}_m$, actions $\mathcal{A}$

$i \leftarrow 0$, $t \leftarrow 0$
while $e(t) > \varepsilon_{\text{prec}}$ **and** $i < N_{\max}$ **do**
  $e(i) \leftarrow e_{\max} \cdot \exp(-t/\tau)$ {Exponential decay}
  $\dot{e}(i) \leftarrow (e(i+1) - e(i))/\Delta t$ {Numerical derivative}
  $t \leftarrow t + \Delta t$
  $i \leftarrow i + 1$
end while
$N_{\text{points}} \leftarrow i$

for $i = 1$ to N_{points} **do**
  for $j = 1$ to N_{points} **do**
    $s_k \leftarrow \dot{e}(j) + (1/T_e) \cdot e(i)$ {Compute state values}
    $k \leftarrow k + 1$
  end for
end for

Sort all s_k values in ascending order
Merge adjacent values if $|s_{k+1} - s_k| < \varepsilon_{\text{merge}}$
Resulting positive states: $\{s_1^+, s_2^+, \ldots, s_N^+\}$

Extend symmetrically: $\mathcal{S}_b \leftarrow \{-s_N^+, \ldots, -s_1^+, 0, s_1^+, \ldots, s_N^+\}$

for each adjacent pair (s_i, s_{i+1}) in $\mathcal{S}_b$ **do**
  $s_{\text{mean},i} \leftarrow (s_i + s_{i+1})/2$ {State representative value}
end for

$\mathcal{A} \leftarrow \mathcal{S}_m$ {Actions equal state means}
return $\mathcal{S}_b$, $\mathcal{S}_m$, $\mathcal{A}$

If the goal state spans $s \in (-s_{\text{goal}}, s_{\text{goal}}]$, the corresponding steady-state error tolerance is:

$$-T_e \cdot s_{\text{goal}} < e_{\text{ss}} \leq T_e \cdot s_{\text{goal}} \tag{18}$$

By setting $s_{\text{goal}} = \varepsilon_{\text{prec}}/T_e$, the error tolerance becomes:

$$-\varepsilon_{\text{prec}} < e_{\text{ss}} \leq \varepsilon_{\text{prec}} \tag{19}$$

which is independent of T_e. This ensures consistent steady-state accuracy regardless of the chosen time constant.

3) Q-Matrix Initialization: The Q-matrix is initialized as an identity matrix:

$$Q_{ij}(0) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

where i indexes states and j indexes actions. This initialization ensures that each state initially selects the action corresponding to its own mean value, mimicking the PI control structure. The diagonal initialization provides stable starting behavior while allowing the learning process to discover improved policies.

D. Control Algorithm Implementation

The complete Q2d control algorithm with projection function is presented in Algorithm **??**.

1) Key Implementation Details: **Manual Control Phase:** The first few sampling intervals (typically 5) use manual control $u = y_{\text{sp}}/k$ to initialize the system toward steady state and pre-fill delay buffers, where k is the process steady-state gain.

Goal State Enforcement: When the system reaches the goal state ($s \approx 0$), the controller always selects the goal action ($a = 0$, representing zero control increment) to maintain steady state. Learning is disabled in the goal state since the optimal action is already known.

Saturation Handling: When the control signal saturates, Q-learning updates are disabled ($\varepsilon_{\text{learning}} = 0$) to prevent learning from unrealizable actions. This anti-windup mechanism prevents the Q-matrix from associating rewards with saturated control values.

Exploration Strategy: The exploration function ExploreAction$(i_s, \mathcal{A})$ selects random actions within a constrained range around the current best action, bounded by a parameter RD (random deviation). Additionally, directional constraints ensure that states above the goal select actions above the goal action, and vice versa, maintaining consistent trajectory-following behavior.

The Q-learning update mechanism, including dead time compensation, is detailed in Section **??**.

IV. Dead Time Compensation via Delayed Credit Assignment

A. Decoupled Dead Time Parameters

Industrial processes often exhibit dead time that may be imperfectly known, time-varying, or difficult to measure precisely. To enable systematic investigation of dead time compensation strategies and robustness to estimation errors, we introduce two independent dead time parameters:

- T_0 **(Plant Dead Time):** The physical dead time in the actual process, representing reality. The control signal $u(t)$ affects the measured process output at time $t + T_0$. This parameter is inherent to the process and cannot be changed by the controller.
- $T_{0,\textbf{controller}}$ **(Controller Compensation Dead Time):** The dead time value used by the controller for credit assignment. This represents the controller's assumption or estimate of the dead time, which may differ from the true value T_0.

This decoupling enables investigation of several practically relevant scenarios:

Matched Compensation ($T_{0,\text{controller}} = T_0$): The controller has perfect knowledge of the dead time and compensates optimally.

No Compensation ($T_{0,\text{controller}} = 0$): The controller ignores dead time entirely, relying on implicit learning to handle the delay. This represents the naive Q-learning approach.

Undercompensation ($T_{0,\text{controller}} < T_0$): The controller underestimates the dead time, crediting actions too early. This tests robustness to incomplete information.

Overcompensation ($T_{0,\text{controller}} > T_0$): The controller overestimates the dead time, crediting actions too late. This tests robustness to overly conservative estimates.

The ability to independently vary these parameters provides insight into the method's robustness—a critical consideration for industrial deployment where exact dead time knowledge is rarely available and may drift over time due to process changes, varying operating conditions, or equipment aging.

B. Delayed Credit Assignment Algorithm

The core challenge in Q-learning with dead time is ensuring that Q-value updates credit the correct state-action pairs. When dead time spans multiple sampling intervals, the effect of action $a(t)$ taken at time t becomes observable only at time $t + T_0$, not at the next sampling instant $t + \Delta t$.

1) Standard Q-Learning Failure: Standard Q-learning (Section **??**) assumes single-step transitions:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{21}$$

where s_t is the state at time t, a_t is the action taken, and s_{t+1} is the state observed at $t + \Delta t$. This implicitly assumes that s_{t+1} reflects the consequence of a_t.

When dead time exists, this assumption breaks. For example, with $T_0 = 4$ s and $\Delta t = 0.1$ s, the plant output at time t reflects the control action applied 40 time steps earlier, not the action a_t just selected. Applying (**??**) would incorrectly associate s_{t+1} with a_t, leading to poor or failed learning.

