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.**

In this paper, we present a model-free Q-learning controller with two key extensions that address these industrial challenges. First, a projection function enables bumpless initialization when the desired closed-loop time constant T_e differs from the baseline integral time T_I of the existing PI controller. The projection term compensates for this mismatch through an additional control contribution derived from first-order trajectory requirements, with sign protection to ensure stability near the setpoint. Second, a delayed credit assignment mechanism handles dead time compensation by decoupling the physical plant delay T_0 from the controller's compensation strategy $T_{0,\text{controller}}$, implemented through FIFO buffers that defer Q-value updates until action consequences become observable. The buffer design includes temporal alignment correction and bootstrap override for numerical stability.

The proposed Q2d controller bumplessly replaces existing PI controllers, starting with equivalent initial performance through proper initialization of the Q-matrix as an identity matrix and appropriate selection of controller gain $K_Q = K_P$. The controller then gradually improves online through reinforcement learning without requiring a process model or offline training. We validate the approach on first- and second-order inertia processes with dead times up to 4 seconds. Notably, the method exhibits graceful degradation: even 50% undercompensation of dead time (setting $T_{0,\text{controller}} = T_0/2$) provides substantially better convergence 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

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.

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 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. In our previous work, we addressed this challenge by initializing the Q-learning controller from existing PI 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 and $e = Y_{sp} - Y$ is the control error. By initializing the Q-matrix as an identity matrix and setting the controller gain $K_Q = K_P$, the Q2d controller starts with predictable 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 when the desired closed-loop time constant T_e differs from the baseline integral time T_I:

Challenge 1: Initialization with Time Constant Mismatch. The Q2d approach achieves bumpless switching when initialized with $T_e = T_I$, matching the existing PI integral time. However, in many industrial scenarios, operators desire faster closed-loop response than the existing PI tuning provides (e.g., $T_e = 5$ s vs. $T_I = 20$ s). Direct initialization with $T_e \neq T_I$ breaks the equivalence between PI and Q2d control laws, potentially causing performance degradation at deployment. A mechanism is needed to enable bumpless initialization even when T_e and T_I differ.

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. When the dead time T_0 spans multiple sampling intervals, naive application of the Q-learning update rule incorrectly associates current observations with recent actions rather than the actions that actually caused those observations. Existing Q-learning literature provides limited guidance for systematic dead time compensation in continuous control applications.

In this paper, we address both challenges through extensions to the Q2d framework:

1) **Projection Function for Bumpless Initialization:** We derive a compensation term $\Delta U_{\text{proj}} = -e \cdot (1/T_e - 1/T_I)$ that enables bumpless switching even when $T_e \neq T_I$. The projection function compensates for the structural difference between the PI control law (based on T_I) and the desired Q2d trajectory (based on T_e), allowing initialization with the Q-matrix as an identity matrix and $K_Q = K_P$ regardless of the T_e–T_I mismatch. This eliminates the restriction $T_e = T_I$ at deployment while maintaining predictable initial performance.
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 $T_{0,\text{controller}}/T_s$, 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 convergence 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. We validate the approach on first- and second-order inertia processes with dead times up to 4 seconds, demonstrating that dead time compensation with even partial knowledge significantly improves learning convergence compared to 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

We consider a single-input single-output (SISO) dynamical process with control signal $U(t)$, process output $Y(t)$, setpoint $Y_{sp}(t)$, and control error $e(t) = Y_{sp}(t) - Y(t)$. 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 and $\dot{e}(t) = de/dt$ is the error derivative. 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:

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

where K_{PI} is the proportional gain, T_I is the integral time, and T_s 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.

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 $s(t + T_s)$. When dead time T_0 spans multiple sampling intervals ($T_0 \gg T_s$), 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 $T_s = 0.1$ 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 $s(t + T_s)$ with $a(t)$, leading to poor or failed learning. In Section **??**, we address 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: Enabling Bumpless Initialization

The primary purpose of the projection function is to enable bumpless switching from an existing PI controller to the Q2d controller even when the desired trajectory time constant T_e differs from the baseline integral time T_I. Without the projection function, bumpless initialization requires $T_e = T_I$, restricting the controller's initial operating point.

1) Structural Mismatch Problem: When we initialize the Q2d controller with:

- Q-matrix as identity matrix
- Controller gain $K_Q = K_{PI}$
- State/action spaces generated for time constant T_e

the resulting control law structurally implements trajectory dynamics based on T_e via the merged state (**??**). However, the existing PI controller operates with trajectory dynamics based on T_I via (**??**). When $T_e \neq T_I$, this creates a mismatch that prevents equivalent initial behavior.

2) Projection Compensation: To achieve bumpless switching despite $T_e \neq T_I$, we introduce a projection term that compensates for the structural difference. Consider the desired trajectory requirement with T_e:

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

versus the PI controller's implicit trajectory with T_I:

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

The difference is:

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

We compensate for this mismatch by adding a control term:

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

The negative sign ensures correct compensation direction. When $T_e < T_I$ (faster response desired), $(1/T_e - 1/T_I) > 0$, and the projection adds negative correction for positive errors. When $T_e = T_I$, the projection vanishes.

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

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

where $a(s)$ is the action selected for the current merged state s. The control signal is saturated:

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

with typical bounds $U_{\min} = 0\%$ and $U_{\max} = 100\%$ for industrial applications.

4) Projection Implementation with Sign Protection: For improved numerical precision, the projection is implemented using the continuous merged state value rather than the discretized action. The effective action value becomes:

$$a_{\text{eff}} = s - \Delta U_{\text{proj}} = \left(\dot{e} + \frac{e}{T_e}\right) - e \cdot \left(\frac{1}{T_e} - \frac{1}{T_I}\right) = \dot{e} + \frac{e}{T_I} \tag{17}$$

This formulation ensures that the control law precisely matches the PI velocity form (**??**), avoiding quantization errors that would arise from using the discretized action value.

