processes

MDPI

Article

Tuning of PID Controllers Using Reinforcement Learning for Nonlinear System Control

Gheorghe Bujgoi and Dorin Sendrescu *

Department of Automatic Control and Electronics, University of Craiova, 200585 Craiova, Romania; gheorghe.bujgoi@edu.ucv.ro
* Correspondence: dorin.sendrescu@edu.ucv.ro

Abstract: This paper presents the application of reinforcement learning algorithms in the tuning of PID controllers for the control of some classes of continuous nonlinear systems. Tuning the parameters of the PID controllers is performed with the help of the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, which presents a series of advantages compared to other similar methods from machine learning dedicated to continuous state and action spaces. The TD3 algorithm is an off-policy actor–critic-based method and is used as it does not require a system model. Double Q-learning, delayed policy updates and target policy smoothing make TD3 robust against overestimation, increase its stability, and improve its exploration. These enhancements make TD3 one of the state-of-the-art algorithms for continuous control tasks. The presented technique is applied for the control of a biotechnological system that has strongly nonlinear dynamics. The proposed tuning method is compared to the classical tuning methods of PID controllers. The performance of the tuning method based on the TD3 algorithm is demonstrated through a simulation, illustrating the effectiveness of the proposed methodology.

Keywords: learning-based control; nonlinear system control; PID controller; bioprocess

Academic Editor: Jie Zhang

Received: 31 January 2025
Revised: 24 February 2025
Accepted: 1 March 2025
Published: 3 March 2025

Citation: Bujgoi, G.; Sendrescu, D. Tuning of PID Controllers Using Reinforcement Learning for Nonlinear System Control. *Processes* **2025**, *13*, 735. https://doi.org/10.3390/pr13030735

1. Introduction

Control algorithms are essential to industrial automation and process control, ensuring efficient, safe, and reliable operations. These algorithms govern the behavior of machinery, systems, and processes in industries like manufacturing, biotechnology, energy, robotics, and automotives. Implementing these algorithms in hardware allows for the real-time, high-performance control of industrial systems. Hardware implementations of control algorithms in industrial applications are critical to achieving high-performance, reliable, and efficient operations. By leveraging microcontrollers, digital signal processors (DSPs), field-programmable gate arrays (FPGAs), programmable logic controllers (PLCs), and custom application-specific integrated circuits (ASICs), industries can ensure real-time responses, reduce downtime, and optimize their processes. As technology advances, the integration of AI, edge computing, and other innovations will continue to shape the future of control systems in industrial applications.

The numerical implementation of controllers allows the use of very complex algorithms with ease. However, in practice, due to its proven advantages, the proportional–integral–derivative (PID) controller (and its variants) is widely used in industrial control systems, as well as in many other applications that require continuous control. Most of the methods used to tune the parameters of PID controllers are based on time-invariant linear models of the processes, which, in practice, can lead to the poor performance of the

control system. Despite the existence of a wide range of advanced control methods, most industrial processes use classical PID-type control laws as control methods. This is due to the robustness of these control laws to disturbances, to modeling errors, or to the time variations of different parameters, but also due to the simplicity of implementation on both analog and digital devices.

The following are some specific examples of the use of PID controllers.

- In industrial processes, PID controllers are used to control the temperature, pressure, levels, and other important variables. They can help to keep these variables within safe and effective limits [1,2].
- In the aerospace industry, PID controllers are used to control the flight of aircraft. They can help to keep the aircraft straight and on course, even in difficult conditions [3].
- In the automotive industry, PID controllers are used to control the engine, transmission, and other systems. They can help to improve vehicles' performance, efficiency, and safety [4].

The tuning of classic PID controllers consists of setting the values of only three parameters—Kp, Ki, and Kd—corresponding to the three actions specified in the name of these controllers—proportional (P), integral (I), and derivative (D). Although, for linear systems, the tuning of a PID controller is straightforward, using various methods [5], the practical implementation raises numerous problems because real systems are nonlinear (or the linearity zone is very narrow) or have variable parameters. Moreover, some processes allow aggressive control, while others require smooth control. Thus, the selection of the parameters must be carried out according to the specific characteristics of the process, which is why, in industrial practice, various approaches have been developed for the tuning of the parameters of PID controllers [6]. The simplest tuning method is trial and error. In this method, the parameters of the controller are adjusted according to the response of the system to various test signals. It is a method that requires experienced personnel and can lead to equipment failure if not performed carefully. In addition, in this category of methods, we can include tuning using the Ziegler–Nichols frequency response method [7]. In this case, the closed-loop system is brought to the stability limit and the parameters are adjusted based on practical rules. Another category of methods is based on obtaining a simplified first-order plus time delay model (FOPTD) and using preset tuning formulas to adjust the aggressiveness and robustness of the response to various types of input signals.

While PID controllers are particularly effective for linear systems, they can also be applied to certain nonlinear systems, with some limitations. The challenge with nonlinear systems is that their behavior may vary across different operating points, and a fixed set of PID parameters may not provide optimal control across the entire range of system dynamics [8]. In some cases, nonlinearities can lead to instability or poor performance when using a standard PID controller.

However, there are several strategies to using PID controllers with nonlinear systems, among which are linearization [8], gain scheduling [9], and adaptive control [10].

Most of these methods assume good knowledge of the system model (which must be obtained through mathematical modeling or through identification techniques), with the performance of the control system being directly influenced by the quality of this model.

Linearization: One approach is to linearize the system around an operating point and design a PID controller for this linearized model. This can work well if the nonlinearities are relatively small within the operating range.

Gain Scheduling: Another method is gain scheduling, where different sets of PID parameters are used for different operating conditions or operating points in the system's state space. The controller parameters are adjusted based on the system's nonlinearities.

Adaptive Control: Adaptive control techniques can be employed to adjust the PID parameters online based on the changing characteristics of the nonlinear system. This requires a feedback mechanism to continuously update the controller parameters.

This paper presents a method based on machine learning to tune the parameters of PID controllers for the automatic control of some classes of nonlinear systems. The development of machine learning methods in the field of artificial intelligence has been extended to the field of process control. Thus, the most well-known algorithms in the field of reinforcement learning (the field of AI closest to automatic control) have been tested and implemented in applications of control systems [11,12]. In particular, the actor–critic methods in RL have been intensively used for system control and optimization problems [13–16].

The first applications of RL in control systems were dedicated to discrete systems, mainly because the states and actions were discrete and finite. RL methods for the control of discrete systems, such as Q-learning or SARSA, are designed to operate in this context and may use tables to store and update the values of states and actions [17]. RL methods for the control of continuous systems often involve a number of additional challenges, such as efficiently exploring continuous action spaces, managing trade-offs between exploration and exploitation, and ensuring the convergence and stability of the algorithms in this environment. An example of an off-policy RL method used in tuning controllers is adaptive dynamic programming (ADP), which is a class of techniques used to solve optimization and decision-making problems in uncertain environments. It combines elements of reinforcement learning, dynamic programming, and function approximation [18]. ADP techniques have been successfully used in the control of nonlinear systems using various control techniques: optimal [19] and adaptive control [20] or fuzzy control [21].

Twin Delayed DDPG (TD3) is an off-policy reinforcement learning algorithm designed for continuous action spaces. It is an extension of the Deep Deterministic Policy Gradient (DDPG) algorithm, with several modifications to enhance its stability and performance. TD3 was introduced by Scott Fujimoto et al. in their 2018 paper titled, "Addressing Function Approximation Error in Actor-Critic Methods" [22], and it has been successfully used in several control applications [13–16,23].

The main novelties presented in this article are the development and design of a structure for the tuning of PID controller parameters using the TD3 algorithm, the application of this method in a strongly nonlinear system in the field of biotechnological systems, and a comparison between classical PID controller tuning methods and the reinforcement learning-based technique.

In this study, the TD3 algorithm was used to tune the parameters of the PID controller from the control loop of a nonlinear system.

This paper is structured as follows: in Section 2, the TD3 algorithm is presented in detail; Section 3 presents the tuning of the PID controller parameters using the TD3 algorithm; Section 4 presents the classical approach of PID tuning for a nonlinear system; Section 5 presents the simulation results for the two approaches; and Section 6 is dedicated to the conclusions regarding the results obtained and possible future approaches.

2. Twin Delayed Deep Deterministic Policy Gradient (TD3) Algorithm

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an off-policy actor–critic reinforcement learning algorithm. TD3 is an advanced reinforcement learning (RL) algorithm specifically designed for continuous action spaces. It is a successor (an enhanced version) to the Deep Deterministic Policy Gradient (DDPG) algorithm, and it addresses some of the shortcomings of DDPG, such as overestimation bias and numerical instability [24]. Since TD3 is built on DDPG, with some modifications, we will present the DDPG algorithm first.

2.1. Basics of Reinforcement Learning Algorithms

Reinforcement learning (RL) is a type of machine learning where an agent learns to behave in an environment by trial and error. The agent receives rewards for taking actions that lead to desired outcomes and punishments for taking actions that lead to undesired outcomes. Over time, the agent learns to take actions that maximize its rewards. RL is a powerful technique that can be used to solve a wide variety of problems, including game playing, robotics, or finance [11].

RL is a challenging field of machine learning, but it is also one of the most promising. RL has the potential to solve problems that are beyond the reach of other machine learning techniques. Some of the key concepts in reinforcement learning are the following (see also Figure 1) [24].

- Agent: The agent is the entity that is learning to behave in an environment. The most common structure for the agent is composed of two elements: the critic and actor. The critic estimates the expected cumulative reward (value) associated with being in a certain state and following the policy defined by the actor. The actor is responsible for learning and deciding on the optimal policy—the mapping from states to actions. It is essentially the decision-maker or policy function.
- Environment: The environment is the world that the agent interacts with.
- State: The state is the current condition of the environment.
- Action: An action is something that the agent can do in the environment.
- Reward: A reward is a signal that indicates whether an action was good or bad.
- Policy: A policy is a rule that tells the agent which action to take in a given state.
- Value function: A value function is a measure of how beneficial it is to be in a given state.