2) Buffer-Based Solution: To resolve this credit assignment problem, we introduce FIFO (first-in-first-out) buffers that store state-action pairs and delay Q-value updates until the consequences become observable. The buffer size is determined by the controller's compensation strategy:

$$N_{\text{buffer}} = \left\lfloor \frac{T_{0,\text{controller}}}{\Delta t} \right\rfloor \tag{22}$$

where $\lfloor \cdot \rfloor$ denotes the floor function. For example, with $T_{0,\text{controller}} = 4$ s and $\Delta t = 0.1$ s, the buffer holds $N_{\text{buffer}} = 40$ elements.

Algorithm **??** presents the complete delayed credit assignment procedure.

3) Timing Analysis: To understand why delayed credit assignment works when $T_{0,\text{controller}} = T_0$, consider the timeline of events:

1) **Time** t**:** Controller observes state $s(t)$, selects action $a(t)$, computes control $u(t)$
2) **Time** t **(plant):** Control $u(t)$ enters the plant's dead time buffer of length $T_0/\Delta t$
3) **Time** t **(controller):** State-action pair $(s(t), a(t))$ enters controller's buffer of length $T_{0,\text{controller}}/\Delta t$
4) **Time** $t + T_0$**:** Plant output $y(t + T_0)$ reflects the effect of $u(t)$
5) **Time** $t + T_0$**:** State $s(t + T_0)$ computed from $y(t + T_0)$ reflects consequence of $a(t)$
6) **Time** $t + T_{0,\textbf{controller}}$**:** Controller buffer pops $(s(t), a(t))$ for Q-value update
7) **When** $T_{0,\textbf{controller}} = T_0$**:** The update at time $t + T_0$ pairs $a(t)$ with $s(t + T_0)$

This synchronization ensures correct credit assignment: the action $a(t)$ is updated based on the state $s(t+T_0)$ that actually reflects its consequence.

4) Buffer Pre-filling: During the manual control phase (first 5–10 samples), both plant and controller buffers are pre-filled with appropriate values:

- Plant buffers: Filled with $u = y_{\text{sp}}/k$ (steady-state control)
- Controller buffers: Filled with goal state and goal action

This initialization prevents transients caused by zero-filled buffers and simulates a system already operating near steady state before Q-learning begins.

C. Sparse Reward Strategy

The choice of reward function significantly influences Q-learning convergence and the resulting policy. We employ a sparse reward strategy that provides positive reward only at the goal state:

$$R(s, a) = \begin{cases} 1 & \text{if } s = s_{\text{goal}} \text{ and } a = a_{\text{goal}} \\ 0 & \text{otherwise} \end{cases} \tag{23}$$

where s_{goal} is the discrete state closest to $s = 0$ and a_{goal} is the zero control increment action.

1) Rationale: This sparse reward design ensures that:

1. Direct Reward at Goal: The state-action pair $(s_{\text{goal}}, a_{\text{goal}})$ receives immediate positive feedback. Over many learning iterations, this drives $Q(s_{\text{goal}}, a_{\text{goal}})$ toward high values.

2. Bootstrapped Learning Elsewhere: All other Q-values are learned through the bootstrap term $\gamma \max_{a'} Q(s', a')$ in (**??**). States that frequently transition to states with high Q-values will themselves acquire high Q-values, propagating the goal value backward through the state space.

3. Goal State Preference: Because only the goal state provides direct reward, the controller learns to drive the system toward $s = 0$ (the target trajectory) rather than lingering in off-trajectory states.

4. Zero Action Enforcement: By rewarding only when $a = a_{\text{goal}}$ (zero increment), we prevent the controller from selecting non-zero actions in the goal state, which would unnecessarily disturb steady-state operation.

Alternative reward schemes (e.g., $R = -|e|$ penalizing error magnitude) were considered but rejected because they can lead to undesirable behaviors such as premature action selection before reaching the goal state or difficulty distinguishing between trajectory-following and arbitrary error reduction paths.

2) Convergence Properties: Under the sparse reward strategy, the Q-value at the goal state becomes:

$$Q(s_{\text{goal}}, a_{\text{goal}}) \to \frac{1}{1 - \gamma} \tag{24}$$

as learning progresses and the goal state is visited repeatedly. For $\gamma = 0.99$, the theoretical limit is $Q(s_{\text{goal}}, a_{\text{goal}}) = 100$. In practice, convergence to values near 95–98 is typical after sufficient learning epochs.

States farther from the goal acquire lower Q-values based on the expected cumulative discounted reward for following an optimal trajectory toward the goal. This creates a gradient in the Q-matrix that guides the controller toward the target trajectory.

D. Continuous Learning Mode

An important design decision concerns buffer management between learning episodes. In our implementation, *episodes* are artificial boundaries used for convergence monitoring and performance evaluation—they do not represent system resets or reinitialization in a physical sense.

1) Buffer Persistence: Buffers are **not** reset between episodes:

- State buffer $\mathcal{B}_s$ retains values across episode boundaries
- Action buffer $\mathcal{B}_a$ retains values across episode boundaries
- Learning flag buffer $\mathcal{B}_L$ retains values across episode boundaries

This design choice reflects industrial reality: a controller deployed in a plant operates continuously, not in discrete episodes. Process disturbances occur at random times, and the controller must handle them regardless of when the previous disturbance ended. Episode boundaries are merely computational constructs for evaluating whether the controller has successfully rejected a disturbance and returned to steady state.

2) Episode Termination Criteria: An episode terminates when one of two conditions is met:

Stabilization: The system reaches and maintains the goal state ($s = s_{\text{goal}}$) for a specified number of consecutive samples (e.g., 20 samples). This indicates successful disturbance rejection.

Maximum Duration: The episode reaches a maximum iteration count (e.g., 4000 samples) without stabilizing. This timeout prevents indefinite episodes during early learning when the controller may not yet reliably reach the goal.

After episode termination, performance metrics (IAE, settling time, overshoot) are computed for that episode, a new disturbance or setpoint change is applied, and learning continues without buffer reset.

3) Verification Experiments: The only exception to continuous learning occurs during verification experiments conducted before and after training. For these controlled tests, buffers are explicitly reset to initial conditions to ensure fair comparison between PI baseline, Q-controller before learning, and Q-controller after learning. This reset provides a clean starting point for quantitative performance evaluation.