To ensure stability near the setpoint, the projection incorporates sign protection based on error magnitude. We define a threshold $e_{\text{th}} = 2 \cdot \text{prec}$ to distinguish between transient and steady-state operation:

Large Error (Transient): When $|e| > e_{\text{th}}$, the projection is applied without sign protection. This allows proper trajectory translation during setpoint changes, where the projection may legitimately reverse the control direction. For example, with $T_e = 5$ s and $T_I = 20$ s, a positive error produces negative projection that adjusts the more aggressive T_e-based control toward the slower T_I-based baseline.

Small Error (Near Steady State): When $|e| \leq e_{\text{th}}$, sign protection is applied. If the projection would reverse the sign of the control increment, it is disabled to prevent oscillations:

$$\Delta U_{\text{proj}} = \begin{cases} -e \cdot \left(\frac{1}{T_e} - \frac{1}{T_I}\right) & \text{if } \text{sign}(a - \Delta U_{\text{proj}}) = \text{sign}(a) \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

This conditional application ensures stable behavior near the setpoint while maintaining correct trajectory translation during transients.

C. State and Action Generation: Geometric Distribution

Unlike the exponential distribution used in Q3d [**?**], we employ a geometric distribution that provides efficient scaling and maintains desired precision properties. This section

describes the generation procedure implemented in our MATLAB function `f_generuj_stany_v2.m` when operating with projection function enabled (`f_rzutujaca_on = 1`).

1) Geometric Action Distribution: We first generate actions geometrically, then place states at midpoints between consecutive actions. The procedure for positive actions is:

Step 1: Scale upper control limit

$$U_{\text{scaled}} = \frac{U_{\max}}{K_Q \cdot T_s} \tag{19}$$

This accounts for controller gain and sampling time in the discretization.

Step 2: Define smallest action

$$a_1 = \frac{2 \cdot \text{prec}}{T_e} \tag{20}$$

where prec is the precision parameter defining steady-state accuracy. The factor 2 ensures that when states are placed at action midpoints, the smallest state value becomes prec/T_e, which guarantees $\pm$prec error tolerance (see Subsection **??**).

Step 3: Calculate geometric ratio

$$q = \left(\frac{U_{\text{scaled}}}{a_1} \right)^{1/(N-1)} \tag{21}$$

where N is the expected number of positive actions (user-specified).

Step 4: Generate positive actions

$$\begin{aligned} a_1 &= 0 \quad \text{(goal action)} \\ a_2 &= \frac{2 \cdot \text{prec}}{T_e} \\ a_i &= a_2 \cdot q^{i-2} \quad \text{for } i = 3, 4, \ldots, N \end{aligned} \tag{22}$$

This creates dense action spacing near zero (precise control) and coarse spacing at large values (aggressive correction).

2) State Generation as Action Midpoints: States are placed at midpoints between consecutive actions:

$$s_i = \frac{a_{i+1} + a_i}{2} \quad \text{for } i = 1, 2, \ldots, N-1 \tag{23}$$

This results in $N - 1$ positive states. The smallest positive state is:

$$s_1 = \frac{a_2 + a_1}{2} = \frac{a_2}{2} = \frac{\text{prec}}{T_e} \tag{24}$$

3) Symmetric Extension: To handle negative errors and control increments, we extend symmetrically:

$$\begin{aligned} \mathcal{S} &= [-s_{N-1}, -s_{N-2}, \ldots, -s_1, s_1, \ldots, s_{N-1}] \\ \mathcal{A} &= [-a_N, -a_{N-1}, \ldots, -a_2, 0, a_2, \ldots, a_N] \end{aligned} \tag{25}$$

resulting in:

- $N_s = 2(N-1)$ states
- $N_a = 2N - 1$ actions
- Goal state index: $i_{\text{goal}} = N - 1$ (center)
- Goal action index: $j_{\text{goal}} = N$ (center, zero increment)

4) Precision Invariance Property: A critical property of this discretization is T_e-invariant error tolerance. In steady state, $\dot{e} = 0$, so the merged state becomes:

$$s_{ss} = \frac{1}{T_e} e_{ss} \tag{26}$$

The goal state spans $s \in (-s_1, s_1]$ where $s_1 = \text{prec}/T_e$. Substituting:

$$-\frac{\text{prec}}{T_e} < \frac{e_{ss}}{T_e} \leq \frac{\text{prec}}{T_e} \tag{27}$$

Multiplying by T_e yields:

$$-\text{prec} < e_{ss} \leq \text{prec} \tag{28}$$

Thus, the steady-state error tolerance remains $\pm$prec regardless of T_e. This property is essential if T_e were changed during operation (e.g., in staged learning approaches), though in this work T_e remains constant.

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

$$Q_{ij}(0) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{29}$$

where i indexes states and j indexes actions. This ensures each state initially selects its corresponding action (diagonal elements), providing stable starting behavior equivalent to the PI controller structure when combined with the projection function.

D. Control Algorithm Implementation

Algorithm **??** presents the Q2d control loop with projection function.

1) Key Implementation Details: **Manual Control Phase:** The first few samples (typically 5) use manual control $U = Y_{sp}/k$ to initialize toward steady state and pre-fill dead time buffers.

Goal State Enforcement: When $s \approx 0$ (goal state), the controller always selects $a = 0$ (goal action) to maintain steady state.

Saturation Handling: When U saturates, Q-learning updates are disabled to prevent learning from unrealizable actions (anti-windup).

