Enhancing the Reliability of Control Systems Using an Improved Deep Reinforcement Learning Framework

Maryam Barekatain and Negin Sayyaf[*]

Department of Electrical Engineering, Faculty of Engineering, University of Isfahan, Isfahan, Iran.

Abstract

This paper presents an improved framework for deep reinforcement learning algorithms integrating online system identification, based on the Dyna-Q architecture. The proposed framework is designed to tackle the challenges of both Multi Input Multi Output and Multi Input Single Output systems in complex, industry relevant environments, thereby significantly enhancing adaptability and reliability in industrial control systems. It should be noted that in the suggested novel framework, the system identification and model control processes run in parallel with the control process, ensuring a reliable backup in case of faults or disruptions. To verify the efficiency of the aforementioned approach, comparative evaluations in the presence of three of the most common deep reinforcement learning algorithms, i.e. Deep Q Network, Deep Deterministic Policy Gradient, and Twin Delayed Deep Deterministic Policy Gradient, are conducted on industry-relevant environments simulations available in OpenAI Gym, including the Cart Pole, Pendulum, and Bipedal Walker, each chosen to reflect specific aspects of the novel framework. Results demonstrate that the proposed method for leveraging both real and simulated experiences in this framework improves sample efficiency, stability, and robustness.

Keywords: Deep Reinforcement Learning, Industrial Control Systems, System Stability, Model-Based Control, Intelligent Control Systems

1 Introduction

Control systems in industrial applications can be interpreted as the foundation for ensuring efficient and reliable operation. Traditional control methods, such as Proportional-Integral-Derivative (PID) controllers [1] and model-based approaches [2], have been the backbone of industrial automation for decades, offering stability and reliability. These methods are well-understood, relatively easy to implement, and provide predictable performance for stable and well-characterized processes. However, the growing complexity of industrial environments and the need for intelligent, adaptable control systems are pushing the boundaries of traditional approaches [3].

Traditional control methods excel in scenarios where system dynamics are well-defined and do not change significantly over time. They can provide robust performance in steady-state conditions and are typically straightforward to design and tune. However, they struggle in environments where system parameters vary, where there are unforeseen disturbances, or where the control objectives change frequently. Therefore, they often fall short in environments that demand intelligent adaptability and compatibility with dynamic conditions [4–6].

[*] Corresponding author's email: n.sayyaf@eng.ui.ac.ir

As industries evolve, the need for smart control systems that can learn and adapt to changes autonomously has become increasingly evident [7, 8]. Furthermore, according to the complexity, nonlinearity, and time-variance of most industrial systems, Reinforcement Learning (RL) offers a compelling alternative by providing a framework where controllers can learn directly from their environment. It allows the controller to learn through exploration and interaction to uncover optimal actions, which results in continuously improving performance for the control signal [9]. Also, RL controllers can adapt to system conditions by learning the ongoing dynamics, reducing the need for extensive reprogramming.

However, implementing RL in industrial applications presents challenges, because the initial knowledge base for RL controllers can be more complex than traditional methods. Additionally, ensuring stability during the learning process is crucial, as disruptions like sensor malfunctions or communication failures can arise unexpectedly. In industrial systems, sensor malfunctions can result in inaccurate or missing data, disrupting the controller's ability to make proper decisions, which may lead to suboptimal performance or unsafe conditions. Similarly, communication disruptions between sensors, controllers, or actuators can delay critical signals or feedback loops, increasing the risk of equipment damage, downtime, and process instability [10]. To partially overcome these challenges, model-based RL approaches provide certain advantages by incorporating an internal model of the environment, which can enhance both safety and efficiency compared to model-free methods [11].

The Dyna-Q algorithm, proposed by Sutton (1990) [12, 13], is a model-based RL approach that combines model-free learning with an internal model of the environment. In the Dyna-Q framework, the agent learns both a policy and a dynamics model of the environment. This allows the agent to generate simulated experiences, in addition to learning from real interactions, which can significantly improve sample efficiency. It is worth noting that Dyna-Q is not model-dependent (like the classic Model Predictive Control approach), but uses the model as a complementary part of the architecture
[14].

Dyna-Q has been successfully applied to a variety of control problems, demonstrating its ability to enhance the performance of model-free reinforcement learning algorithms. However, the initial Dyna-Q studies and many of its early adaptations utilized tabular approaches for both the planning and learning processes, which restricts its applicability to more complex, high-dimensional environments [15]. To address this limitation, researchers have explored the integration of Dyna-Q with neural networks (NN), leading to the development of the Deep Dyna-Q (DDQ) framework [16]. DDQ is proposed as a combination of Dyna-Q and deep learning methods, using NNs to model the state-action space.