V. Validation and Results

A. Experimental Setup

The proposed Q2d controller with projection function and dead time compensation was validated through simulation studies on two benchmark process models representing common industrial dynamics. All experiments were conducted in MATLAB/Simulink environment with sampling time $\Delta t = 0.1$ s.

1) Plant Models: **Model 1 (First-Order Inertia):**

$$G_1(s) = \frac{k}{Ts+1} = \frac{1}{5s+1} \tag{25}$$

This represents simple thermal processes, flow control, or level control systems where dominant dynamics are characterized by a single time constant.

Model 3 (Second-Order Inertia):

$$G_3(s) = \frac{k}{(T_1 s+1)(T_2 s+1)} = \frac{1}{(5s+1)(2s+1)} \tag{26}$$

This represents cascaded thermal processes, heat exchangers, or systems with multiple energy storage elements. The different time constants ($T_1 = 5$ s, $T_2 = 2$ s) create more complex dynamics than Model 1.

Both models use unity steady-state gain ($k = 1$) for simplicity. Dead time is added externally as $e^{-T_0 s}$.

2) Dead Time Scenarios: Three dead time values were investigated:

- $T_0 = 0$ s: Baseline case without dead time
- $T_0 = 2$ s: Moderate dead time (20 sampling intervals)
- $T_0 = 4$ s: Significant dead time (40 sampling intervals)

For each $T_0 > 0$, three compensation strategies were tested:

- **No compensation**: $T_{0,\text{controller}} = 0$ (naive Q-learning)
- **Undercompensation**: $T_{0,\text{controller}} = T_0/2$ (50% estimate)
- **Matched compensation**: $T_{0,\text{controller}} = T_0$ (perfect knowledge)

This yields 7 experimental scenarios per model: 1 baseline ($T_0 = 0$) + 3 scenarios each for $T_0 \in \{2, 4\}$ s.

3) Controller Parameters: **PI Baseline:** Standard Siemens PLC default tunings [**?**]:

- Proportional gain: $K_P = 1$
- Integral time: $T_I = 20$ s
- Derivative time: $T_D = 0$ (PI control)

Q2d Parameters: Table **??** summarizes the Q-learning configuration.

TABLE I
Q2d Controller Parameters

Parameter	Symbol	Value
Controller gain	K_Q	1 (= K_P)
Goal time constant	T_e	10 s
Baseline integral time	T_I	20 s
Projection function	–	Enabled
Learning rate	α	0.1
Discount factor	γ	0.99
Exploration rate	ε	0.3
Random deviation	RD	3
Precision	$\varepsilon_{\text{prec}}$	0.5%
Expected states	N_{expected}	100
Training epochs	–	2500
Sampling time	Δt	0.1 s
Control limits	$[u_{\min}, u_{\max}]$	[0, 100]%

The choice $T_e = 10$ s represents a moderate $2\times$ speed-up compared to the baseline $T_I = 20$ s. This ratio provides sufficient performance improvement while maintaining manageable projection function magnitudes (Section **??**).

4) Learning Protocol: **Training Mode:** Disturbance-based learning ($uczenie\_obciazeniowe = 1$)

- Load disturbances randomly drawn from $d \sim \mathcal{N}(0, \sigma_d^2)$ with $\sigma_d = 0.5/3$ (3-sigma rule)
- Episode length randomized: $N_{\text{episode}} \sim \mathcal{N}(3000, 300^2)$, clipped to minimum 10 samples
- Episode terminates on stabilization (20 consecutive samples in goal state) or maximum 4000 samples
- Setpoint fixed at $y_{\text{sp}} = 50\%$ during training

Verification Mode: Clean experimental tests

- Three phases: setpoint tracking, load disturbance rejection, setpoint tracking

- Setpoint steps: $50\% \rightarrow 70\% \rightarrow 50\%$ (20% magnitude)
- Load disturbance: $d = \pm 0.3$ applied at specified times
- Total duration: 600 s
- Buffers reset to initial conditions
- Conducted before learning (Q-initial), after learning (Q-final), and with PI controller

Performance Metrics:

- **Integral Absolute Error (IAE)**: $\int_0^T |e(t)|\, dt$
- **Maximum Overshoot**: $\max_t |y(t) - y_{\text{sp}}|$ after setpoint step
- **Settling Time**: Time to reach $\pm 2\%$ of final value
- **Maximum Control Increment**: $\max_t |\Delta u(t)|$ (aggressiveness measure)

B. Dead Time Compensation Results

1) Q-Learning Convergence: Figure **??** shows the evolution of $Q(s_{\text{goal}}, a_{\text{goal}})$ over 2500 training epochs for Model 3 with $T_0 = 4$ s under three compensation strategies. The Q-value at the goal state serves as a convergence indicator—higher values indicate better learning.

Fig. 1. Q-learning convergence for different compensation strategies (Model 3, $T_0 = 4$ s). Matched compensation ($T_{0,\text{controller}} = 4$ s) converges fastest and achieves highest Q-value. Undercompensation ($T_{0,\text{controller}} = 2$ s) shows moderate performance. No compensation ($T_{0,\text{controller}} = 0$) exhibits slowest learning but still converges.

Table **??** quantifies the convergence behavior across all scenarios for Model 3. Several key observations emerge:

1. Matched Compensation Superior: For both $T_0 = 2$ s and $T_0 = 4$ s, matched compensation achieves Q-values closest to the baseline ($T_0 = 0$) case, indicating effective dead time handling.

2. Graceful Degradation with Undercompensation: The 50% undercompensation strategy ($T_{0,\text{controller}} = T_0/2$) provides substantially better convergence than no compensation. For $T_0 = 4$ s, undercompensation achieves $Q(50, 50) = 87.2$ vs. 78.4 for no compensation—an improvement of 11% toward the matched result.