Exploration Strategy with Same-Side Matching: The function ExploreAction(i_s, Q) selects actions within a constrained range around the best action, subject to a critical directional constraint called *same-side matching*. This constraint ensures that explored actions maintain correct control direction:

- States above the goal ($i_s > i_{\text{goal}}$, corresponding to $s < 0$) may only select actions above the goal action ($i_a > i_{\text{goal\_action}}$, corresponding to negative control increments)
- States below the goal ($i_s < i_{\text{goal}}$, corresponding to $s > 0$) may only select actions below the goal action ($i_a < i_{\text{goal\_action}}$, corresponding to positive control increments)

This constraint prevents counterproductive exploration, such as applying positive control increments when the error is already positive and growing. The exploration range is further limited by the parameter RD (random deviation), which constrains action selection to $[i_{a,\text{best}} - RD, i_{a,\text{best}} + RD]$ where $i_{a,\text{best}}$

Algorithm 1 Q2d Control with Projection Function

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

Initialize: $U \leftarrow Y_{sp}/k$, $e_{\text{prev}} \leftarrow 0$, $i_{s,\text{prev}} \leftarrow i_{\text{goal}}$, $i_{a,\text{prev}} \leftarrow i_{\text{goal\_action}}$
for iteration $t = 1$ to $N_{\text{iterations}}$ **do**
  {State Calculation}
  $e \leftarrow Y_{sp} - Y$
  $\dot{e} \leftarrow (e - e_{\text{prev}})/T_s$
  $s \leftarrow \dot{e} + (1/T_e) \cdot e$
  $i_s \leftarrow \text{FindState}(s, \mathcal{S})$

  {Save Previous Iteration Values (for Q-update)}
  $i_{s,\text{old}} \leftarrow i_{s,\text{prev}}$; $i_{a,\text{old}} \leftarrow i_{a,\text{prev}}$; $R_{\text{old}} \leftarrow R_{\text{prev}}$

  {Action Selection (before buffering)}
  if $i_s = i_{\text{goal}}$ **then**
    $i_a \leftarrow i_{\text{goal\_action}}$; $R \leftarrow 1$ {Force zero at goal}
  else
    $R \leftarrow 0$
    Draw $\xi \sim \mathcal{U}(0, 1)$
    if $\xi < \varepsilon$ **then**
      $i_a \leftarrow \text{ExploreAction}(i_s, Q)$ {Same-side matching}
    else
      $i_a \leftarrow \arg\max_j Q(i_s, j)$
    end if
  end if

  {Control Computation with Projection}
  $a \leftarrow s$ {Use continuous state value}
  $\Delta U_{\text{proj}} \leftarrow \text{ComputeProjection}(e, a)$ {With sign protection}
  $U \leftarrow U + K_Q \cdot (a - \Delta U_{\text{proj}}) \cdot T_s$
  $U \leftarrow \text{sat}_{[U_{\min}, U_{\max}]}(U)$

  {Plant Simulation}
  $Y \leftarrow \text{SimulatePlant}(U, T_0)$

  {Q-Update (see Algorithm **??**)}
  Update Q-matrix using $(i_{s,\text{old}}, i_{a,\text{old}}, R_{\text{old}}, i_s)$

  $i_{s,\text{prev}} \leftarrow i_s$; $i_{a,\text{prev}} \leftarrow i_a$; $R_{\text{prev}} \leftarrow R$
  $e_{\text{prev}} \leftarrow e$
end for

is the best action for the current state according to the Q-matrix. If no valid action satisfying both constraints can be found within a maximum number of attempts, the algorithm falls back to exploitation without Q-update.

The Q-learning update mechanism with 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 $t + T_s$.

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

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{30}$$

where s_t is the state at time t, a_t is the action taken, and s_{t+1} is the state observed at $t + T_s$. 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 $T_s = 0.1$ 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:

$$N_{\text{buffer}} = \text{round}\left(\frac{T_{0,\text{controller}}}{T_s}\right) + 1 \tag{31}$$

The additional sample (+1) accounts for the discrete-time update sequence: the plant output $Y(k)$ at iteration k reflects the control signal from iteration $k - N_{\text{buffer}}$, requiring one additional buffer element for correct temporal alignment. For example, with $T_{0,\text{controller}} = 4$ s and $T_s = 0.1$ s, the buffer holds $N_{\text{buffer}} = 41$ elements.

Algorithm **??** presents our delayed credit assignment procedure.

3) Reward Logic for Goal State Maintenance: The reward assignment in Algorithm **??** includes two conditions that grant positive reward:

Goal Arrival: $R = 1$ when $s_{\text{next}} = i_{\text{goal}}$, rewarding transitions that successfully reach the goal state.

Goal Maintenance: $R = 1$ when $s_{\text{delayed}} = i_{\text{goal}}$ and $a_{\text{delayed}} = a_{\text{goal}}$, ensuring that the state-action pair $(i_{\text{goal}}, a_{\text{goal}})$ continues to receive positive reward even when disturbances cause temporary deviations from the goal.

This dual condition ensures that $Q(i_{\text{goal}}, a_{\text{goal}})$ maintains the highest Q-value in the matrix, which is essential for the value gradient that guides the controller toward the goal state.

4) Bootstrap Override for Numerical Stability: When the buffered state-action pair corresponds to the goal state with goal action, the bootstrap state is overridden to the goal state regardless of the actual next state. This prevents numerical drift over the buffer delay from contaminating the Q-value updates. Small discretization errors accumulated over N_{buffer} time steps can cause the observed next state to be adjacent to (rather than exactly at) the goal state. Without the override, this would cause $Q(i_{\text{goal}}, a_{\text{goal}})$ to bootstrap from a lower-valued adjacent state, inappropriately reducing its value over time.

5) 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/T_s
3) **Time** t **(controller):** State-action pair $(s(t), a(t))$ enters controller's buffer of length $T_{0,\text{controller}}/T_s$
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.

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

- Plant buffers: Filled with $U = Y_{sp}/k$ (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 based on goal state presence:

$$R = \begin{cases} 1 & \text{if } s_{\text{next}} = s_{\text{goal}} \text{ (goal arrival)} \\ 1 & \text{if } s = s_{\text{goal}} \text{ and } a = a_{\text{goal}} \text{ (goal maintenance)} \\ 0 & \text{otherwise} \end{cases} \tag{32}$$

where s_{goal} is the discrete state closest to $s = 0$, a_{goal} is the zero control increment action, and s_{next} is the observed next state after taking action a in state s.