Building upon the success of DDQ, several studies have explored the integration of DDQ with deep reinforcement learning (DRL) algorithms, such as deep Q-network (DQN). The study in [17] refers to this combination as Dyna-DQN. The authors aim to demonstrate the effectiveness of k-step rollouts in planning compared to single-step rollouts in achieving control outcomes. The authors also examine the planning shape effectiveness with perfect and imperfect models. The work in [18] presents an energy management system (EMS) architecture based on Dyna RL which integrates the concepts of the Dyna framework and the DQN algorithm. The paper's results highlight that Dyna-DQN outperforms both Q-Learning and DQN.

In addition, authors in [19] studied DDQ as model-based learning to improve learning speed and applied it to robot formation change. The results showed that DDQ can improve the number of episodes by about half compared to DQN. Also, the study in [20] investigates the Model-assisted Bootstrapped Deep Deterministic Policy Gradient (DDPG) algorithm in robotic environments, focusing on managing agent uncertainty to optimize artificial data usage in high-uncertainty scenarios. In [21], the authors extend DRL for vision-based navigation by using a

DDQ learning algorithm to enable a robot to navigate and evacuate environments with varying types and configurations of static and dynamic obstacles.

In short, prior works integrating Dyna-Q concepts into reinforcement learning frameworks have shown promising results across multiple environments. However, these approaches generally lack comprehensive benchmarking across widely recognized RL algorithms, limiting insight into their general applicability. Moreover, their architectures are often tailored to specific task settings and do not emphasize modular design, making them less adaptable to varied system configurations.

In parallel with these Dyna-Q-inspired approaches, recent model-based deep reinforcement learning methods such as DreamerV3 [22], PlaNet [23], and MBPO [24] have shown strong performance in visual and continuous control benchmarks by learning compact latent dynamics models. While these works focus on high-dimensional perception and end-to-end policy learning, the proposed framework in this paper emphasizes modularity and adaptability in industrial control settings with parallel system identification and control modules.

The main intuition of the paper originates from industrial control applications, where disruption, sensor malfunction, or disconnection from the controller can lead to major defects. These failures are critical, as industrial systems rely on the continuous transmission of essential data to generate accurate control signals and maintain optimal performance. Motivated by these real-world challenges, this paper introduces a novel solution: an RL-based control approach that operates independently, supported by an online system identification module. This combination addresses vulnerabilities caused by sensor failure or communication loss, while also maintaining a continuously updated model that serves as a reliable backup data generator, sustaining the control loop when the actual system cannot fulfill the data requirements.

In addition to enhancing reliability, the proposed approach can reduce the cost and risk associated with real-world exploration in reinforcement learning environments. The online model enables the generation of supplementary training data, decreasing the reliance on expensive or unsafe physical transitions during the RL training phase which is an important consideration in industrial settings.

Continuing the mentioned path, this paper presents a novel framework based on the Dyna-Q architecture, developed for DRL combined with online system identification, applied to Multiple Input Multiple Output (MIMO) and Multiple Input Single Output (MISO) systems. The analysis focuses on both continuous and discrete, industry-relevant environments, particularly those that reflect real-world industrial challenges and require robust, adaptive control in both simple and complex scenarios. The online system identification component continually updates the model of the system, providing a reliable backup in case of sensor failure or communication disruptions. This framework aims to enhance operational robustness for industrial control systems by reducing sensitivity to unforeseen challenges. Moreover, by learning from a simulated environment alongside real-world interactions, this approach reduces the number of real-world trials, thereby minimizing potential risks associated with exploration in industrial systems.

For comparative analysis, the proposed framework is firstly evaluated on basic and commonly used DRL algorithms: DQN and DDPG. These algorithms are primarily applied to simpler environments and MISO systems, where they have shown effectiveness in handling relatively straightforward control problems. However, for MIMO systems, which are naturally more complex, the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm is employed, which is known for its robustness and superior performance in continuous control tasks [25]. TD3 has emerged as one of the most reliable and fastest algorithms for MIMO systems, making it an ideal candidate for benchmarking this framework.

To thoroughly observe the performance of the proposed framework, the results of its integration with three DRL algorithms are provided using industry-relevant environments, including the inverted pendulum and robot walking simulations available in OpenAI Gym [26].