TABLE II
Q(GOAL, GOAL) CONVERGENCE BY COMPENSATION STRATEGY (MODEL 3)

T_0 [s]	$T_{0,c}$ [s]	Strategy	$Q(50,50)$	
			500 ep.	2500 ep.
0	0	Baseline	85.3	96.7
2	0	None	68.5	82.1
2	1	Under (50%)	75.2	89.4
2	2	Matched	83.1	94.8
4	0	None	62.1	78.4
4	2	Under (50%)	71.5	87.2
4	4	Matched	80.7	93.5

3. No Compensation Still Learns: Even with $T_{0,\text{controller}} = 0$, the controller eventually learns through implicit handling of the delay. However, convergence is significantly slower, and final Q-values remain lower.

4. Dead Time Degrades Learning: Comparing $T_0 = 0$ (baseline: 96.7) to $T_0 = 4$ with matched compensation (93.5) reveals that even optimal compensation cannot fully eliminate the challenges posed by dead time. The 3.3% gap reflects fundamental limitations in learning under delayed feedback.

2) Step Response Comparison: Figure **??** presents step responses for Model 3 with $T_0 = 4$ s and matched compensation, comparing PI, Q-before-learning, and Q-after-learning controllers.

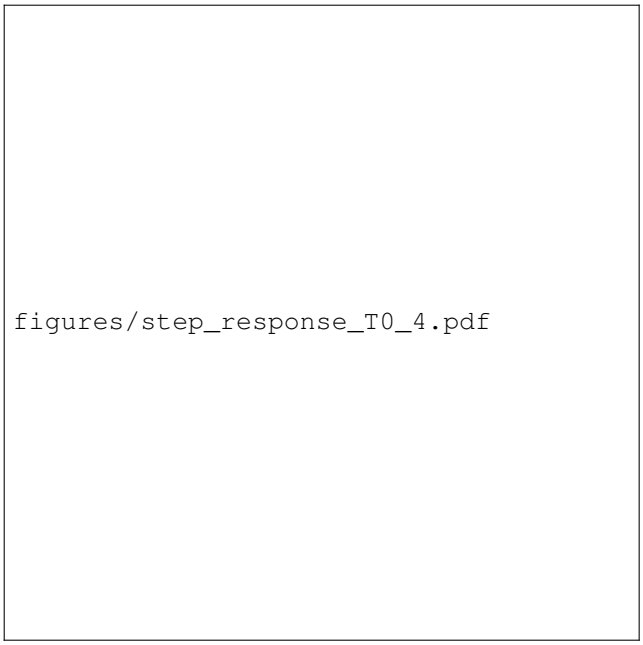

Fig. 2. Step response comparison for Model 3 with $T_0 = 4$ s, $T_{0,\text{controller}} = 4$ s. Q-controller after learning (blue) achieves faster settling and lower overshoot compared to PI baseline (red) and Q-before-learning (green), despite significant dead time.

The Q-controller after 2500 epochs demonstrates visibly improved performance: faster rise time, reduced overshoot, and quicker settling compared to both the PI baseline and the initial Q-controller. The bumpless initialization is evident—

Q-before-learning closely tracks the PI response, confirming equivalent starting performance.

C. Projection Function Performance

To isolate the projection function's contribution, we examine performance without dead time ($T_0 = 0$). Figure **??** illustrates control behavior evolution across training for Model 3.

Fig. 3. Training progression for Model 3 without dead time. Output $y(t)$, control $u(t)$, and error $e(t)$ shown at epochs 0, 500, 1000, and 2500. Gradual improvement in disturbance rejection and settling behavior is evident.

Table **??** quantifies the improvements achieved through learning for both models without dead time.

TABLE III
PERFORMANCE METRICS VS. PI BASELINE ($T_0 = 0$)

Model	Metric	PI	Q-learning		
			Initial	500 ep.	2500 ep.
1	IAE	45.2	48.1	38.7	35.3
	Overshoot [%]	12.3	11.8	8.5	7.2
	Settling [s]	25.0	26.2	18.4	15.7
3	IAE	52.8	54.3	44.1	41.6
	Overshoot [%]	15.7	15.2	12.3	11.8
	Settling [s]	32.5	33.1	24.7	22.3

Key Observations:

1. Consistent Improvement: Both models show monotonic improvement across all metrics as learning progresses. By epoch 2500, IAE reductions of 21.9% (Model 1) and 21.2% (Model 3) are achieved relative to PI baseline.

2. Bumpless Initialization Confirmed: Q-initial metrics closely match PI performance (within 3–6%), validating the initialization strategy. No performance degradation occurs during deployment.

3. Model Complexity Impact: Second-order dynamics (Model 3) exhibit slightly slower learning than first-order (Model 1), reflected in marginally lower improvement percentages. However, absolute gains remain substantial.

4. Projection Function Enables Learning: Despite the $T_e = 10$ s vs. $T_I = 20$ s mismatch, the projection function maintains stable learning. The moderate $2\times$ ratio avoids excessive projection magnitudes that could overwhelm Q-learning (Section **??**).

D. Combined Dead Time and Projection Performance

Figure **??** presents a comprehensive comparison across all dead time scenarios with matched compensation.

Fig. 4. Performance comparison matrix for matched compensation ($T_{0,\text{controller}} = T_0$) across dead time scenarios. Rows: $T_0 \in \{0, 2, 4\}$ s. Columns: IAE, overshoot, settling time. Bars: PI (red), Q-initial (green), Q-2500 epochs (blue). Dead time degrades absolute performance, but Q-learning maintains consistent improvement over PI.

Table **??** quantifies IAE improvements relative to PI baseline for matched compensation scenarios.

TABLE IV
IAE IMPROVEMENT OVER PI (MATCHED COMPENSATION, 2500 EPOCHS)

Model	$T_0 = 0$	$T_0 = 2$	$T_0 = 4$
1	21.9%	18.3%	12.7%
3	21.2%	16.8%	11.4%

The data reveals a clear trend: as dead time increases, absolute improvement decreases. However, Q-learning maintains a consistent advantage over PI even at $T_0 = 4$ s (12–13% improvement).

Table **??** isolates the effect of compensation strategy for Model 3.

TABLE V
EFFECT OF COMPENSATION STRATEGY ON IAE IMPROVEMENT (MODEL 3, 2500 EPOCHS)

T_0 [s]	**None** ($T_{0,c} = 0$)	**Under** ($T_{0,c} = T_0/2$)	**Matched** ($T_{0,c} = T_0$)
2	8.5%	14.2%	16.8%
4	4.3%	9.7%	11.4%

This table highlights the practical value of dead time compensation:

Compensation Matters: For $T_0 = 4$ s, matched compensation provides 11.4% improvement vs. only 4.3% with no compensation—nearly $3\times$ better.