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 asymptotically approaches:

$$Q(s_{\text{goal}}, a_{\text{goal}}) \to \frac{1}{1-\gamma} \tag{33}$$

as learning progresses and the goal state is visited repeatedly. For $\gamma = 0.99$, the theoretical limit is $Q(s_{\text{goal}}, a_{\text{goal}}) = 100$. 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: We maintain buffers across episode boundaries:

- 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, we record convergence metrics for that episode, apply a new disturbance or setpoint change, and continue learning 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, we explicitly reset buffers to initial conditions to ensure fair comparison between PI baseline, Q-controller before learning, and Q-controller after learning. Importantly, during verification experiments, we disable exploration ($\varepsilon = 0$) to evaluate the learned policy in pure exploitation mode. This provides a clean starting point for quantitative performance evaluation without the influence of random exploratory actions.

V. Validation and Results

A. Experimental Setup

We validate the proposed Q2d controller with projection function and dead time compensation through simulation studies on two benchmark process models representing common industrial dynamics. All experiments are conducted in MATLAB/Simulink environment with sampling time $T_s = 0.1$ s.

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

$$G_1(s) = \frac{k}{Ts+1} = \frac{1}{5s+1} \tag{34}$$

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):

$$G_3(s) = \frac{k}{(T_1 s+1)(T_2 s+1)} = \frac{1}{(5s+1)(2s+1)} \tag{35}$$

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: We investigate three dead time values:

- $T_0 = 0$ 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 $T_0 > 0$, we test three compensation strategies:

- **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, totaling 14 experiments.

3) Controller Parameters: **PI Baseline:** We use standard Siemens PLC default tunings [**?**]:

- Proportional gain: $K_{PI} = 1$
- 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_{PI})
Goal time constant	T_e	5 s
Baseline integral time	T_I	20 s
Projection function	–	Enabled
Learning rate	α	0.1
Discount factor	γ	0.99
Exploration rate (training)	ε	0.3
Exploration rate (verification)	ε	0
Random deviation	RD	3
Precision	prec	0.5%
Expected states	–	100
Training epochs	–	2500
Sampling time	T_s	0.1 s
Control limits	$[U_{\min}, U_{\max}]$	[0, 100]%

The choice $T_e = 5$ s represents a 4× speed-up compared to the baseline $T_I = 20$ s. This ratio provides significant performance improvement potential while remaining within the capabilities of typical industrial actuators.

4) Learning Protocol: **Training Mode:** Disturbance-based learning

- Exploration enabled: $\varepsilon = 0.3$
- Load disturbances randomly drawn from normal distribution 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 $Y_{sp} = 50\%$ during training
- Total training duration: 2500 epochs

Verification Mode: Clean experimental tests

- **Exploration disabled**: $\varepsilon = 0$ (pure exploitation of learned policy)
- Three phases: setpoint tracking, load disturbance rejection, setpoint tracking
- Setpoint steps: $50\% \to 70\% \to 50\%$ (20% magnitude)
- Load disturbance: $d = \pm 0.3$ applied at specified times
- Total duration: 600 s
- Buffers reset to initial conditions before each verification run
- Conducted three times: with PI controller, with Q-controller before learning, with Q-controller after learning

The critical distinction is that verification experiments use pure exploitation ($\varepsilon = 0$) to evaluate the learned policy without random exploratory actions affecting the measurements.

Performance Metrics:

We compute the following metrics from verification experiments:

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

B. Experimental Results

[Note: This section will be populated with actual experimental data after running the 14 validation scenarios described above. Tables and figures will present quantitative comparisons between PI baseline, Q-controller before learning, and Q-controller after 2500 epochs of training.]

1) Q-Learning Convergence Analysis: We will track the evolution of $Q(s_{\text{goal}}, a_{\text{goal}})$ over the 2500 training epochs for all scenarios. The Q-value at the goal state serves as a convergence indicator—values approaching the theoretical limit of $1/(1-\gamma) = 100$ indicate successful learning.

Planned Analysis:

- Convergence rate comparison across compensation strategies
- Effect of dead time magnitude on learning speed
- Identification of plateau points indicating convergence

Figure 1 (to be generated): Q-learning convergence curves showing $Q(s_{\text{goal}}, a_{\text{goal}})$ vs. epoch number for Model 3 with $T_0 = 4$ s under three compensation strategies (no compensation, 50% undercompensation, matched compensation).

2) Dead Time Compensation Effectiveness: We will quantify how compensation strategy affects final learning performance.

Table 1 (to be populated): Q-value convergence by compensation strategy for Model 3

Expected observations:

- Matched compensation should yield highest Q-values
- Undercompensation (50%) should show moderate performance
- No compensation should exhibit slowest convergence but eventual learning

TABLE II
Q(GOAL, GOAL) CONVERGENCE BY COMPENSATION STRATEGY (MODEL 3) - *Results Pending*

T_0 [s]	$T_{0,c}$ [s]	Strategy	Q @ 500 ep.	Q @ 2500 ep.
0	0	Baseline	–	–
2	0	None	–	–
2	1	Under (50%)	–	–
2	2	Matched	–	–
4	0	None	–	–
4	2	Under (50%)	–	–
4	4	Matched	–	–

Figure 2 (to be generated): Step response comparison showing output $Y(t)$ for PI, Q-before-learning, and Q-after-learning controllers (Model 3, $T_0 = 4$ s, matched compensation).

3) Projection Function Performance: To isolate the projection function's contribution, we will examine performance without dead time ($T_0 = 0$).

Table 2 (to be populated): Performance metrics vs. PI baseline for $T_0 = 0$

TABLE III
PERFORMANCE METRICS VS. PI BASELINE ($T_0 = 0$) - *Results Pending*

Model	Metric	PI	Q-init	Q-500ep	Q-2500ep
1	IAE	–	–	–	–
1	Overshoot [%]	–	–	–	–
1	Settling [s]	–	–	–	–
3	IAE	–	–	–	–
3	Overshoot [%]	–	–	–	–
3	Settling [s]	–	–	–	–

Key validation points:

- Q-initial metrics should closely match PI (bumpless initialization verification)
- Q-final metrics should show improvement over PI (learning effectiveness)
- Second-order model (Model 3) expected to show more challenging dynamics than first-order (Model 1)

Figure 3 (to be generated): Training progression showing output $Y(t)$, control $U(t)$, and error $e(t)$ at epochs 0, 500, 1000, and 2500 for Model 3 with $T_0 = 0$.