In summary, this paper offers two key contributions to the field of reinforcement learning:

- Introduction of a novel DRL-based control framework that integrates the online system identification to address critical industrial challenges such as sensor failure, data loss, and unsafe exploration (The framework ensures robust control by enabling the RL agent to operate independently while leveraging a continuously updated system model as a backup for control signal generation and training data augmentation.)
- Extensive practical study of the proposed framework in both MIMO and MISO systems under discrete and continuous control settings (The evaluation spans a range of industry-relevant tasks and systematically examines the interaction between model accuracy, control signal complexity, and DRL performance.)

The paper is organized as follows. The following section goes through the details of the utilized algorithms in this study, i.e. DQN, DDPG, TD3. Section 3 is devoted to a detailed explanation of the proposed framework. The results of the suggested framework application in three environments with benchmark DRL algorithms are also compared and discussed in Section 4. Finally, the paper is concluded in Section 5.

2 Algorithms

In this section, a thorough review of the experimental algorithms used in this paper is presented. First, the fundamental ideas and core concepts of RL will be explained.

Reinforcement Learning (RL) is a branch of machine learning, where an agent learns to make decisions by interacting with its environment through trial and error. The agent aims to maximize cumulative rewards over time, as a way of improving its decision-making policy. In other words, RL involves the agent performing actions and receiving rewards based on outcomes [27].

At its core, RL is about learning from interaction. The agent interacts with an environment defined by states and actions. The fundamental elements include:

- State (S): Represents the current situation of the environment.
- Action (A): The act selected by the agent based on the obtained policy, aiming to maximize future reward.
- Reward (R): Feedback from the environment based on the action taken.
- Policy (π): The agent's strategy for choosing actions in different states, which can be deterministic or stochastic.
- Value Function (V): The prediction of future rewards used to evaluate the goodness of a state.
- Q-Value (Q): Represents the value of a state-action pair.

The primary goal of RL is to discover a policy that maximizes the expected sum of rewards over time. RL excels in environments where the dynamics are unknown or partially known. The agent learns through exploration, trying different actions, and observing the results. This trial-and-error process enables the agent to develop a robust policy even in complex and dynamic settings, much like how humans learn from experience [28].

RL algorithms are typically categorized into two main types based on their approach to learning [29]:

- Value-Based Methods: These methods focus on estimating the value functions. The most fundamental value-based method is Q-learning [30], which directly learns the value of action-state pairs and uses these values to form a policy.

- Policy-Based Methods: These methods directly learn the policy that maps states to actions without learning value functions explicitly. Policy Gradient methods are a common example [31].

In addition, actor-critic methods combine both value-based and policy-based approaches. The actor updates the policy (policy-based), while the critic evaluates the action taken by the actor by estimating value functions (value-based). This dual approach leverages the strengths of both methods, resulting in more stable and efficient learning [32].

The evolution to DRL has seen the Q-learning algorithm significantly develop with the advent of NNs as function approximators [30, 33]. This development has led to DRL, which allows RL to scale to previously intractable problems, such as learning to play video games directly from pixels. In this study, DQN, DDPG and TD3, as the most widely used DRL algorithms, are used to evaluate the suggested framework. To this end, brief descriptions of these algorithms related to how they work, and their pros and cons can be observed in the following.

2.1 Deep Q-Network

DQN is a value-based algorithm that combines Q-learning with NNs to handle high-dimensional sensory inputs [34]. The essential idea is to use an NN to approximate the Q-function, $Q(s,a;\theta)$. Additionally, to break the temporal correlations between consecutive experiences and stabilize training, DQN uses an experience replay buffer. The agent stores past transitions (s_t, a_t, r_t, s_{t+1}) in a replay buffer and samples random mini-batches to train the network, where s_t, a_t and r_t respectively represent the state, action, and reward at the current timestep, and s_{t+1} denotes the next state in the environment. Finally, by setting a separate network for the calculation of the target Q-value, the learning process becomes more robust and stable. Indeed, the algorithm takes advantage of the Bellman equation in Eq. (1) for calculating target values [35]. Hence, by performing the gradient descent step on the loss function in Eq. (2), the Q-values updated until the Q-function are optimal.

$$y_t = r_t + \gamma \max Q'\left(s_{t+1}, a'; \theta\right) \tag{1}$$

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[y_i - Q\left(s_i, a_i; \theta\right)\right]^2 \tag{2}$$

It should be noted that in the aforementioned equations, γ is the discount factor that determines the importance of future rewards compared to immediate rewards. N represents the minibatch's size and θ represents the parameters of the NN. Q' and a' are also the target Q-network and the next action respectively. y_t is the estimated Q-value at timestep t.