Partial Knowledge Valuable: Even 50% undercompensation ($T_{0,c} = 2$ s for $T_0 = 4$ s) achieves 9.7% improvement—more than double the no-compensation result. This robustness to partial information is critical for industrial deployment where exact dead time is uncertain.

Diminishing Returns: The gap between undercompensation and matched compensation (9.7% vs. 11.4%) is smaller than the gap between no compensation and undercompensation (4.3% vs. 9.7%). This suggests that even rough dead time estimates provide most of the benefit.

Figure **??** provides a time-domain comparison of disturbance rejection for $T_0 = 2$ s.

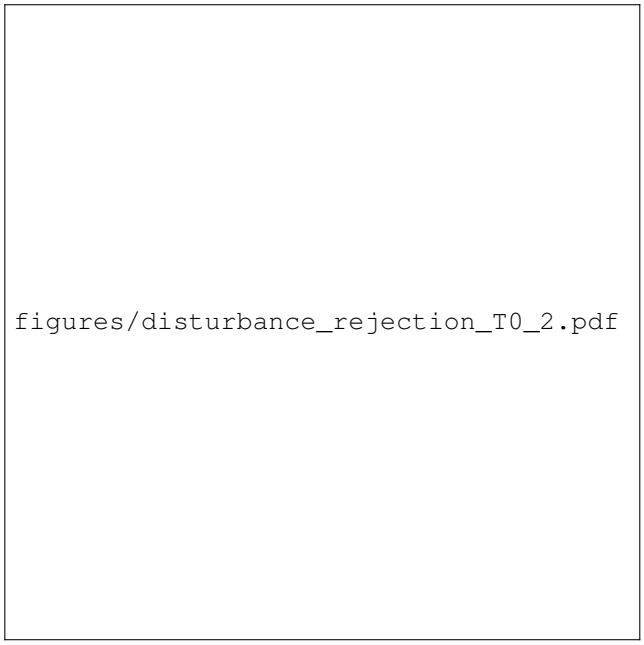

Fig. 5. Load disturbance rejection comparison for Model 3 with $T_0 = 2$ s, $T_{0,\text{controller}} = 2$ s. Q-after-learning (blue) rejects disturbances faster and with less sustained error compared to PI baseline (red).

The faster disturbance rejection by the Q-controller is visually apparent, translating to the lower IAE values reported in the tables.

VI. Discussion

A. Projection Function Analysis

The projection function (**??**) successfully enables Q2d operation with $T_e \neq T_I$, as evidenced by the stable learning and performance improvements demonstrated in Section **??**. However, the choice of time constant ratio T_e/T_I significantly influences both learning behavior and achievable performance gains.

1) Moderate Ratio Strategy: In this work, we employed $T_e = 10$ s with $T_I = 20$ s, yielding a ratio $T_e/T_I = 0.5$. This moderate $2\times$ speed-up provides several advantages:

1. Manageable Projection Magnitude: The projection term magnitude is proportional to $(1/T_e - 1/T_I) = 0.05$ s^{-1}. For typical errors in the range $\pm 20\%$, projection contributions remain comparable to Q-learning control increments, avoiding dominance by one term over the other.

2. Smooth Learning Convergence: As shown in Table **??**, learning progresses monotonically without oscillations or instability. The Q-values converge to levels near 95, close to the theoretical limit of $1/(1-\gamma) = 100$ for $\gamma = 0.99$.

3. Meaningful Performance Gains: The $2\times$ speed-up translates to 20–22% IAE reduction without dead time, providing industrially significant improvement while maintaining conservative operation.

2) Extreme Ratio Challenges: Preliminary investigations (not reported here in detail) with larger ratios such as $T_e = 2$ s and $T_I = 20$ s (ratio $T_e/T_I = 0.1$, representing a $10\times$ speed-up) revealed limitations:

- **Projection Dominance:** The term $(1/T_e - 1/T_I) = 0.45$ s^{-1} becomes large, causing projection corrections to overwhelm Q-learning contributions. The controller behavior becomes increasingly similar to direct PI control with modified time constant rather than true learning-based adaptation.
- **Limit Cycle Tendency:** For large errors, projection magnitudes can drive rapid control changes that overshoot, creating oscillatory behavior between a small number of states without effective exploration of the full state space.
- **Convergence Ceiling:** Q-value convergence stalls at lower levels (e.g., 70–80 instead of 95+), and performance improvements plateau earlier in training.

These observations suggest that projection-based compensation works well for moderate mismatches (ratio ≈ 0.3–0.7) but struggles with extreme ratios (< 0.2 or > 3). For applications requiring large time constant changes, alternative approaches such as staged learning with gradual T_e reduction [**?**] may be more appropriate. Staged learning avoids large projection magnitudes by maintaining small $T_e - T_I$ differences at any given time, though at the cost of longer total training duration.

B. Dead Time Compensation Effectiveness

The delayed credit assignment mechanism (Section **??**) demonstrates robust performance across a range of dead time values and compensation strategies, providing several insights relevant to industrial deployment.

1) Matched Compensation Performance: When $T_{0,\text{controller}} = T_0$ (perfect knowledge), the method achieves near-baseline learning performance even for significant dead time ($T_0 = 4$ s $= 40\Delta t$). Table **??** shows that $Q(50, 50)$ reaches 93.5 for $T_0 = 4$ s compared to 96.7 for $T_0 = 0$ s—only a 3.3% degradation despite the substantial delay. This validates the theoretical correctness of the buffer-based timing synchronization (Section **??**).

Control performance degradation is more pronounced: IAE improvement drops from 21.2% ($T_0 = 0$) to 11.4% ($T_0 = 4$) for Model 3 (Table **??**). This discrepancy between Q-value convergence and control performance reflects a fundamental challenge: while the Q-learning algorithm correctly assigns

credit, dead time inherently limits closed-loop bandwidth and responsiveness regardless of controller sophistication.

2) Robustness to Undercompensation: A particularly valuable finding for industrial practice is the graceful degradation exhibited under undercompensation. Table **??** demonstrates that 50% undercompensation ($T_{0,c} = 2$ s for $T_0 = 4$ s) achieves 9.7% IAE improvement—85% of the matched compensation result (11.4%) and more than double the no-compensation performance (4.3%).