4) Combined Dead Time and Projection Performance: We will evaluate overall performance across all dead time scenarios with matched compensation.

Table 3 (to be populated): IAE comparison across dead time scenarios

TABLE IV
IAE VALUES FOR MATCHED COMPENSATION - *Results Pending*

Model	Controller	$T_0 = 0$	$T_0 = 2$	$T_0 = 4$
1	PI	–	–	–
1	Q-2500ep	–	–	–
3	PI	–	–	–
3	Q-2500ep	–	–	–

Expected trend: Dead time should degrade absolute IAE for both PI and Q-learning, but Q-learning should maintain relative advantage.

Table 4 (to be populated): Effect of compensation strategy on performance

TABLE V
IAE FOR DIFFERENT COMPENSATION STRATEGIES (MODEL 3) - *Results Pending*

T_0 [s]	**None** ($T_{0,c} = 0$)	**Under** ($T_{0,c} = T_0/2$)	**Matched** ($T_{0,c} = T_0$)
2	–	–	–
4	–	–	–

Key hypothesis to test: Even 50% undercompensation should provide substantially better performance than no compensation.

Figure 4 (to be generated): Performance comparison matrix showing IAE, overshoot, and settling time for $T_0 \in \{0, 2, 4\}$ s with matched compensation.

Figure 5 (to be generated): Load disturbance rejection comparison between PI and Q-after-learning for $T_0 = 2$ s, showing faster recovery time.

C. Summary of Validation Approach

Our validation strategy systematically evaluates:

1) **Bumpless Initialization:** Q-initial performance should match PI across all scenarios
2) **Learning Effectiveness:** Q-2500ep should outperform PI when dead time is properly compensated
3) **Dead Time Compensation:** Matched compensation ($T_{0,c} = T_0$) should yield best results
4) **Robustness:** Undercompensation ($T_{0,c} = T_0/2$) should still provide significant benefit over no compensation
5) **Projection Function:** Should enable bumpless switching despite $T_e = 5$ s vs. $T_I = 20$ s mismatch

The 14 experimental scenarios (7 per model) provide comprehensive coverage of the design space defined by process order, dead time magnitude, and compensation strategy. All performance metrics are computed from verification experiments run in pure exploitation mode ($\varepsilon = 0$) to eliminate the influence of exploratory actions.

[Actual numerical results, completed tables, and generated figures will be inserted here after experimental runs are completed.]

VI. DISCUSSION

A. Projection Function: Enabling Bumpless Initialization

The projection function serves a specific purpose in our Q2d controller architecture: it enables bumpless switching from an existing PI controller to the Q-learning controller even when the desired trajectory time constant T_e differs from the baseline integral time T_I. This capability is critical for industrial deployment scenarios where an existing PI controller with tuning K_{PI} and T_I is already operational.

Without the projection function, bumpless initialization would require $T_e = T_I$, which restricts the controller's initial operating point. While this constraint can be satisfied, it limits the designer's flexibility in choosing the target trajectory time constant based on performance requirements or process characteristics.

The projection term $\Delta U_{\text{proj}} = -e \cdot (1/T_e - 1/T_I)$ compensates for the structural difference between the Q2d controller's trajectory dynamics (based on T_e via the merged state definition) and the PI controller's implicit trajectory dynamics (based on T_I via the integral term). When $T_e = T_I$, the projection term vanishes, and the controller operates without this compensation. When $T_e \neq T_I$, the projection ensures that the initial control law matches the PI baseline behavior despite the time constant mismatch.

The implementation uses the continuous merged state value rather than the discretized action value for the projection calculation. This eliminates quantization errors that would otherwise accumulate, particularly when the projection magnitude is comparable to or larger than the Q-action magnitude. The resulting effective action $a_{\text{eff}} = s - \Delta U_{\text{proj}} = \dot{e} + e/T_I$ precisely matches the PI velocity form.

Additionally, the projection incorporates sign protection based on error magnitude to ensure stability near the setpoint. During transients (large errors), the projection is applied without restriction, allowing proper trajectory translation. Near steady state (small errors), sign protection prevents the projection from reversing the control direction, which could cause oscillations. This conditional application balances the need for accurate trajectory translation during setpoint changes against stability requirements near the operating point.

This design choice reflects a practical engineering consideration: industrial controllers must integrate seamlessly into existing control systems without causing transients or requiring plant shutdowns for commissioning. The projection function addresses this requirement by decoupling the choice of trajectory time constant from the need for bumpless initialization.

B. Dead Time Compensation Through Delayed Credit Assignment

The delayed credit assignment mechanism addresses a fundamental challenge in applying Q-learning to processes with dead time: ensuring that Q-value updates credit the correct state-action pairs. Our buffer-based approach provides an explicit solution to the temporal credit assignment problem by deferring Q-value updates until action consequences become observable in the measured process output.

The effectiveness of this approach depends critically on the accuracy of the dead time estimate used by the controller. When $T_{0,\text{controller}} = T_0$ (matched compensation), the buffers synchronize perfectly with the physical delay, ensuring that each action $a(t)$ is updated based on the state $s(t + T_0)$ that actually reflects its consequence. This temporal alignment is essential for correct Q-value propagation and policy learning.

The buffer size includes an additional sample beyond round$(T_{0,\text{controller}}/T_s)$ to account for the discrete-time update sequence. This ensures that the buffered state-action pair is paired with the state that actually reflects its consequence, rather than being off by one time step. Additionally, when updating Q-values for goal state-goal action pairs, we override

the bootstrap state to the goal state to prevent numerical drift from contaminating the value function. Over many buffer delays, small discretization errors can accumulate, causing the observed next state to be adjacent to (rather than exactly at) the goal. The override ensures that $Q(s_{\text{goal}}, a_{\text{goal}})$ maintains its maximum value despite these numerical effects.

The decoupled parameterization (T_0 for the physical plant, $T_{0,\text{controller}}$ for the controller compensation) enables systematic investigation of robustness to dead time estimation errors. Industrial processes often exhibit time-varying delays due to changing operating conditions, flow rates, or measurement configurations. The ability to maintain reasonable performance even when $T_{0,\text{controller}} \neq T_0$ would be valuable for practical deployment.