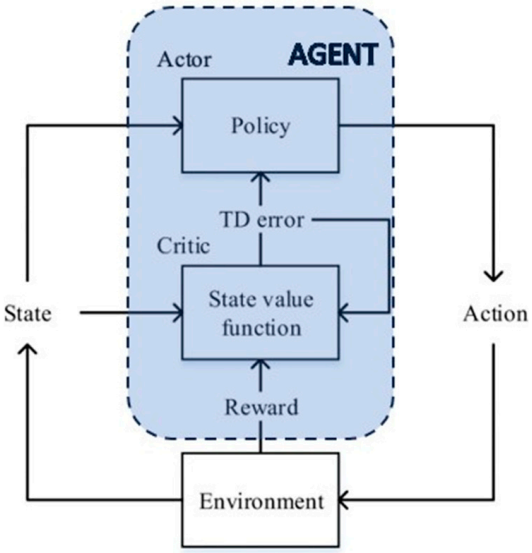

Figure 1. General scheme of reinforcement learning.

The basic idea of RL-type algorithms is to improve their policies, and, from this point of view, two main approaches have been developed: on-policy and off-policy learning. On-policy and off-policy are two categories of reinforcement learning algorithms that differ in how they use the collected data for learning. The key distinction lies in whether the learning policy (the policy being optimized) is the same as the policy used to generate the data [12]. In on-policy algorithms, the learning agent follows a specific policy while collecting data, and this policy is the one being improved over time (the data used for learning (experience) come from the same policy that is being updated). In off-policy algorithms, the learning agent has its own exploration policy for the collection of data, but

it learns and improves a different target policy (the data used for learning can come from a different (possibly older) policy than the one being updated). Representative examples are SARSA (State–Action–Reward–State–Action) for online learning and Q-learning for offline learning.

RL algorithms for continuous states have evolved significantly over the years, driven by the need to address real-world problems with continuous and high-dimensional state spaces. The early days of RL were dominated by discrete state and action spaces. Dynamic programming algorithms, such as the Bellman equation, were effective in solving problems with small, discrete state spaces. However, they were not suitable for continuous spaces due to the curse of dimensionality. To handle continuous states, researchers introduced function approximation techniques. Value function approximation using methods like tile coding and coarse coding helped to extend RL to continuous state spaces. This approach laid the foundation for the handling of larger and more complex state representations. Policy gradient methods emerged as an alternative to value-based approaches. Instead of estimating the value function, these methods directly learn a parameterized policy. Algorithms like REINFORCE and actor–critic methods became popular for problems with continuous state and action spaces. These algorithms are particularly effective when dealing with high-dimensional and complex problems. Deep Q-networks (DQN) extended RL to problems with high-dimensional state spaces, and later algorithms like Deep Deterministic Policy Gradient (DDPG) and Trust Region Policy Optimization (TRPO) addressed continuous action spaces. These methods leverage neural networks to approximate complex mappings from states to actions [24].

2.2. Deep Deterministic Policy Gradient

The main elements of the DDPG and TD3 algorithms are represented by two types of deep neural networks: the actor neural network and critic neural network. The general structure of these networks is shown in Figure 2 [25], where s—state, a—action and Q—Q-function (critic). The actor is associated with policy-based methods. It can learn a policy that maps states in the environment to actions, trying to find an optimal policy that maximizes the long-term rewards. The actor network receives as input the state of the environment and has as output the action to be applied in this state.

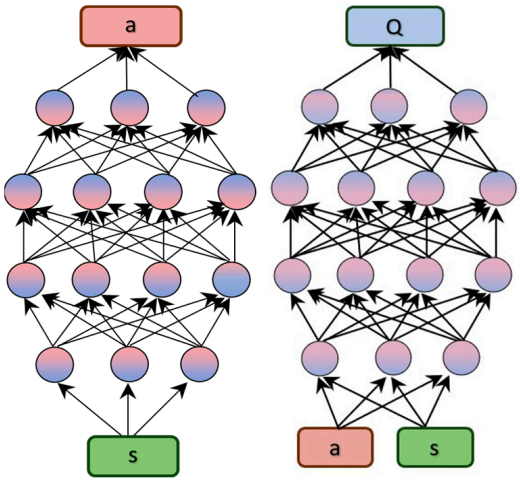

Figure 2. Actor and critic networks.

The critic estimates the value functions of the states or of the pairs (state, action) in the environment. The value function indicates the reward that one is expected to receive by starting from a certain state and taking a certain action. The critic evaluates the actions taken by the actor in a given environment and provides feedback regarding how well these

actions performed in achieving the desired goals or rewards. The most used technique in establishing the value function is Q-learning. In Q-learning, a function (usually denoted by Q) that associates state–action pairs with the expected value of the future reward is estimated using a deep neural network [26].

In the following, we shall use the usual notations from RL, presented in Table 1.

Table 1. Notations of main elements of DDPG and TD3 algorithms.

Notation	RL Element
s_k	current state
s_{k+1}	next state
a_k	current action
a_{k+1}	next action
r_k	reward at state s_k
Q	Q-function (critic)
π	policy function (actor)
δ	TD target
$\mathbb{A}$	action space
$\mathbb{S}$	state space

DDPG is an actor–critic algorithm that presents the advantages of both policy-based and value-based methods and learns optimal estimates for the both policy and value functions. Using the Bellman equation and off-policy data, the Q-function is learnt and then is used to learn the policy [22]. The main idea is the same as in Q-learning: if the optimal Q-function (denoted $Q^{opt}(s,a)$) is known, the optimal action is obtained by solving the following equation [24]:

$$a^{opt}(s) = arg\max_a Q^{opt}(s,a). \tag{1}$$

The starting point in learning an approximator for $Q^{opt}(s,a)$ is the well-known Bellman equation [12]:

$$Q^{opt}(s_k,a_k) = \mathbb{E}\left[r_k + \gamma \max_{a\in\mathbb{A}} Q^{opt}(s_{k+1},a)\right]. \tag{2}$$

Q-learning solves the Bellman optimality equation using the temporal difference (TD) [12]:

$$Q(s_k,a_k) \leftarrow Q(s_k,a_k) + \alpha\left[r_k + \gamma \max_{a\in\mathbb{A}} Q(s_{k+1},a) - Q(s_k,a_k)\right], \tag{3}$$

where

$\delta = \left[r_k + \gamma \max_{a\in\mathbb{A}} Q(s_{k+1},a) - Q(s_k,a_k)\right]$—TD error;

α—learning rate.

In order to use Q-learning for continuous state and action spaces, neural networks were used as function approximators for Q-values and function policies. So, considering that Q_w is the critic network parametrized by w and π_θ is the actor network parametrized by θ, relation (3) becomes [12]

$$Q_w(s_k,a_k) \leftarrow Q_w(s_k,a_k) + \alpha\left[r_k + \gamma \max_{a\in\mathbb{A}} Q_w(s_{k+1},a) - Q_w(s_k,a_k)\right]. \tag{4}$$

In relation (4), the term $\varphi_k = r_k + \gamma \max_{a\in\mathbb{A}} Q_w(s_{k+1},a)$ represents the target (the desired or goal value that the algorithm is trying to approximate or learn). The target φ_k depends on the parameter w that is to be optimized, so the target varies, and this causes significant difficulties for supervised learning. The solution to this problem in DDPG is to use a new

neural network (called the critic target network) that has a similar structure to the critic network but whose parameters do not change rapidly. Moreover, the target φ_k contains the maximization over $a \in \mathbb{A}$, which can be expensive since the Q-value function is a complex network with continuous inputs. To address this problem, a target actor network is used to approximate the optimal policy that maximizes $Q_w(s_{k+1}, a)$. The target networks are denoted in the following by

$\hat{Q}_{\hat{w}}$—target critic network parametrized by $\hat{w}$;
$\hat{\pi}_{\hat{\theta}}$—target actor network parametrized by $\hat{\theta}$.
So, the target becomes

$$\varphi_k = r_k + \gamma \cdot \hat{Q}_{\hat{w}}\left(s_{k+1}, \hat{\pi}_{\hat{\theta}}(s_{k+1})\right). \tag{5}$$

To train the critic Q_w, using a minibatch of N samples and a target critic $\hat{Q}_{\hat{w}}$, we compute

$$\varphi_i = r_i + \gamma \cdot \hat{Q}_{\hat{w}}\left(s_{i+1}, \hat{\pi}_{\hat{\theta}}(s_{i+1})\right), \tag{6}$$

and update w by minimizing the loss function L_C (calculated as the mean square error (MSE) between the current Q-value and the target Q-value):

$$L_c = \frac{1}{N}\sum_i (\varphi_i - Q_w(s_i,\ a_i))^2, \tag{7}$$

In order to update the actor network parameters, the policy gradient is used:

$$\nabla_\theta J = \frac{1}{N}\sum_i (\nabla_a Q_w(s_i,\ a_i) \cdot \nabla_\theta \pi_\theta(s_i)), \tag{8}$$

To update the target network parameters, DDPG performs "soft updates" using Polyak averaging:

$$\begin{array}{l} \hat{w} \leftarrow \tau \cdot w + (1-\tau) \cdot \hat{w} \\ \hat{\theta} \leftarrow \tau \cdot \theta + (1-\tau) \cdot \hat{\theta} \end{array}', \tag{9}$$

where $\tau \in [0, 1], \tau \ll 1$.