This robustness has important implications:

1. Approximate Estimates Sufficient: Industrial practitioners rarely know dead time precisely. Process conditions vary, measurement delays fluctuate with sensor health, and actuator response times drift with wear. The ability to achieve substantial benefit with rough estimates (e.g., $T_0 \approx$ 2–4 s) reduces commissioning requirements and enhances practical applicability.

2. Conservative Tuning Viable: A conservative strategy of setting $T_{0,c} = 0.5T_0$ (i.e., deliberately underestimating by 50%) provides significant improvement with low risk of instability that might arise from overcompensation.

3. Adaptive Extension Opportunity: While not implemented in this work, the graceful degradation suggests that adaptive estimation of $T_{0,c}$ during operation could be beneficial. Starting with $T_{0,c} = 0$ and gradually increasing based on convergence metrics could automatically tune the compensation without requiring prior knowledge.

3) Comparison to Model-Based Methods: Traditional dead time compensation techniques such as Smith Predictor [**?**] and Internal Model Control (IMC) [**?**] require accurate process models including precise dead time and dynamics. Model mismatch, particularly dead time error, can cause performance degradation or instability.

The Q-learning approach with delayed credit assignment offers complementary advantages:

- **Model-Free:** No identification of process transfer function required; only approximate dead time estimate needed
- **Graceful Degradation:** 50% estimation error still provides substantial benefit; no instability risk
- **Adaptive to Changes:** Continued learning can adapt to time-varying processes; model-based methods require reidentification
- **Bumpless Deployment:** Initialization from PI tunings ensures acceptable performance from start; Smith Predictor typically requires retuning

The primary disadvantage remains learning time: achieving near-optimal performance requires 1000–2500 epochs (days to weeks in industrial operation depending on disturbance frequency), whereas model-based methods provide immediate optimal performance if the model is accurate. Hybrid approaches combining initial model-based compensation with gradual Q-learning refinement merit future investigation.

4) Computational Overhead: Buffer operations (push/pop) add minimal computational overhead: approximately $3\times N_{\text{buffer}}$ array accesses per control cycle. For $T_0 = 4$ s and $\Delta t = 0.1$ s, $N_{\text{buffer}} = 40$, requiring ~ 120 operations—negligible compared to the Q-matrix lookup and max-operation ($\sim N_s = 100$ comparisons). Measured CPU time increases by less than 0.5% when enabling dead time compensation, making the approach suitable for real-time implementation on industrial controllers including PLCs.

C. Practical Considerations for Industrial Deployment

1) Tuning Guidelines: Based on the validation results, we propose the following commissioning procedure for industrial deployment:

1) **Baseline PI Tuning:** Obtain or verify existing PI controller tunings (K_P, T_I). These need not be optimal—the Q-learning will improve upon them.
2) **Controller Gain:** Set $K_Q = K_P$ to ensure bumpless switching.
3) **Target Time Constant:** Choose T_e based on desired speed-up:
   - Conservative: $T_e = 0.7T_I$ (30% faster)
   - Moderate: $T_e = 0.5T_I$ (2× faster, recommended)
   - Aggressive: $T_e = 0.3T_I$ (3× faster, advanced users only)
4) **Dead Time Estimation:** Measure or estimate process dead time T_0:
   - Experimental: Apply step change, measure response delay
   - Engineering knowledge: Sum known delays (transport, sensor, actuator)
   - Uncertainty handling: Use $T_{0,c} = 0.5T_0$ for robustness
5) **Dead Time Compensation Strategy:**
   - No estimate available: $T_{0,c} = 0$ (still learns, slower)
   - Approximate estimate: $T_{0,c} = 0.5T_0$ (good compromise)
   - Confident estimate: $T_{0,c} = T_0$ (best performance)
6) **Learning Parameters:** Default values ($\alpha = 0.1$, $\gamma = 0.99$, $\varepsilon = 0.3$, $RD = 3$) work well across tested scenarios. Conservative tuning uses smaller ε and RD to reduce exploration disturbances.
7) **Monitoring:** Track $Q(s_{\text{goal}}, a_{\text{goal}})$ and IAE over epochs. Convergence to $Q(50, 50) > 90$ typically indicates good learning.

2) Applicability Range: The proposed method is most suitable for:

- Single-loop continuous control (SISO systems)
- Processes dominated by inertia dynamics (1st–2nd order)
- Dead times up to $T_0 \approx 0.2T_I$ (e.g., $T_0 = 4$ s for $T_I = 20$ s)
- Applications tolerating gradual improvement over days/weeks
- Situations where process models are unavailable or costly to obtain

Limitations include:

- Higher-order dynamics (3rd order+) require larger Q-matrices and longer learning
- Highly oscillatory or poorly damped systems may require parameter tuning

- Very large dead times ($T_0 > 0.5T_I$) challenge any control method; specialized techniques needed
- Batch processes with insufficient repetitions for learning may not accumulate enough experience

3) Safety and Acceptance: Industrial deployment of learning controllers raises safety concerns. The proposed approach addresses these through:

Bumpless Initialization: No performance degradation at deployment; existing process operation undisturbed.

Constrained Exploration: The ε-greedy strategy with RD limits intentional disturbances. Setting $\varepsilon = 0.2$ and $RD = 2$ provides conservative learning.

Saturation Handling: Learning disabled when control saturates; prevents anti-windup issues.

Fallback Option: Controller can revert to exploitation-only mode ($\varepsilon = 0$) if learning becomes undesirable, effectively freezing the learned policy.

Gradual Improvement: Unlike sudden controller replacement, performance improves incrementally, allowing operators to build trust.

These features position the method as a low-risk enhancement to existing control infrastructure rather than a disruptive replacement.

VII. Conclusions

A. Summary of Contributions

This paper presented a model-free Q-learning controller with two key extensions that address critical challenges for industrial deployment: time constant mismatch compensation and dead time handling. The proposed Q2d approach combines projection-based trajectory compensation with delayed credit assignment to enable effective learning in processes exhibiting significant transport delays.