An important observation from our algorithm design is that even with no explicit compensation ($T_{0,\text{controller}} = 0$), Q-learning should eventually learn a policy that accounts for the delay implicitly through the state-action value function. However, we expect this implicit learning to require significantly more training episodes and potentially converge to suboptimal policies compared to explicit compensation. The experimental results will quantify this trade-off.

The sparse reward strategy ($R = 1$ for goal arrival or goal maintenance) interacts with dead time compensation in a subtle way. Because reward is assigned based on arrival at the goal state or maintaining the goal state with goal action, the buffering mechanism ensures that actions leading to goal state arrival receive positive reward, while actions leading away from the goal receive zero reward. This creates a value gradient in the Q-matrix that propagates backward through the state space via the bootstrap term $\gamma \max_{a'} Q(s', a')$. The dead time compensation ensures this propagation follows the correct temporal causality.

C. State Space Design and Discretization

The geometric action distribution employed in Q2d provides several practical advantages over alternative discretization schemes. By generating actions first using the geometric progression starting from $a_1 = 2 \cdot \text{prec}/T_e$ and then placing states at action midpoints, we achieve a natural density gradient: fine resolution near the goal state (where precise control is needed) and coarse resolution far from the goal (where aggressive correction is appropriate).

This density gradient emerges automatically from the geometric ratio calculation and does not require manual tuning of placement parameters. The single user-specified parameter—expected number of states—determines the geometric ratio q through the constraint that the largest action must reach the scaled control limit. The resulting distribution adapts to different process characteristics and control constraints without requiring problem-specific adjustments.

An important property of our discretization is the T_e-invariant error tolerance. Because the smallest state value is $s_1 = \text{prec}/T_e$ and the goal state spans $s \in (-s_1, s_1]$, the steady-state error tolerance remains $\pm$prec regardless of the trajectory time constant. This invariance would be particularly valuable in scenarios where T_e changes during operation (such as staged learning approaches), though in the present work T_e remains constant throughout training and verification.

The Q-matrix initialization as an identity matrix ensures that each state initially selects its corresponding diagonal action. Combined with the symmetric placement of states and actions around zero, this initialization provides stable baseline behavior before learning begins. The structure does not encode any problem-specific knowledge beyond the basic principle that states and actions should be similarly scaled.

D. Exploration Strategy and Learning Efficiency

The ε-greedy exploration strategy with constrained action selection balances the need for exploration (discovering potentially better policies) against the risk of destabilizing exploratory actions (causing large errors or control saturation). Our implementation enforces *same-side matching*: states above the goal state can only explore actions above the goal action, and vice versa. This directional constraint reduces the exploration space but prevents obviously counterproductive actions such as applying positive control increments when the error is already positive and growing.

The same-side matching constraint is essential for maintaining control stability during exploration. Without it, the ε-greedy policy could select actions that move the system away from the goal, creating oscillatory behavior or prolonged transients. By constraining exploration to actions on the "correct side" of the goal action, we ensure that even exploratory actions contribute to error reduction, albeit potentially at a suboptimal rate. This constraint is particularly important in the control context where actions have immediate physical consequences, unlike discrete decision problems where suboptimal actions may only delay reward accumulation.

The exploration rate $\varepsilon = 0.3$ during training represents a relatively high exploration level compared to typical Q-learning applications. This choice reflects the continuous nature of the control problem and the need to visit a wide range of state-action pairs to learn a robust policy. Industrial processes exhibit stochastic disturbances and varying operating conditions, so the learned policy must generalize beyond the specific conditions encountered during training.

A critical distinction in our methodology is the separation of training mode (with exploration) and verification mode (pure exploitation, $\varepsilon = 0$). During verification experiments, we disable exploration entirely to evaluate the learned policy without the influence of random exploratory actions. This separation ensures that performance metrics (IAE, overshoot, settling time) reflect the actual learned behavior rather than a mixture of learned policy and random exploration.

The episode termination criteria—stabilization (reaching and maintaining the goal state) or timeout (maximum iteration count)—provide natural boundaries for convergence monitoring. Episodes that terminate by stabilization indicate successful disturbance rejection, while timeouts during early training reflect incomplete learning. The continuous learning mode (buffers not reset between episodes) simulates realistic industrial operation where disturbances occur at arbitrary times without system resets.

E. Practical Tuning Guidelines

Deploying the Q2d controller in an industrial setting requires selecting several design parameters. We offer the following qualitative guidelines based on the theoretical properties of the algorithm and our implementation experience:

Trajectory Time Constant (T_e): Should be chosen based on desired closed-loop response speed. Faster response (smaller T_e) requires more aggressive control action and may be limited by actuator constraints or process dynamics. A reasonable starting point is to match the existing PI integral time ($T_e = T_I$) for bumpless initialization, though the projection function enables other choices.

Controller Gain (K_Q): Should match the existing PI proportional gain ($K_Q = K_{PI}$) for bumpless initialization. This ensures that the initial control effort magnitude is consistent with the baseline controller.

Precision Parameter (prec): Determines the steady-state error tolerance. Smaller values provide tighter regulation but require finer discretization and potentially longer learning times. Typical values of 0.5–1.0% of the setpoint range balance accuracy and computational efficiency.

Expected Number of States: Controls the discretization granularity. Larger values (e.g., 100–200) provide finer resolution but increase Q-matrix size and memory requirements. For PLC implementations, this represents a trade-off between control precision and available memory.

Learning Rate (α): Controls the speed of Q-value updates. Values of 0.1–0.2 provide reasonable convergence rates without excessive sensitivity to individual transitions. Too large values can cause oscillations in Q-values; too small values slow convergence.

Discount Factor (γ): Should be close to but less than 1 (typical values 0.95–0.99). Larger values emphasize long-term rewards and create stronger value gradients across the state space, but values too close to 1 can cause numerical issues.