In summary, the main steps of the DDPG algorithm are as follows [12]:

1. Initialize the critic and actor networks;
2. Collect data from the environment and store them in the replay buffer;
3. Sample a batch of experiences from the replay buffer;
4. Compute the target Q-value using the target networks and update the critic using the mean-squared Bellman error loss;
5. Update the actor policy using the sampled batch, aiming to maximize the estimated Q-value;
6. Periodically update the target networks with a soft update;
7. Repeat steps 3–6 until the desired performance is achieved.

2.3. TD3—The Main Characteristics

The general structure of the TD3 algorithm is presented in Figure 3. It uses six neural networks, namely two critics, two critic targets, an actor, and a corresponding target. During each interaction with the environment, the agent collects experiences in the form of tuples (state, action, reward, next state). Instead of immediately using these experiences to update the policy or value function, the experiences are stored in the replay buffer. The replay buffer is a mechanism used in DDPG and TD3 to store and replay past experiences, promoting more stable and efficient learning in continuous action space reinforcement learning problems. A replay buffer is a key component used to store and sample experiences

from the agent's interactions with the environment. The replay buffer typically has a fixed size, and new experiences overwrite the oldest ones once the buffer is full. This ensures the continuous flow of new experiences while still benefiting from historical data. Training a neural network with sequential experiences can lead to strong correlations between consecutive samples, which may hinder learning. The replay buffer allows for the random sampling of experiences, breaking the temporal correlations and providing more diverse training data.

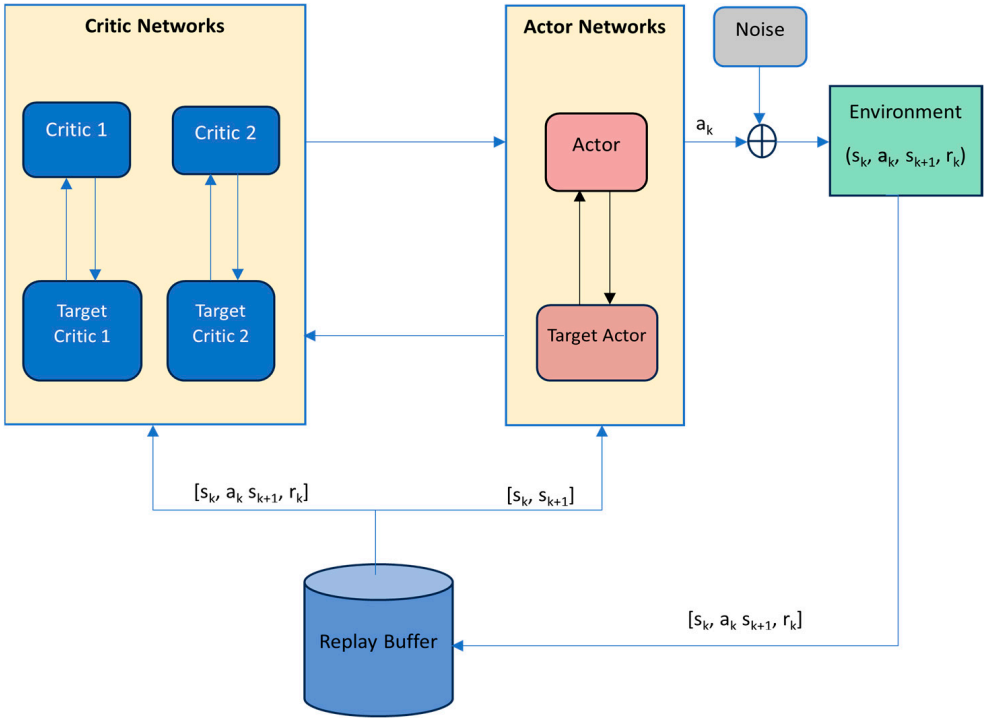

Figure 3. General structure of TD3 algorithm.

Overconfidence bias (a term that comes from psychology) defines a disparity between one's self-assessment of one's skills and abilities and the actual reality. This phenomenon extends beyond human behavior and is prevalent among RL agents, known in RL terminology as "overestimation bias". Another drawback of Q-learning algorithms is the possible numerical instability generated when using function approximation to estimate Q-values for continuous or large state–action spaces. Instabilities can arise when training neural networks, especially if they are deep. Issues like vanishing gradients or exploding gradients can affect the stability of the learning process. Consequently, the TD3 algorithm was developed to address these challenges within the actor–critic RL framework, specifically targeting the limitations observed in the DDPG algorithm. TD3 concentrates specifically on the actor–critic framework, implementing three techniques to enhance the DDPG algorithm: (1) clipped double Q-learning, (2) target policy smoothing, and (3) delayed policy and target updates.

Clipped double Q-learning is a means of reducing the overestimation bias in the Q-function. Double Q-learning addresses the overestimation problem by using two sets of Q-values (Q_1 and Q_2), and, during the updates, it uses one set to select the best action and the other set to evaluate this action [23]. Clipped double Q-learning moves a step further by introducing a clipping mechanism during the Q-value updates (as summarized in Figure 4). The idea is to prevent overestimation by limiting the impact of the maximum estimated Q-value.

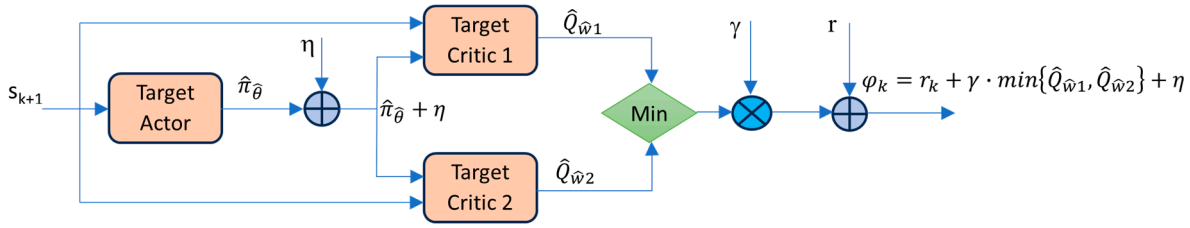

Figure 4. Flow of data through the target network to calculate the TD target using the clipped double Q-learning approach.

When updating the Q-values, the algorithm compares the Q-values from the two sets and chooses the smaller one. This clipped value is then used in the update rule. Since only a single actor π is used, a single TD target is then used to update both Q_1 and Q_2 [22].

$$\varphi_k = r_k + \gamma \cdot min\left\{\hat{Q}_{\hat{w}1}\left(s_{k+1}, \hat{\pi}_{\hat{\theta}}(s_{k+1})\right),\ \hat{Q}_{\hat{w}2}\left(s_{k+1}, \hat{\pi}_{\hat{\theta}}(s_{k+1})\right)\right\}. \tag{10}$$

Target policy smoothing is a means of making the updates to the policy less aggressive. This helps to prevent the policy from diverging too far from the current policy, which can lead to instability. In order to prevent this, a perturbation η (usually chosen as small Gaussian noise) is added to the action of the next state, so that the value evaluation is more accurate. Additionally, the noise itself, as well as the perturbed action, is clipped. The noise is clipped to ensure that it applies to only a small region around the action, while the perturbed action is clipped to ensure that it lies within the range of valid action values.

$$\varphi_k = r_k + \gamma \cdot min\left\{\hat{Q}_{\hat{w}1}\left(s_{k+1}, \hat{\pi}_{\hat{\theta}}(s_{k+1})\right),\ \hat{Q}_{\hat{w}2}\left(s_{k+1}, \hat{\pi}_{\hat{\theta}}(s_{k+1})\right)\right\} + \eta, \tag{11}$$

where $\eta \sim clip(N(0, \sigma), -c, c)$.

So, the update of w_1 and w_2 (critic parameters) is realized by minimizing two loss functions [22]:

$$\begin{aligned} L_{c1} &= \tfrac{1}{N}\textstyle\sum_i (\varphi_i - Q_{w1}(s_i,\ a_i))^2 \\ L_{c2} &= \tfrac{1}{N}\textstyle\sum_i (\varphi_i - Q_{w2}(s_i,\ a_i))^2, \end{aligned} \tag{12}$$

Delayed policy and target updates is a technique by which the target networks are updated less often than the main networks. In TD3, the policy update is delayed. Instead of updating the policy at every time step, the policy network is updated less frequently, typically after a fixed number of iterations or time steps. This delay helps in reducing the correlation between consecutive policy updates, leading to more stable learning and the better exploration of the action space. In addition to delaying the policy updates, TD3 also introduces delayed updates to the target networks (both the actor and the critic networks). In DDPG, the target networks are updated by directly copying the weights from the main networks periodically. However, in TD3, the target networks are updated less frequently than the main networks. Specifically, the target networks are updated less often than the policy network. This delayed updating of the target networks helps in stabilizing the learning process by providing more consistent target values for the temporal difference (TD) error calculation, thus reducing the variance in the update process. These two strategies, delayed policy updates and delayed target updates, work together to improve the stability and performance of the TD3 algorithm, making it more effective in training deep reinforcement learning agents for complex tasks. Similar to DDPG, we use the Polyak averaging technique with the formula below (τ has a very small value) [22]:

$$\begin{aligned} \hat{w}_1 &\leftarrow \tau \cdot w_1 + (1-\tau)\cdot \hat{w}_1, \\ \hat{w}_2 &\leftarrow \tau \cdot w_2 + (1-\tau)\cdot \hat{w}_2\ , \\ \hat{\theta}\ \ &\leftarrow \tau \cdot \theta + (1-\tau)\cdot \hat{\theta}\ . \end{aligned} \tag{13}$$

TD3 is a complex algorithm, but it is relatively easy to implement. Summarizing the above, the Algorithm 1 is as follows [22].