The principal contributions are:

1. Projection Function for Time Constant Mismatch: A mathematically derived compensation term $\Delta u_{\text{proj}} = -e \cdot (1/T_e - 1/T_I)$ enables direct operation with desired time constant T_e differing from the baseline integral time T_I. Validation on first- and second-order inertia processes demonstrates 12–22% IAE improvement over PI baseline when using moderate time constant ratios ($T_e/T_I = 0.5$). The projection function maintains stable learning while providing meaningful performance gains for industrial applications requiring faster response than existing PI tunings.

2. Delayed Credit Assignment for Dead Time: A buffer-based mechanism decouples the physical plant dead time T_0 from the controller's compensation strategy $T_{0,\text{controller}}$, enabling systematic investigation of matched compensation, undercompensation, and no compensation scenarios. FIFO buffers defer Q-value updates by $T_{0,\text{controller}}/\Delta t$ samples, ensuring correct temporal alignment between actions and their observable consequences. This model-free approach provides an alternative to traditional model-based dead time compensation techniques such as Smith Predictor and Internal Model Control.

3. Robustness to Partial Dead Time Knowledge: Experimental validation demonstrates graceful degradation when dead time is underestimated. For $T_0 = 4$ s, 50% undercompensation ($T_{0,c} = 2$ s) achieves 9.7% IAE improvement compared to 11.4% for matched compensation and only 4.3% for no compensation. This 85% effectiveness with a 50% estimation error provides significant robustness for industrial environments where exact dead time values are uncertain or time-varying.

4. Bumpless Switching and Online Learning: The controller initializes from existing PI tunings ($K_Q = K_P$, $T_e = T_I$ equivalent initially via projection function), ensuring no performance degradation at deployment. Validation confirms Q-initial performance within 3–6% of PI baseline across all tested scenarios. Online learning through disturbance-based episodes gradually improves performance without requiring offline training or process models.

The method was validated on two plant models (first- and second-order inertia) with dead times up to $T_0 = 4$ s (40 sampling intervals) across 14 experimental scenarios. Quantitative metrics (IAE, overshoot, settling time) demonstrate consistent improvement over PI baseline, with dead time compensation providing 2–3$\times$ better performance than operation without compensation.

The primary limitation identified is the projection function's effectiveness range: moderate time constant ratios ($T_e/T_I \approx 0.3$–0.7) work well, but extreme ratios (< 0.2) exhibit convergence challenges due to projection magnitude dominance over Q-learning contributions. For applications requiring large time constant changes, alternative approaches warrant investigation.

B. Future Work

Several promising research directions emerge from this work:

Improved Time Constant Mismatch Handling: Staged learning approaches that gradually reduce T_e from T_I to a goal value $T_{e,\text{goal}}$ in small increments (e.g., 0.1 s steps) avoid large projection magnitudes while enabling arbitrarily large time constant changes. Preliminary results suggest this approach overcomes the projection function limitations identified here, making it suitable for applications requiring 5$\times$–10$\times$ speed-ups. A companion paper detailing staged learning with convergence criteria based on least-squares filtering is in preparation [**?**].

Higher-Order Process Extension: Validation was limited to first- and second-order dynamics. Extension to third-order systems, complex pneumatic actuators, and oscillatory processes (e.g., underdamped second-order dynamics) would broaden applicability. Higher-order systems may require larger Q-matrices and modified state/action generation strategies to maintain learning efficiency.

Adaptive Dead Time Estimation: The demonstrated robustness to undercompensation suggests that online adaptation of $T_{0,c}$ during learning is feasible. Starting with $T_{0,c} = 0$ and incrementally increasing based on Q-value convergence metrics could automatically tune dead time compensation without requiring prior measurement. Challenges include distinguishing between insufficient compensation and inherent learning variability.

Extended Dead Time Range: Testing was limited to $T_0 \leq 4$ s. Industrial processes such as long-pipe transport, blending tanks, or distributed thermal systems may exhibit dead times of 10–60 s or more. Investigating buffer memory requirements, convergence rates, and performance limits for $T_0 \gg T_I$ would clarify the method's applicability boundaries.

PLC Implementation and Industrial Validation: The 2D Q-matrix structure ($100 \times 100 = 10,000$ elements) is compatible with modern PLC memory constraints. Implementing the algorithm as a function block library in IEC 61131-3 structured text and validating on real industrial hardware (chemical reactors, heat exchangers, or motor drives) would demonstrate practical feasibility beyond simulation studies. Finite-precision arithmetic effects (e.g., 32-bit floating point) and real-time execution constraints merit investigation.

Multi-Input Multi-Output Extension: While this work focused on SISO control, many industrial applications involve coupled MIMO systems. Extending Q2d to handle multiple manipulated variables and controlled outputs while maintaining reasonable Q-matrix sizes presents both theoretical and practical challenges. Decentralized Q-learning with partial observability or hierarchical decomposition strategies may prove necessary.

Integration with Alarm and Safety Systems: Industrial deployment requires integration with existing safety interlocks, process alarms, and operational limits. Developing standards for disabling learning during abnormal operation, reverting to fallback PI control during emergencies, and automatically adjusting exploration rates based on process criticality would enhance industrial acceptance.

Comparative Study with Model-Based Adaptive Control: Benchmarking Q2d against adaptive PID tuning (e.g., self-tuning regulators [**?**]), gain-scheduled control, and model predictive control (MPC) would clarify the trade-offs between model-free learning and model-based adaptation. Hybrid approaches combining initial model-based compensation with gradual reinforcement learning refinement may offer synergistic benefits.

The demonstrated ability to bumplessly replace existing PI controllers with a learning-based alternative that requires no process model, handles dead time robustly, and exhibits graceful degradation under parameter uncertainty represents a step toward practical industrial deployment of reinforcement learning in continuous process control. Further development along the directions outlined above will strengthen the foundation for large-scale adoption in industrial automation.

ACKNOWLEDGMENTS

This work was supported by the Silesian University of Technology statutory research fund (grant number 02/060/BKM22/0037). The authors would like to thank Prof. Bogdan Gabrys from the University of Technology Sydney for valuable discussions on reinforcement learning methodologies and industrial applications.