Exploration Rate (ε): Should balance exploration and exploitation during training. Values of 0.2–0.4 enable sufficient exploration without excessive random actions. Should be set to 0 during verification and deployment for pure exploitation of the learned policy.

Training Duration: Depends on process dynamics, dead time magnitude, and desired convergence level. Monitoring the Q-value at the goal state ($Q(s_{\text{goal}}, a_{\text{goal}})$) provides a convergence indicator—values approaching $1/(1-\gamma)$ suggest near-optimal learning.

These guidelines provide starting points for parameter selection. The specific optimal values depend on process characteristics, control objectives, and practical constraints. Simulation studies with process models (when available) can help refine parameter choices before deployment.

F. Limitations and Assumptions

Our approach makes several assumptions that should be considered when evaluating applicability to specific industrial scenarios:

Single-Input Single-Output (SISO) Processes: The Q2d formulation addresses single-loop control problems. Extension to multivariable processes would require different state space formulations or coordinated single-loop controllers.

Constant Process Dynamics: We assume process dynamics remain sufficiently constant during learning. Processes with significant time-varying behavior or multiple operating regimes may require adaptive mechanisms or regime-specific Q-matrices.

Known Dead Time: The delayed credit assignment mechanism requires knowledge of the dead time magnitude. Processes with unknown or highly variable dead time may need dead time estimation methods or robust compensation strategies.

Linear Control Saturation: We assume hard limits on control signals $[U_{\min}, U_{\max}]$. More complex actuator dynamics (rate limits, hysteresis, deadband) would require additional modeling in the control algorithm.

Measurement Availability: We require sampled measurements of the process output at regular intervals. Processes with irregular sampling, missing data, or measurement delays may need modified update rules.

Disturbance-Based Learning: Our training protocol uses load disturbances to generate learning episodes. Processes where disturbances are rare or unacceptable during training may require alternative excitation strategies.

First-Order Trajectory Target: The merged state definition assumes a first-order exponential trajectory is appropriate. Processes requiring overshoot prevention or specific settling characteristics may benefit from different trajectory formulations.

Despite these limitations, many industrial regulatory control loops satisfy these assumptions. Processes such as flow control, level control, pressure control, and temperature control (with appropriate dead time compensation) often exhibit behavior compatible with our approach.

VII. Conclusions

We have presented a model-free Q-learning controller for industrial process control that addresses two critical challenges: bumpless initialization from existing PI controllers and robust operation in the presence of process dead time. The proposed Q2d controller integrates a projection function for initialization flexibility and a delayed credit assignment mechanism for dead time compensation, providing a practical framework for deploying reinforcement learning in industrial environments.

A. Summary of Contributions

Our work makes the following contributions to Q-learning-based process control:

1. Projection Function for Bumpless Initialization: We introduced a projection term $\Delta U_{\text{proj}} = -e \cdot (1/T_e - 1/T_I)$ that enables bumpless switching from PI control to Q-learning control even when the desired trajectory time constant T_e differs from the baseline integral time T_I. The implementation uses continuous state values for numerical precision and incorporates sign protection to ensure stability near the setpoint while allowing proper trajectory translation during transients. This capability decouples the choice of trajectory

time constant from the initialization requirement, providing designers with greater flexibility in specifying closed-loop performance objectives.

2. Delayed Credit Assignment for Dead Time: We developed a buffer-based Q-learning update mechanism that correctly assigns credit to state-action pairs in the presence of process dead time. By decoupling the physical plant dead time (T_0) from the controller's compensation strategy ($T_{0,\text{controller}}$), our approach enables systematic investigation of compensation accuracy requirements and robustness to dead time estimation errors. The buffer design includes temporal alignment correction (+1 sample) and bootstrap override for goal state maintenance, ensuring numerical stability over extended training periods.

3. Geometric State-Action Discretization: We formulated a geometric action distribution scheme where actions are generated first using a geometric progression and states are placed at action midpoints. This approach provides natural density gradients (fine resolution near the goal, coarse resolution far from goal) and maintains T_e-invariant steady-state error tolerance equal to the precision parameter.

4. Validation Framework: We designed a comprehensive experimental protocol spanning 14 scenarios across two process models, three dead time magnitudes, and three compensation strategies. The clear separation between training mode (with exploration) and verification mode (pure exploitation, $\varepsilon = 0$) ensures that performance metrics reflect learned policy quality rather than exploratory actions.

5. Same-Side Matching Exploration: We implemented a constrained exploration strategy that enforces directional consistency between states and actions. This prevents counterproductive exploration where actions would move the system away from the goal, ensuring stable learning even with relatively high exploration rates.

6. Sparse Reward Strategy: We demonstrated that a sparse reward function ($R = 1$ for goal arrival or goal maintenance, $R = 0$ otherwise) suffices for learning effective control policies. This simplicity eliminates the need for reward shaping or domain-specific reward engineering, relying instead on the bootstrap mechanism to propagate values throughout the state space.

B. Practical Implications

The Q2d controller with projection function and dead time compensation offers several advantages for industrial deployment:

Model-Free Operation: The controller requires no explicit process model, only the existing PI controller tunings (K_{PI}, T_I) and an estimate of the dead time ($T_{0,\text{controller}}$). This reduces commissioning complexity compared to model-based approaches that require system identification.

Bumpless Integration: The projection function enables seamless replacement of existing PI controllers without causing transients or requiring plant shutdowns. Initial performance matches the baseline PI controller, with gradual improvement through online learning.

Dead Time Robustness: The delayed credit assignment mechanism provides explicit compensation for process dead time, a common characteristic of industrial processes (e.g., material transport, sensor delays, communication latencies). The decoupled parameterization enables operation even with imperfect dead time knowledge.

Memory Efficiency: The Q2d state space merging reduces Q-matrix size from N^3 (Q3d formulation with separate error and derivative dimensions) to N^2 elements, enabling implementation on memory-constrained hardware such as programmable logic controllers (PLCs).

Interpretable Parameters: Key design parameters (T_e, prec, K_Q) have direct physical interpretations (trajectory time constant, steady-state accuracy, control gain), facilitating parameter selection by control engineers familiar with PI tuning.