Algorithm 1. TD3

1. Initialize critic networks Q_{w1}, Q_{w2} and actor network π_θ with random parameters w_1, w_2, θ.

2. Initialize the parameters of the target networks:

$$\hat{w}_1 \leftarrow w_1, \hat{w}_2 \leftarrow w_2, \hat{\theta} \leftarrow \theta$$

3. Initialize replay buffer B

for k= 1 to T do

4. Select action with exploration noise $a \leftarrow \pi_\theta + \eta$ and observe reward r_k and new state s_{k+1}

5. Store transition (s_k, a_k, s_{k+1}, r_k) in B

6. Sample mini-batch of N transitions (s_k, a_k, s_{k+1}, r_k) from B and compute

$$\begin{aligned} &a_k \leftarrow \hat{\pi}_{\hat{\theta}}(s_{i+1}) + \eta \\ &\eta \leftarrow clip(N(0,\sigma), -c, c) \\ &\varphi_k \leftarrow r_k + \gamma \cdot min\{\hat{Q}_{\hat{w}1}, \hat{Q}_{\hat{w}2}\} + \eta \end{aligned} \tag{14}$$

7. Update critic parameters:

$$\begin{aligned} w_1 &\leftarrow arg \min_{w1} \tfrac{1}{N} \sum (\varphi_k - Q_{w1}(s_k, a_k))^2 \\ w_2 &\leftarrow arg \min_{w2} \tfrac{1}{N} \sum (\varphi_k - Q_{w2}(s_k, a_k))^2 \end{aligned} \tag{15}$$

8. Update θ by the deterministic policy gradient

$$\nabla_\theta J = \tfrac{1}{N} \sum_i (\nabla_a Q_{w1}(s_k, a_k) \cdot \nabla_\theta \pi_\theta(s_k)) \tag{16}$$

9. Update target networks:

$$\begin{aligned} \hat{w}_1 &\leftarrow \tau \cdot w_1 + (1-\tau) \cdot \hat{w}_1 \\ \hat{w}_2 &\leftarrow \tau \cdot w_2 + (1-\tau) \cdot \hat{w}_2 \\ \hat{\theta} &\leftarrow \tau \cdot \theta + (1-\tau) \cdot \hat{\theta} \end{aligned} \tag{17}$$

end for

3. Tunning of PID Controllers Using TD3 Algorithm

For PID tuning, TD3 is an excellent choice due to its increased stability, better control over exploration, and superior performance. Compared to DDPG, TD3 offers a higher level of precision and stability. Compared to TRPO, TD3 tends to be simpler to implement and has reduced complexity, making it suitable for PID tuning and other automatic control problems. Regarding its comparison to soft actor–critic (SAC) algorithms, SAC works with stochastic policies, outputting a probability distribution over the actions, while TD3 uses deterministic policies, directly outputting specific actions. Moreover, SAC uses entropy maximization for exploration, automatically balancing exploitation and exploration, while TD3 relies on adding noise to actions, which requires the manual tuning of the exploration parameters.

TD3 reduces the overestimation bias that is common in other reinforcement learning algorithms. TD3 introduces a delay in policy updates, which mitigates the risk of oscillations and instability during learning. The combination of reduced overestimation bias, delayed policy updates, and controlled exploration leads to a stable learning process in TD3. Stability is critical in PID tuning to ensure that the algorithm can converge to a suitable solution without causing fluctuations or inconsistencies in the control parameters. TD3's approach to mitigating overestimation bias and introducing controlled delays generally leads to higher performance and better convergence rates compared to other methods like DDPG.

In the following, we will consider a standard PID-type controller with the following input–output relation:

$$u = \begin{bmatrix} e & \int e dt & \frac{de}{dt} \end{bmatrix} \times \begin{bmatrix} Kp & Ki & Kd \end{bmatrix}^T. \quad (18)$$

Here,

- u is the output of the actor neural network;
- Kp, Ki, and Kd are the PID controller parameters;
- $e(t) = v(t) - y(t)$, where $e(t)$ is the system error, $y(t)$ is the system output, and $v(t)$ is the reference signal.

The proposed control structure with the tuning of the PID controller parameters using the TD3 algorithm is presented in Figure 5. Given the observations, a TD3 agent decides which action to take using an actor representation. The weights of the actor are, in fact, the gains of the PID controller, so one can model the PID controller as a neural network with one fully connected layer with the error, error integral, and error derivative as inputs (state of the system). The output of the actor neural network represents the command signal (action). The proposed scheme has two time scales: a scale for adapting the PID controller parameters (the weights of the actor) and a scale for analyzing the system's response to a step input.

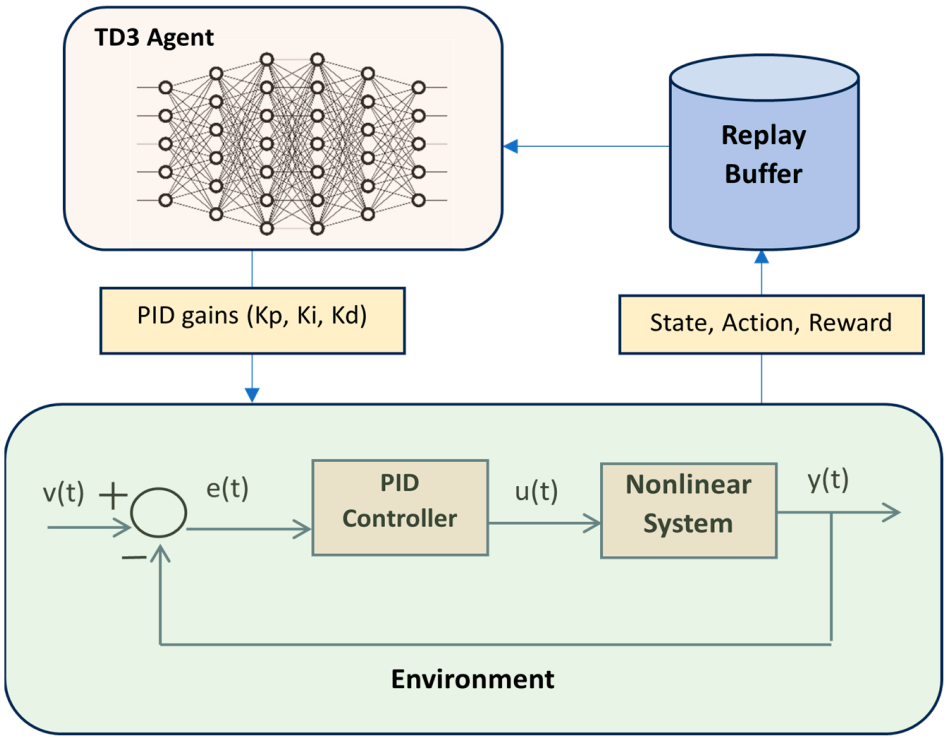

Figure 5. Illustration of TD3-based PID tuning approach.

The reward can be defined using point-based metrics (e.g., settling time, overshoot, etc.) or metrics based on the system trajectory. In our study, we use a reward function based on the LQG criterion (see Section 5).

4. Tuning PID Controller for Biotechnological System—Classical Approach

Designing a PID controller for biotechnological processes is a difficult task because, in general, these processes are very slow (usually, the time constants of the system are measured in hours); in many cases, they are unstable, which makes practical tuning

methods (for example, Ziegler–Nichols-type methods) impossible to apply. For these reasons, to design a PID controller for a biotechnological process, it is necessary first to obtain a linearized model, after which various tuning methods can be applied. We present below the procedure for obtaining a linearized model for a bacterial growth biosystem around an operating point and several tuning methods based on this linearized model.

4.1. Linearization of a Biotechnological System

The modeling of bioprocesses that take place in a fed-batch bioreactor is based on the general mass balance equations [27,28]. Starting from these equations, the general model is represented by a set of nonlinear differential equations of the following form:

$$\dot{\xi}(t) = f(\xi,\ D,\ F) = \Gamma \cdot \Phi(\xi, t) - D\xi(t) + F(t), \tag{19}$$

where

- $\xi(t) \in \Re^{n\times 1}$ represents the state vector (the concentrations of the system's variables);
- $\Phi = \begin{bmatrix} \mu_1 & \mu_2 & \cdots & \mu_m \end{bmatrix}^T$ denotes the vector of the reaction kinetics (the rates of the reactions);
- $\Gamma = \left[\gamma_{ij}\right], i = \overline{1,n}; j = \overline{1,m}$ is the matrix of the yield coefficients;
- $\Gamma \cdot \Phi(\xi, t)$ represents the rate of production;
- $-D\xi(t) + F(t)$ is the exchange between the bioreactor and the exterior.

This model is strongly nonlinear, a specific characteristic of most biotechnological processes. In order to design a PID controller, the system represented by relation (19) could be linearized around an equilibrium point $\left(\widetilde{\xi}, \widetilde{D}, \widetilde{F}\right)$ (which is a solution to the equation $f(\xi, D, F) = 0$). One obtains the following equations [27]:

$$\frac{d}{dt}\left(\xi - \widetilde{\xi}\right) = A\left(\widetilde{\xi}\right) \cdot \left(\xi - \widetilde{\xi}\right) - \left(D - \widetilde{D}\right)\widetilde{\xi} + \left(F - \widetilde{F}\right), \tag{20}$$

with

$$A\left(\widetilde{\xi}\right) = \Gamma \cdot \left[\frac{\partial \Phi(\xi)}{\partial \xi}\right]_{\xi=\widetilde{\xi}} - \widetilde{D} I_2, \tag{21}$$

where $\widetilde{\xi}$ represents the equilibrium value of ξ, and I_2 represents the second-order unity matrix. The linear model (20)–(21) can be used to tune the PID parameters using classical design formulas or computer-aided design approaches.