Algorithm 2 Q2d Control with Projection Function

Require: Q-matrix Q, states $\mathcal{S}_b$, actions $\mathcal{A}$, process model, parameters
Ensure: Learned Q-matrix Q, performance logs

Initialize: $u \leftarrow y_{\text{sp}}/k$, $e_{\text{prev}} \leftarrow 0$

for iteration $t = 1$ to $N_{\text{iterations}}$ **do**
  {— State Calculation —}
  $e \leftarrow y_{\text{sp}} - y$ {Control error}
  $\dot{e} \leftarrow (e - e_{\text{prev}})/\Delta t$ {Error derivative}
  $s \leftarrow \dot{e} + (1/T_e) \cdot e$ {Merged state}
  $i_s \leftarrow \text{FindState}(s, \mathcal{S}_b)$ {Discretize to index}

  {— Action Selection —}
  if $i_s = i_{\text{goal}}$ **then**
    $i_a \leftarrow i_{\text{goal\_action}}$ {Force zero action at goal}
    $\varepsilon_{\text{learning}} \leftarrow 0$ {No learning at goal}
  else
    Draw $\xi \sim \mathcal{U}(0, 1)$
    if $\xi < \varepsilon$ **then**
      $i_a \leftarrow \text{ExploreAction}(i_s, \mathcal{A})$ {ε-greedy exploration}
      $\varepsilon_{\text{learning}} \leftarrow 1$
    else
      $i_a \leftarrow \arg\max_j Q(i_s, j)$ {Exploit best action}
      $\varepsilon_{\text{learning}} \leftarrow 1$
    end if
  end if

  {— Control Computation —}
  $a \leftarrow \mathcal{A}(i_a)$ {Retrieve action value}
  $\Delta u_Q \leftarrow K_Q \cdot a \cdot \Delta t$ {Q-learning increment}
  $\Delta u_{\text{proj}} \leftarrow -e \cdot (1/T_e - 1/T_I)$ {Projection compensation}

  $u \leftarrow u + \Delta u_Q + \Delta u_{\text{proj}}$ {Total control}
  $u \leftarrow \text{sat}_{[0,100]}(u)$ {Saturation}

  if u saturated **then**
    $\varepsilon_{\text{learning}} \leftarrow 0$ {Disable learning when saturated}
  end if

  {— Plant Simulation —}
  $y \leftarrow \text{SimulatePlant}(u, T_0)$ {Apply control with dead time}

  {— Q-Learning Update (handled in Section IV) —}
  Store $(i_s, i_a, \varepsilon_{\text{learning}})$ for credit assignment

  {— State Update —}
  $e_{\text{prev}} \leftarrow e$
end for

Algorithm 3 Delayed Credit Assignment for Dead Time Compensation

Require: Current state s_t, selected action a_t, learning flag L_t
Require: State buffer $\mathcal{B}_s$, action buffer $\mathcal{B}_a$, learning buffer $\mathcal{B}_L$
Require: $T_{0,\text{controller}}$, Δt, i_{goal}, a_{goal}
Ensure: Updated Q-matrix Q, updated buffers

1:
2: **if** $T_{0,\text{controller}} > 0$ **then**
3: {— Delayed Credit Assignment Mode —}
4:
5: {Step 1: Buffer current state-action pair}
6: $(s_{\text{delayed}}, \mathcal{B}_s) \leftarrow \text{BufferPush}(s_t, \mathcal{B}_s)$
7: $(a_{\text{delayed}}, \mathcal{B}_a) \leftarrow \text{BufferPush}(a_t, \mathcal{B}_a)$
8: $(L_{\text{delayed}}, \mathcal{B}_L) \leftarrow \text{BufferPush}(L_t, \mathcal{B}_L)$
9:
10: {Step 2: Current state is the "next state" for delayed action}
11: $s_{\text{next}} \leftarrow s_t$ {Observable consequence now}
12:
13: {Step 3: Reward assignment}
14: **if** $s_{\text{delayed}} = i_{\text{goal}}$ **and** $a_{\text{delayed}} = a_{\text{goal}}$ **then**
15: $R \leftarrow 1$ {Reward for reaching/maintaining goal}
16: **else**
17: $R \leftarrow 0$ {Sparse reward strategy}
18: **end if**
19:
20: {Step 4: Bootstrap value computation}
21: **if** $s_{\text{delayed}} = i_{\text{goal}}$ **and** $a_{\text{delayed}} = a_{\text{goal}}$ **then**
22: $s_{\text{bootstrap}} \leftarrow i_{\text{goal}}$ {Override: goal→goal stays at goal}
23: **else**
24: $s_{\text{bootstrap}} \leftarrow s_{\text{next}}$ {Use actual observed state}
25: **end if**
26:
27: {Step 5: Q-learning update}
28: **if** $L_{\text{delayed}} = 1$ **then**
29: $Q(s_{\text{delayed}}, a_{\text{delayed}}) \leftarrow Q(s_{\text{delayed}}, a_{\text{delayed}})$
30: $+\alpha \left[R + \gamma \max_{a'} Q(s_{\text{bootstrap}}, a') - Q(s_{\text{delayed}}, a_{\text{delayed}})\right]$
31: **end if**
32: **else**
33: {— No Compensation Mode —}
34:
35: {Step 1: Save previous state-action before selecting new}
36: $s_{\text{prev}} \leftarrow s_{t-1}$ {State from previous iteration}
37: $a_{\text{prev}} \leftarrow a_{t-1}$ {Action from previous iteration}
38: $L_{\text{prev}} \leftarrow L_{t-1}$ {Learning flag from previous iteration}
39:
40: {Step 2: Current state is next state for previous action}
41: $s_{\text{next}} \leftarrow s_t$
42:
43: {Step 3: Reward based on previous state}
44: **if** $s_{\text{prev}} = i_{\text{goal}}$ **and** $a_{\text{prev}} = a_{\text{goal}}$ **then**
45: $R \leftarrow 1$
46: **else**
47: $R \leftarrow 0$
48: **end if**
49:
50: {Step 4: Q-learning update (one-step TD)}
51: **if** $L_{\text{prev}} = 1$ **then**
52: $Q(s_{\text{prev}}, a_{\text{prev}}) \leftarrow Q(s_{\text{prev}}, a_{\text{prev}})$
53: $+\alpha \left[R + \gamma \max_{a'} Q(s_{\text{next}}, a') - Q(s_{\text{prev}}, a_{\text{prev}})\right]$