C. Future Research Directions

Several extensions of this work warrant further investigation:

Adaptive Dead Time Estimation: The current approach assumes known (or estimated) dead time. Integrating online dead time estimation methods could enhance robustness to time-varying delays caused by changing flow rates, operating conditions, or measurement configurations.

Multivariable Extensions: Extending Q2d to multivariable processes (MIMO systems) represents a significant challenge due to interaction effects and higher-dimensional state spaces. Coordinated single-loop controllers or structured Q-matrix factorizations may provide tractable approaches.

Nonlinear Process Characteristics: While our validation includes a nonlinear pneumatic actuator model, systematic investigation of Q-learning performance on strongly nonlinear processes (e.g., pH control, batch reactors) would clarify applicability boundaries.

Safety-Constrained Learning: Industrial applications often impose strict safety constraints on process variables (temperature limits, pressure limits, concentration bounds). Incorporating hard constraints into the Q-learning framework through safe exploration strategies or constraint-aware reward functions would enhance industrial applicability.

Transfer Learning Across Operating Regimes: Many processes operate across multiple regimes with different dynamics. Investigating whether Q-matrices learned in one regime can accelerate learning in related regimes (transfer learning) could reduce commissioning time for multi-regime processes.

Staged Learning with Adaptive Time Constant: An alternative to the fixed projection function approach would be staged learning, where T_e starts at T_I (eliminating the projection requirement) and gradually decreases toward a target value as learning progresses. This approach would leverage the T_e-invariant precision property of our discretization, potentially enabling more effective Q-table refinement across the full state space rather than relying on projection compensation.

Comparison with Alternative RL Methods: Our work focuses on tabular Q-learning with discrete state-action spaces. Comparing performance, sample efficiency, and computational requirements against alternative reinforcement learning approaches (actor-critic methods, policy gradient methods, function approximation) would provide insight into algorithm selection trade-offs.

Real-World Pilot Studies: While simulation validation provides controlled experimental conditions, deployment on actual industrial processes would reveal practical challenges related to measurement noise, disturbance characteristics, operator interaction, and long-term reliability.

D. Concluding Remarks

The integration of model-free reinforcement learning into industrial process control represents a promising direction for addressing the persistent challenge of poor PID controller tuning in practice. Our Q2d controller with projection function and dead time compensation demonstrates that Q-learning can be adapted to satisfy the practical requirements of industrial deployment: bumpless initialization, dead time handling, memory efficiency, and interpretable parameters.

The experimental validation framework presented in this work provides a foundation for systematic evaluation of the proposed methods. The pending experimental results will quantify performance improvements, convergence characteristics, and robustness to dead time estimation errors across the 14 planned scenarios.

By building on established PI control structures and requiring only existing controller tunings as prior knowledge, the Q2d approach offers a path toward self-improving control systems that can be deployed by practicing control engineers without specialized expertise in machine learning or system identification. This accessibility is essential for addressing the widespread problem of poorly tuned industrial controllers at scale.

ACKNOWLEDGMENTS

This work was supported by [funding sources to be added]. The authors thank [collaborators/reviewers to be added] for valuable discussions and feedback.

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 Delayed Credit Assignment for Dead Time Compensation

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

if $T_{0,\text{controller}} > 0$ **then**
  {Delayed Credit Assignment Mode}

  {Step 1: Buffer current state-action pair}
  $(s_{\text{delayed}}, \mathcal{B}_s) \leftarrow \text{BufferPush}(s_t, \mathcal{B}_s)$
  $(a_{\text{delayed}}, \mathcal{B}_a) \leftarrow \text{BufferPush}(a_t, \mathcal{B}_a)$
  $(L_{\text{delayed}}, \mathcal{B}_L) \leftarrow \text{BufferPush}(L_t, \mathcal{B}_L)$

  {Step 2: Current state is "next state" for delayed action}

  $s_{\text{next}} \leftarrow s_t$

  {Step 3: Reward assignment (goal arrival OR goal maintenance)}
  if $s_{\text{next}} = i_{\text{goal}}$ **or** ($s_{\text{delayed}} = i_{\text{goal}}$ **and** $a_{\text{delayed}} = a_{\text{goal}}$) **then**
    $R \leftarrow 1$
  else
    $R \leftarrow 0$
  end if

  {Step 4: Bootstrap state with goal override for numerical stability}
  if $s_{\text{delayed}} = i_{\text{goal}}$ **and** $a_{\text{delayed}} = a_{\text{goal}}$ **then**
    $s_{\text{bootstrap}} \leftarrow i_{\text{goal}}$ {Override to prevent drift}
  else
    $s_{\text{bootstrap}} \leftarrow s_{\text{next}}$
  end if

  {Step 5: Q-learning update}
  if $L_{\text{delayed}} = 1$ **then**
    $Q(s_{\text{delayed}}, a_{\text{delayed}}) \leftarrow Q(s_{\text{delayed}}, a_{\text{delayed}})$
      $+\alpha \left[R + \gamma \max_{a'} Q(s_{\text{bootstrap}}, a') - Q(s_{\text{delayed}}, a_{\text{delayed}})\right]$
  end if
else
  {No Compensation Mode (One-Step Q-Learning)}

  {Step 1: Use previous iteration's state-action-reward}
  $s_{\text{prev}} \leftarrow s_{t-1}$; $a_{\text{prev}} \leftarrow a_{t-1}$
  $L_{\text{prev}} \leftarrow L_{t-1}$; $R_{\text{prev}} \leftarrow R_{t-1}$

  {Step 2: Current state is next state for previous pair}
  $s_{\text{next}} \leftarrow s_t$

  {Step 3: Q-learning update using previous reward}
  if $L_{\text{prev}} = 1$ **then**
    $Q(s_{\text{prev}}, a_{\text{prev}}) \leftarrow Q(s_{\text{prev}}, a_{\text{prev}})$
      $+\alpha \left[R_{\text{prev}} + \gamma \max_{a'} Q(s_{\text{next}}, a') - Q(s_{\text{prev}}, a_{\text{prev}})\right]$
  end if
end if