In the following, we present the model of the bacterial growth bioprocess that takes place in a continuous stirred-tank bioreactor. The dynamical model of the process can be expressed in the form of a set of differential equations as follows [27]:

$$\begin{array}{l} \frac{d\xi_B}{dt} = \mu(\xi_S) \cdot \xi_B - D \cdot \xi_B \\ \frac{d\xi_S}{dt} = -k_1 \cdot \mu(\xi_S) \cdot \xi_B - D \cdot \xi_S + F_{in} \end{array}, \tag{22}$$

where ξ_B is the biomass concentration, ξ_S is the substrate concentration, F_{in} is the input feed rate and $F_{in} = D \cdot S_{in}$ (where D denotes the dilution rate), and S_{in} is the concentration of the influent substrate.

The model can be written in the matrix form

$$\frac{d}{dt}\begin{bmatrix} \xi_B \\ \xi_S \end{bmatrix} = \begin{bmatrix} 1 \\ -k_1 \end{bmatrix} \cdot \mu(\xi_S) \cdot \xi_B - D \cdot \begin{bmatrix} \xi_B \\ \xi_S \end{bmatrix} + \begin{bmatrix} 0 \\ D \cdot S_{in} \end{bmatrix}, \tag{23}$$

and, if one denotes

$$\xi = \begin{bmatrix} \xi_B \\ \xi_S \end{bmatrix}; \Gamma = \begin{bmatrix} 1 \\ -k_1 \end{bmatrix}; \Phi(\xi) = \mu(\xi_S) \cdot \xi_B; \; F = \begin{bmatrix} 0 \\ D \cdot S_{in} \end{bmatrix}, \tag{24}$$

one obtains the form (19).

Because it takes into account the inhibitory effect of the substrate at high concentrations, one of the most used models for the specific growth rate is the Haldane model. The Haldane model is described by the following relation [29]:

$$\mu(\xi_S) = \mu_0 \cdot \frac{\xi_S}{K_M + \xi_S + \xi_S^2 / K_S} \tag{25}$$

where K_M represents the Michaelis–Menten constant, K_S is the inhibition constant, and μ_0 is the maximum specific growth rate.

Since

$$\left[\frac{\partial \Phi(\xi)}{\partial \xi_B} \right]_{\xi = \widetilde{\xi}} = \mu\left(\widetilde{\xi}_S\right) = \mu_0 \cdot \frac{\widetilde{\xi}_S}{K_M + \widetilde{\xi}_S + \widetilde{\xi}_S^2 / K_S}$$

and

$$\left[\frac{\partial \Phi(\xi)}{\partial \xi_S} \right]_{\xi = \widetilde{\xi}} = \left. \frac{d\mu(\xi_S)}{d\xi_S} \right|_{\xi = \widetilde{\xi}} \cdot \widetilde{\xi}_B = \mu_0 \cdot \widetilde{\xi}_B \frac{K_M - \widetilde{\xi}_S^2 / K_S}{(K_M + \widetilde{\xi}_S + \widetilde{\xi}_S^2 / K_S)^2} \triangleq \beta$$

the linearized matrix has the following form:

$$\mathrm{A} = \begin{bmatrix} \mu\left(\widetilde{\xi}_S\right) - \widetilde{D} & \beta \\ -k_1 \cdot \mu\left(\widetilde{\xi}_S\right) & -k_1 \cdot \beta - \widetilde{D} \end{bmatrix}. \tag{26}$$

For the bacterial growth bioprocess, it is important to control the substrate concentration since an overly high value for this concentration leads to the inhibition of biomass growth in the bioreactor [18]. So, one considers ξ_S as the controlled variable (output of the system) and the dilution rate D as input. Denoting $y = \xi_S - \widetilde{\xi}_S$ and $u = D - \widetilde{D}$, one obtains the standard state-space representation of a linear system:

$$\begin{cases} \frac{dx}{dt} = A \cdot x + B \cdot u \\ y = C \cdot x \end{cases}, \tag{27}$$

where

$$\mathrm{A} = \begin{bmatrix} \mu\left(\widetilde{\xi}_S\right) - \widetilde{D} & \beta \\ -k_1 \cdot \mu\left(\widetilde{\xi}_S\right) & -k_1 \cdot \beta - \widetilde{D} \end{bmatrix}, \; B = \begin{bmatrix} -\widetilde{\xi}_B \\ -\widetilde{\xi}_S + S_{in} \end{bmatrix}, \; C = [0 \; 1].$$

The linearized model (27) can be used to design a PID controller—for example, using the PID Tuner app from Matlab R2021b [19].

4.2. Ziegler–Nichols Method

The Ziegler–Nichols method is a popular heuristic approach for the tuning of PID controllers. It involves the following steps.

Step 1: Set *Ki* and *Kd* to zero, leaving only *Kp*.

Step 2: Increase *Kp* until the system reaches a critical oscillation (i.e., a steady-state oscillation with a constant amplitude).

Step 3: Record the value of Kp when the system starts oscillating, which is called the ultimate gain (Ku), and the period of the oscillation (Pu).

Step 4: Use the following formulas to calculate the PID parameters:

P controller: $Kp = 0.5 \times Ku$;

PI controller: $Kp = 0.45 \times Ku$, $Ki = 1.2 \times Kp/Pu$;

PID controller: $Kp = 0.6 \times Ku$, $Ki = 2 \times Kp/Pu$, $Kd = Kp \times Pu/8$.

This method is effective for systems with well-defined oscillation behavior but may not be suitable for highly sensitive systems or those with noise.

4.3. Pole Placement Method

The pole placement method is a classical control design technique used to place the poles of a system in specific locations in the complex plane, thereby determining the system's desired behavior (such as stability, damping, and natural frequency). In the context of second-order systems, this method allows us to control key performance characteristics such as overshoot, the settling time, and the damping ratio by appropriately selecting the poles of the system.

The PID controller parameters were tuned using the pole placement method for a second-order system described in [5]. If the process transfer function has the following form

$$P(s) = \frac{b_1 s + b_2}{s^2 + a_1 s + a_2}, \tag{28}$$

the closed-loop system is of the third order, and the characteristic polynomial is

$$s(s^2 + a_1 s + a_2) + (b_1 s + b_2)(K_D s^2 + K_P s + K_I), \tag{29}$$

Choosing the closed-loop characteristic equation of a third-order system is achieved as follows:

$$(s + \alpha\omega_0)(s^2 + 2\xi\omega_0 s + \omega_0^2), \tag{30}$$

with α, ω_0, ξ as the tuning parameters.

Equating coefficients of equal power in s in Equations (29) and (30), one obtains a system of three equations, from which one obtains K_p, K_i, K_d:

$$\begin{aligned} K_p &= \frac{a_2 b_2^2 - a_2 b_1 b_2(\alpha + 2\xi)\omega_0) - (b_2 - a_1 b_1)\left(b_2(1 + 2\alpha\xi)\omega_0^2 + \alpha b_1 \omega_0^3\right.}{b_2^3 - b_1 b_2^2(\alpha + 2\xi)\omega_0 + b_1^2 b_2(1 + 2\alpha\xi)\omega_0^2 - \alpha b_1^3 \omega_0^3} \\ K_i &= \frac{\left(-a_1 b_1 b_2 + a_2 b_1^2 + b_2^2\right)\alpha\omega_0^3}{b_2^3 - b_1 b_2^2(\alpha + 2\xi)\omega_0 + b_1^2 b_2(1 + 2\alpha\xi)\omega_0^2 - \alpha b_1^3 \omega_0^3} \\ K_d &= \frac{-a_1 b_2^2 + a_2 b_1 b_2 + b_2^2(\alpha + 2\xi)\omega_0 - b_1 b_2 \omega_0^2(1 + 2\alpha\xi) + b_1^2 \alpha\omega_0^3}{b_2^3 - b_1 b_2^2(\alpha + 2\xi)\omega_0 + b_1^2 b_2(1 + 2\alpha\xi)\omega_0^2 - \alpha b_1^3 \omega_0^3} \end{aligned} \tag{31}$$

5. Simulation Results

The tuning approaches presented in Sections 3 and 4 were implemented in the Matlab/Simulink environment. The Simulink implementation of the bacterial growth bioprocess is presented in Figure 6. The bioprocess parameters used in the simulations were taken from reference [29]:

$$\mu_0 = 6\,\mathrm{h}^{-1}, K_M = \frac{10\,\mathrm{g}}{l}, K_S = \frac{100\,\mathrm{g}}{l},\ k_1 = 1,\ S_{in} = \frac{100\,\mathrm{g}}{l}$$

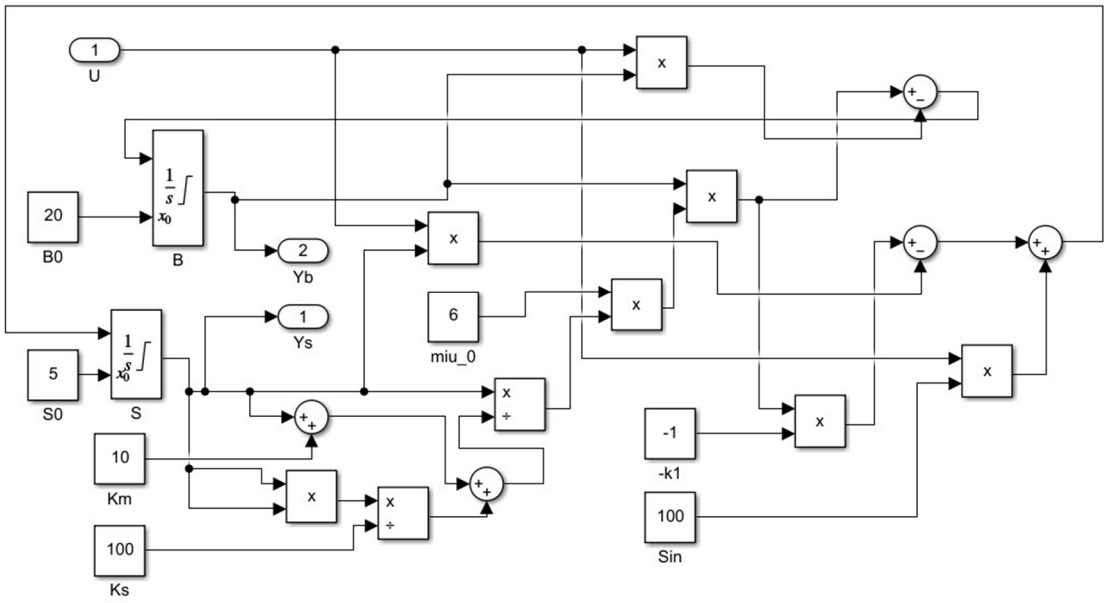

Figure 6. Matlab/Simulink implementation of bacterial growth bioprocess.

5.1. RL Approach

The tuning of the PID controller parameters using the TD3 algorithm is performed offline using a mathematical model of the controlled process. The Matlab implementation of the PID tuning structure shown in Figure 5 is presented in Figure 7 and is based on the RL-TD3 agent implemented in Matlab/Simulink [30].

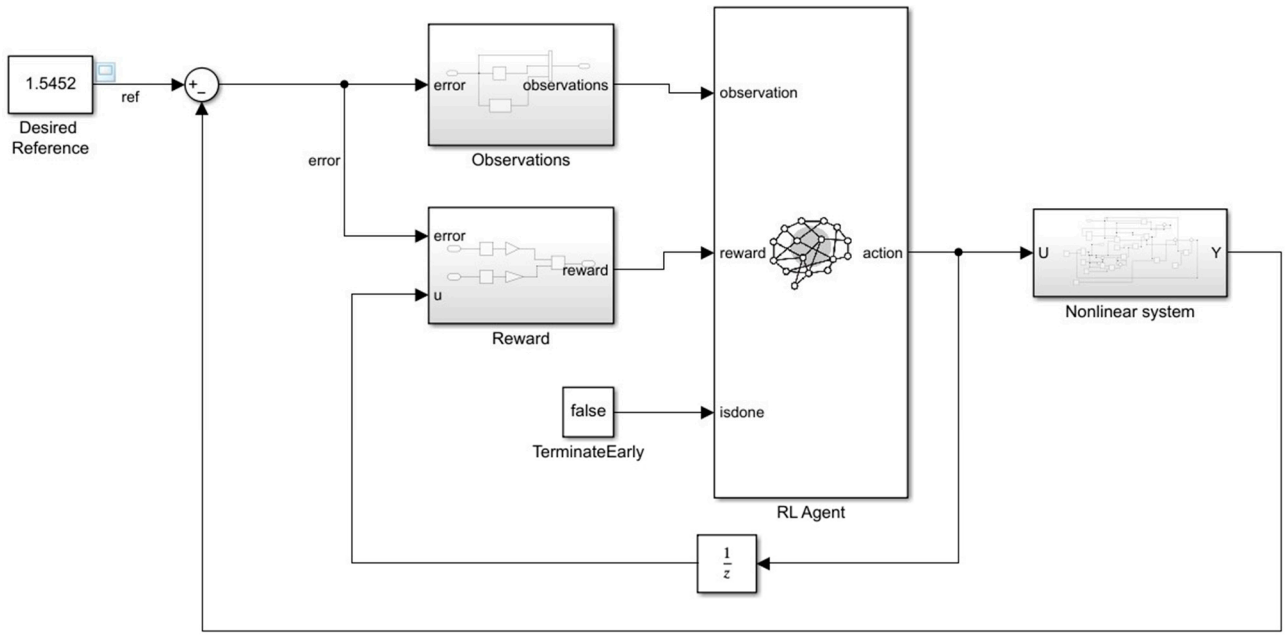

Figure 7. Matlab/Simulink implementation block diagram of the proposed control system using the RL-TD3 agent.

The observation block has the three elements $[P(t_n)I(t_n)D(t_n)]$ (see Figure 8), while the reward block implements the formula for the reward. We used a reward function based on the LQG criterion:

$$J = \lim_{T\to\infty}\left(\frac{1}{T}\int_0^T (a\cdot(ref-y)^2 + \mathrm{b}\cdot u^2(t))dt\right), \tag{32}$$

with a, b being two weighting coefficients (see Matlab implementation in Figure 9).

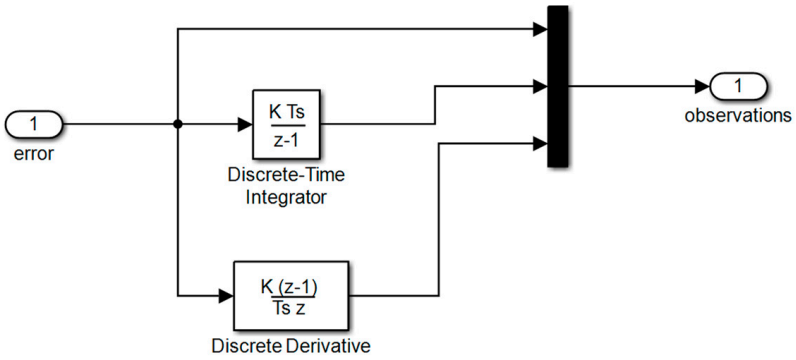

Figure 8. Matlab/Simulink implementation of observation vector.

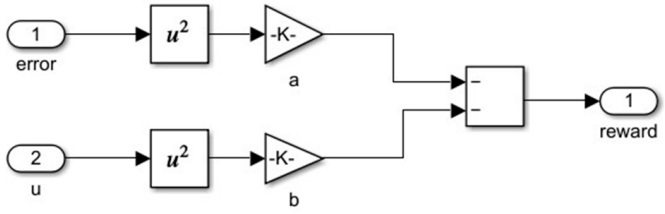

Figure 9. Matlab/Simulink implementation of reward function.

The parameter a corresponds to penalties on state deviations. Larger values of a indicate the stricter control of the corresponding states. It is chosen as the inverse of the square of the maximum acceptable deviation for each state. The b parameter penalizes control efforts. Higher values for this parameter lead to more conservative control actions.

The training parameters for the actor and critic networks are crucial in ensuring stable and efficient learning. They are often tuned through experimentation and hyperparameter optimization techniques like grid search or random search. The goal is to balance stability, the convergence speed, and the accuracy of the Q-value estimates. Below is a detailed breakdown of the key training parameters for both the actor and critic networks. The experience buffer (or replay buffer) is a critical component of TD3 and other off-policy reinforcement learning algorithms. It stores past experiences collected by the agent and allows them to be reused for training. Common sizes range from 10^5 to 10^6. The mini-batch size is the number of experiences sampled from the experience buffer for each training step. Common batch sizes range from 64 to 256. The discount factor (γ) is usually set between 0.95 and 0.99. Higher values (closer to 1) prioritize long-term rewards, while lower values focus on immediate rewards. The learning rates (for the actor and critic) ensure stable policy updates. Common values range from 10^{-2} to 10^{-4}. An episode in TD3 training is a complete sequence of interactions between the agent and the environment, from an initial state to termination. Episodes are used to collect experiences for training, evaluate the agent's performance, and facilitate exploration. The length of an episode depends on the environment and the agent's behavior, and multiple episodes are typically required to train the agent effectively. The Adam optimizer (short for Adaptive Moment Estimation) is a popular optimization algorithm used in training neural networks and other machine learning models. It combines the benefits of two other optimization techniques: the Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). Adam is known for its efficiency, robustness, and ability to handle sparse gradients, making it the default choice for many deep learning tasks, including reinforcement learning algorithms. The parameters that are utilized for the actor and critic networks of the TD3 agent are presented in Table 2.

Table 2. Training parameters for actor and critic networks.

Parameter	Value
Mini-batch size	128
Experience buffer length	500,000
Gaussian noise variance	0.1
Steps in episode	200
Maximum number of episodes	1000
Optimizer	Adam
Discount factor	0.97
Fully connected learning size	32
Critic learning rate	0.01
Reward function	a = 0.1, b = 0.02
Actor learning rate	0.01

The training process is presented in Figure 10.

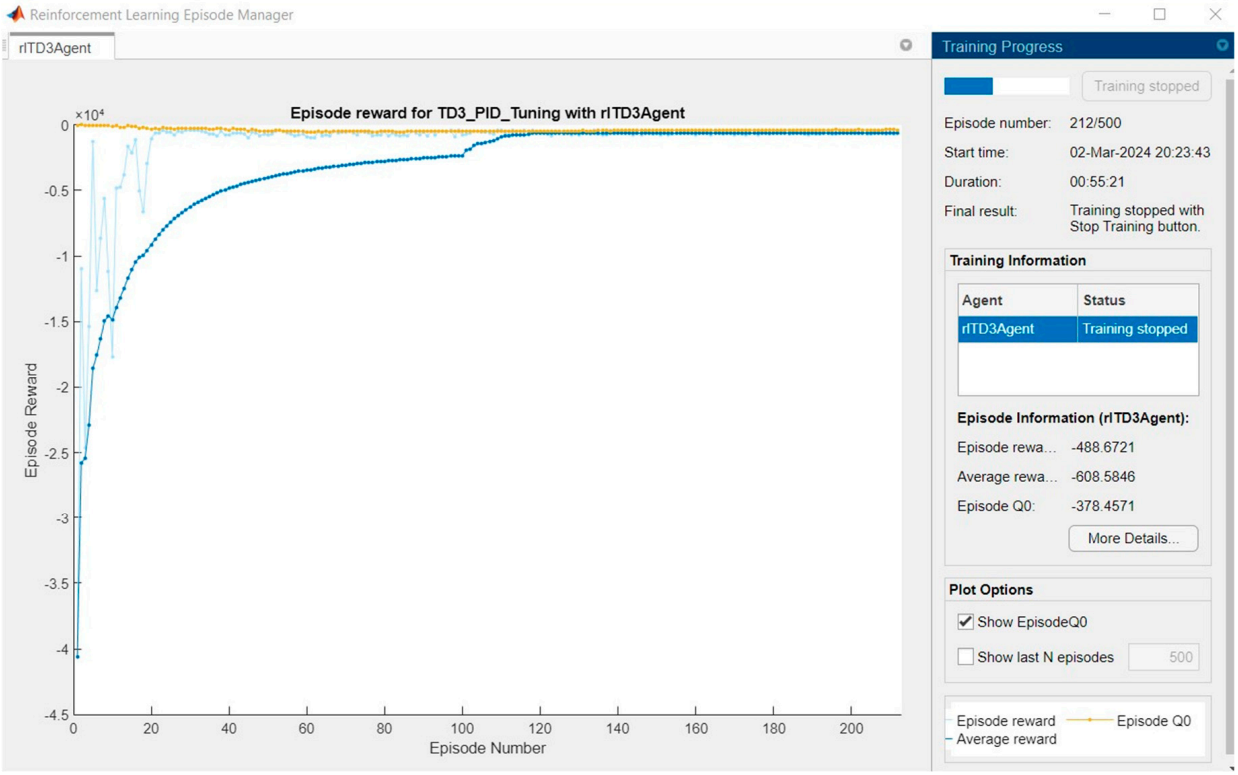

Figure 10. Training of TD3 neural networks.

It can be seen that the algorithm converges after approximately 100 episodes. The gains of the PID controller are the absolute weights of the actor representation. After 200 episodes, the following values of the PID controller were obtained:

$$K_P = 0.62; K_I = 3.57; K_D = 0.00023$$

With the gains obtained from the RL agent, a step response simulation was performed. The time evolution of the output of the system (substrate concentration) is presented in Figure 11, and the command signal and the biomass concentration are presented in Figure 12.

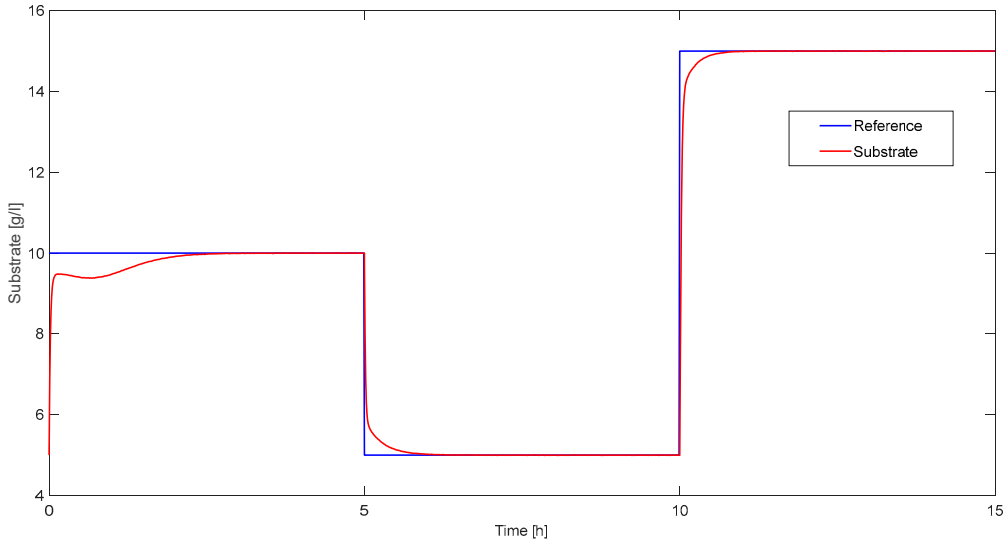

Figure 11. Step response of biotechnological system (RL approach).

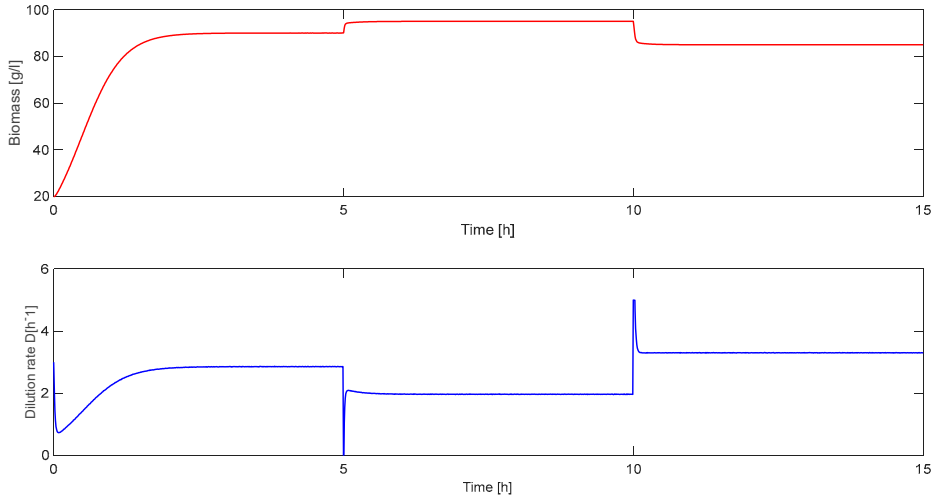

Figure 12. Time evolution of command signal and biomass concentration (RL approach).

5.2. Classical Approach

The nonlinear bioprocess was linearized around the equilibrium point $\widetilde{D} = 3.6\ \mathrm{h}^{-1}$, $\widetilde{\xi}_B = 80\ \mathrm{g/L}, \widetilde{\xi}_S = 23\ \mathrm{g/L}$. One obtains the linear model

$$\begin{cases} \frac{dx}{dt} = A \cdot x + B \cdot u \\ y = C \cdot x \end{cases}, \tag{33}$$

with

$$\mathrm{A} = \begin{bmatrix} 0 & 59.04 \\ -3.60 & -62.65 \end{bmatrix} B = \begin{bmatrix} -80 \\ 77 \end{bmatrix}, C = [0\ 1]$$

The PID controller parameters were tuned using the PID Tuner app from Matlab (see Figure 13). The values of the PID controller are

$$K_P = 0.207; K_I = 28.64; K_D = 0.00037$$

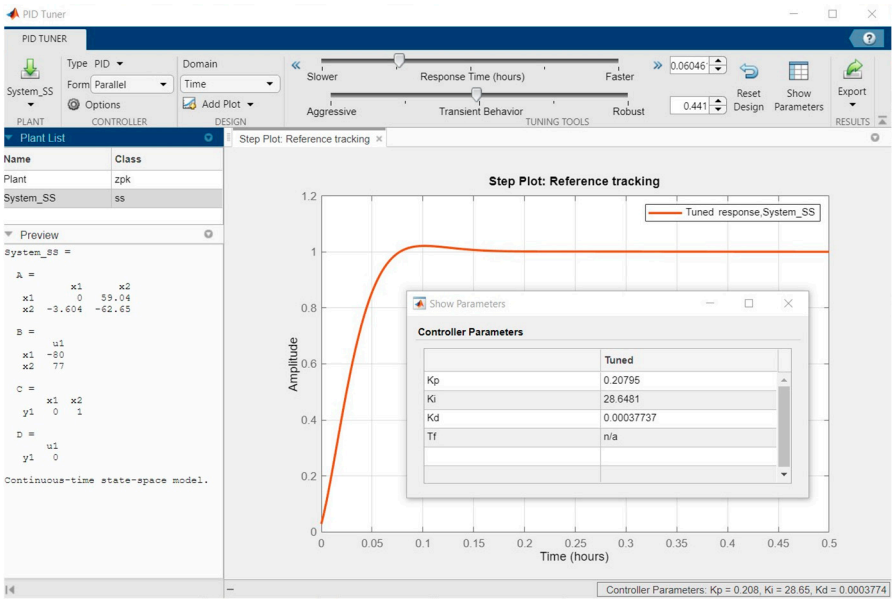

Figure 13. Tuning the controller using the PID Tuner app.

The time evolution of the output of the system (substrate concentration) is presented in Figure 14, and the command signal and the biomass concentration are presented in Figure 15.

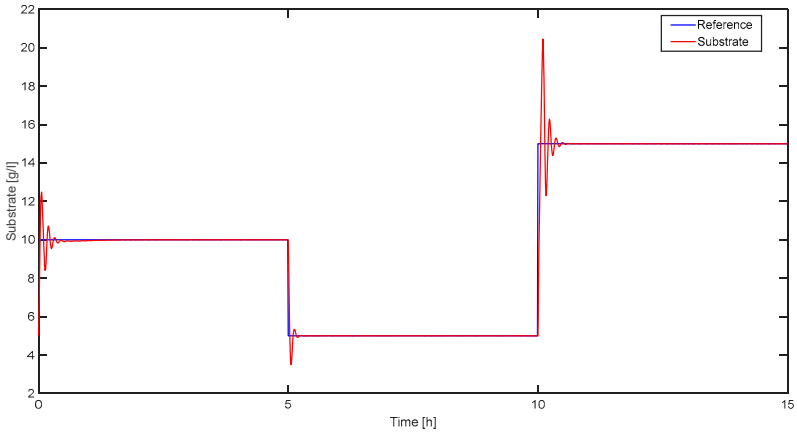

Figure 14. Step response of system output (PID Tuner app).

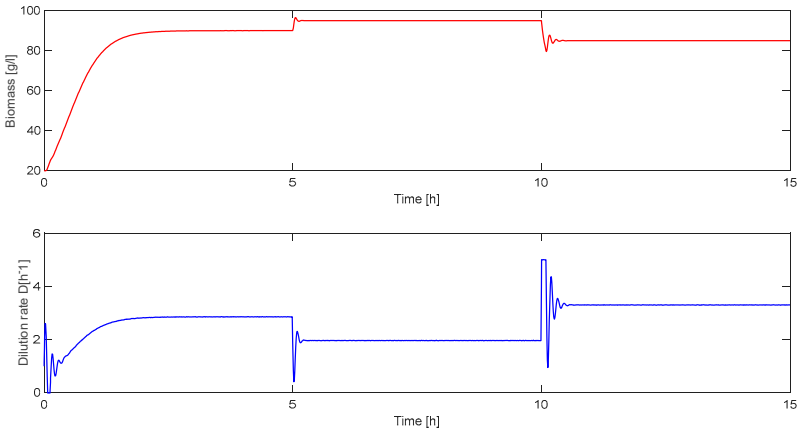

Figure 15. Time evolution of command signal and biomass concentration (PID Tuner app).

From Figures 12 and 15, it can be seen that the control of the substrate is very important (too high a value inhibits the concentration of biomass), with the main purpose of the regulation system being an increase in biomass production.

For the linear system (33), the two tuning methods presented in Sections 4.2 and 4.3 were also applied: the Ziegler–Nichols and pole placement methods. The main results and step response characteristics are summarized in Table 3.

Table 3. Tuned parameters and step response characteristics.

Tuning Method	Kp	Ki	Kd	Overshoot [%]	Settling Time [h]
TD3-based tuning	0.62	3.57	0.00023	0	0.8
PID Tuner app from Matlab	0.207	28.64	0.00037	20–40%	0.4–0.5
Ziegler–Nichols tuning	1.76	11.36	0.00054	35–45%	0.5–0.6
Pole placement tuning	1.32	14.27	0.00043	10–22%	0.5–0.55

The response obtained through the classical approaches is more aggressive, with a large overshoot and a slightly shorter response time. The response in the case of TD3-based tuning is slower and it has no overshoot.

6. Conclusions

In this paper, the use of reinforcement learning techniques for the tuning of PID controllers was proposed. The proposed method is very useful from a practical point of view because many industrial processes still use PID controllers, and their tuning is a particularly important step. The proposed tuning method is based on the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm and is applied to determine the parameters of PID controllers used in the control of continuous nonlinear systems. The RL-based tuning method is compared with classical techniques that are based on the linearization of the nonlinear system around an operating point. In the first case, the tuning of the controller is performed using the TD3 algorithm, which presents a series of advantages compared to other similar RL approaches dedicated to the control of continuous systems. This algorithm is an off-policy actor–critic-based method, and it was chosen as it does not require a system model and works in environments with continuous action and state spaces. In the classical approach, the nonlinear system was linearized around an equilibrium point and then standard tuning methods were used to determine the PID parameters. The presented techniques were applied to the control of a biotechnological system—a bacterial growth process—that took place in a fed-batch bioreactor. The simulations demonstrate the possibility of using machine learning algorithms to tune the parameters of PID controllers with the aim of controlling nonlinear systems.

Author Contributions: Conceptualization, G.B. and D.S.; methodology, D.S.; software, G.B.; validation, G.B. and D.S.; formal analysis, D.S.; investigation, G.B. and D.S.; resources, G.B. and D.S.; data curation, G.B.; writing—original draft preparation, D.S.; writing—review and editing, G.B.; visualization, G.B.; supervision, D.S; project administration, D.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest: The authors declare that they have no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

PID	Proportional–Integral–Derivative
TD3	Twin Delayed Deep Deterministic Policy Gradient
FOPTD	First-Order Plus Time Delay
RL	Reinforcement Learning
AI	Artificial Intelligence
SARSA	State–Action–Reward–State–Action
DQN	Deep Q-Network
DDPG	Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimization

References

1. Borase, R.P.; Maghade, D.; Sondkar, S.; Pawar, S. A review of PID control, tuning methods and applications. *Int. J. Dyn. Control.* **2021**, *9*, 818–827. [CrossRef]
2. Bucz, Š.; Kozáková, A. Advanced methods of PID controller tuning for specified performance. In *PID Control for Industrial Processes*; BoD–Books on Demand: Norderstedt, Germany, 2018; pp. 73–119.
3. Noordin, A.; Mohd Basri, M.A.; Mohamed, Z. Real-Time Implementation of an Adaptive PID Controller for the Quadrotor MAV Embedded Flight Control System. *Aerospace* **2023**, *10*, 59. [CrossRef]
4. Amanda Danielle, O.S.D.; André Felipe, O.A.D.; João Tiago, L.S.C.; Domingos, L.A.N.; Carlos Eduardo, T.D. PID Control for Electric Vehicles Subject to Control and Speed Signal Constraints. *J. Control. Sci. Eng.* **2018**, *2018*, 6259049. [CrossRef]
5. Aström, K.J.; Hägglund, T. *Advanced PID Control*; ISA-The Instrumentation, Systems, and Automation Society: Research Triangle Park, NC, USA, 2006; Volume 461.
6. Liu, G.; Daley, S. Optimal-tuning PID control for industrial systems. *Control Eng. Pract.* **2001**, *9*, 1185–1194. [CrossRef]
7. Aström, K.J.; Hägglund, T. Revisiting the Ziegler–Nichols step response method for PID control. *J. Process Control.* **2004**, *14*, 635–650. [CrossRef]
8. Lee, H.G. *Linearization of Nonlinear Control Systems*; Springer: Singapore, 2022.
9. Wilson, J.; Rugh, J.; Shamma, S. Research on gain scheduling. *Automatica* **2000**, *36*, 1401–1425. [CrossRef]
10. Ge, Z.; Liu, F.; Meng, L. Adaptive PID Control for Second Order Nonlinear Systems. In Proceedings of the Chinese Control And Decision Conference (CCDC), Hefei, China, 22–24 August 2020; pp. 2926–2931. [CrossRef]
11. Lillicrap, T.P.; Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:150902971.
12. Brunton, S.L.; Kutz, J.N. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*; Cambridge University Press: Cambridge, UK, 2019.
13. Shi, H.; Zhang, L.; Pan, D.; Wang, G. Deep Reinforcement Learning-Based Process Control in Biodiesel Production. *Processes* **2024**, *12*, 2885. [CrossRef]
14. Chowdhury, M.A.; Al-Wahaibi, S.S.S.; Lu, Q. Entropy-maximizing TD3-based reinforcement learning for adaptive PID control of dynamical systems. *Comput. Chem. Eng.* **2023**, *178*, 108393. [CrossRef]
15. Yin, F.; Yuan, X.; Ma, Z.; Xu, X. Vector Control of PMSM Using TD3 Reinforcement Learning Algorithm. *Algorithms* **2023**, *16*, 404. [CrossRef]
16. Liu, R.; Cui, Z.; Lian, Y.; Li, K.; Liao, C.; Su, X. AUV Adaptive PID Control Method Based on Deep Reinforcement Learning. In Proceedings of the China Automation Congress (CAC), Chongqing, China, 17–19 November 2023; pp. 2098–2103. [CrossRef]
17. Kofinas, P.; Dounis, A.I. Fuzzy Q-Learning Agent for Online Tuning of PID Controller for DC Motor Speed Control. *Algorithms* **2018**, *11*, 148. [CrossRef]
18. Massenio, P.R.; Naso, D.; Lewis, F.L.; Davoudi, A. Assistive Power Buffer Control via Adaptive Dynamic Programming. *IEEE Trans. Energy Convers.* **2020**, *35*, 1534–1546. [CrossRef]
19. Zhao, J. Adaptive Dynamic Programming and Optimal Control of Unknown Multiplayer Systems Based on Game Theory. *IEEE Access* **2022**, *10*, 77695–77706. [CrossRef]
20. El-Sousy, F.F.M.; Amin, M.M.; Al-Durra, A. Adaptive Optimal Tracking Control Via Actor-Critic-Identifier Based Adaptive Dynamic Programming for Permanent-Magnet Synchronous Motor Drive System. *IEEE Trans. Ind. Appl.* **2021**, *57*, 6577–6591. [CrossRef]

21. Massenio, P.R.; Rizzello, G.; Naso, D. Fuzzy Adaptive Dynamic Programming Minimum Energy Control of Dielectric Elastomer Actuators. In Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), New Orleans, LA, USA, 23–26 June 2019; pp. 1–6. [CrossRef]
22. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1587–1596.
23. Muktiadji, R.F.; Ramli, M.A.M.; Milyani, A.H. Twin-Delayed Deep Deterministic Policy Gradient Algorithm to Control a Boost Converter in a DC Microgrid. *Electronics* **2024**, *13*, 433. [CrossRef]
24. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
25. Yao, J.; Ge, Z. Path-Tracking Control Strategy of Unmanned Vehicle Based on DDPG Algorithm. *Sensors* **2022**, *22*, 7881. [CrossRef] [PubMed]
26. Hasselt, H.V.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-learning. In Proceedings of the AAAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30, pp. 1–7.
27. Rathore, A.S.; Mishra, S.; Nikita, S.; Priyanka, P. Bioprocess Control: Current Progress and Future Perspectives. *Life* **2021**, *11*, 557. [CrossRef] [PubMed]
28. Roman, M.; Selişteanu, D. Modeling of microbial growth bioprocesses: Equilibria and stability analysis. *Int. J. Biomath.* **2016**, *9*, 1650067. [CrossRef]
29. Sendrescu, D.; Petre, E.; Selisteanu, D. Nonlinear PID controller for a Bacterial Growth Bioprocess. In Proceedings of the 2017 18th International Carpathian Control Conference (ICCC), Sinaia, Romania, 28–31 May 2017; Institute of Electrical and Electronics Engineers (IEEE): Piscataway, NJ, USA, 2017; pp. 151–155.
30. MathWorks—Twin-Delayed Deep Deterministic Policy Gradient Reinforcement Learning Agent. Available online: https://www.mathworks.com/help/reinforcement-learning/ug/td3-agents.html (accessed on 20 February 2025).