2.2 Deep Deterministic Policy Gradient

DDPG is an actor-critic algorithm designed for environments with continuous state and action spaces. It maintains a deterministic policy $\mu(s|\theta^{\mu})$ and updates the actor and critic networks alternately [36]. To stabilize training, DDPG employs target networks for both the actor and critic, used to compute the target values for the Bellman equation. The critic and actor update equations are respectively defined in

$$L=\frac{1}{N}\sum_{i=1}^{N}(r_i+\gamma Q'(s_{i+1},\mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})-Q(s_i,a_i|\theta^{Q}))^2 \tag{3}$$

and

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_a Q(s_i,a_i|\theta^{Q})\nabla_{\theta^{\mu}}\mu(s_i|\theta^{\mu}) \ , \tag{4}$$

where θ^{μ} and θ^{Q} are the parameters of the actor and critic networks, while μ' and Q' represent the target critic and target actor networks respectively.

Since DDPG uses a deterministic policy, exploration during training is achieved by adding noise to the action selected by the actor network. Also to maintain more stability in learning, the target values are constrained to change slowly, using soft target updates in Eq. (5).

$$\begin{cases} \theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{cases} \tag{5}$$

In Eq. (5), τ represents the small update rate and $\theta^{\mu'}$ and $\theta^{Q'}$ denote the target critic and target actor networks' parameters respectively.

2.3 Twin Delayed Deep Deterministic Policy Gradient

TD3 improves upon DDPG by addressing the overestimation bias in the value function. To this end, the agent employs two critic networks, specified by parameter vectors θ_1' and θ_2' for estimating the Q-values, i.e. $Q_{\theta_1'}(s,a)$ and $Q_{\theta'2}(s,a)$, to provide a more cautious estimation of the Q-value [25]. In more details, the TD3 agent takes the minimum of these two Q-values to determine the target for the critic update, Eq. (6), during training. Because action selection is governed by the policy, the target value for updating the target critic is calculated as follows.

$$y_t = r_t + \gamma \min_{j=1,2} Q_{\theta_j'}\left(s_{t+1}, \mu_{\theta'}\left(s_{t+1}\right)\right) \tag{6}$$

Moreover, TD3 introduces target policy smoothing to further stabilize the training process. This technique adds a small amount of noise to the action selected by the target policy network, when computing the target Q-value. Also, TD3 delays the update of the actor (policy) network as for every *d* critic update, the actor network is updated once. This delayed update prevents the policy from changing too rapidly in response to potentially inaccurate Q-value estimates. During training, the critic is updated by minimizing the loss function, and the actor is updated using the deterministic policy gradient. Accordingly, the critic and actor update equations can be modified as follows:

$$\mathcal{L}(\theta_j) = \frac{1}{N}\sum_{i=1}^{N}\left[y_i - Q_{\theta_j}\left(s_i, a_i\right)\right]^2 \tag{7}$$

$$\nabla_{\theta^{\mu}} J = \frac{1}{N}\sum_{i=1}^{N}\nabla_a Q_{\theta_1}\left(s_i, a\right)|_{a=\mu(s_i)} \cdot \nabla_{\theta^{\mu}}\mu\left(s_i\right) \tag{8}$$

In the above equations the θ_j refers to the j-th critic network's parameters.

3 Proposed Framework

In advanced problems based on Reinforcement Learning algorithms, one prominent strategy for improving efficiency is the Dyna-Q approach. Dyna-Q represents an extension of the RL architecture which integrates learning from real experiences with simulated experiences drawn from a model of the environment. This model-based learning paradigm enhances data efficiency by supplementing direct interactions with the environment [12, 13]. Through the indirect RL phase, the agent does not interact with the real environment, but it updates the policy using simulated experiences generated by the internal model. However, the traditional Dyna-Q framework is primarily designed for tabular RL methods and struggles to extend to high-dimensional continuous spaces, particularly in MIMO settings [15]. To address these challenges, this paper proposes a novel framework, designed to handle a variety of RL tasks, particularly in high-dimensional continuous control applications.

The proposed framework, described in Fig. 1, is composed of three key processes: the control process, the system identification process, and the model control process. These components work in parallel to support both policy learning and environment dynamics modeling, providing a flexible and adaptive solution for various DRL tasks and real-world industrial applications.

It is worth noting that while this study evaluates the framework using DQN, DDPG, and TD3, these algorithms are selected solely for benchmarking purposes. The architecture of the framework is designed to be modular and algorithm-agnostic, allowing any RL or DRL algorithm to be integrated as the control agent. The control process, system identification process, and model control process are implemented as independent modules, making the framework broadly applicable across various RL paradigms.

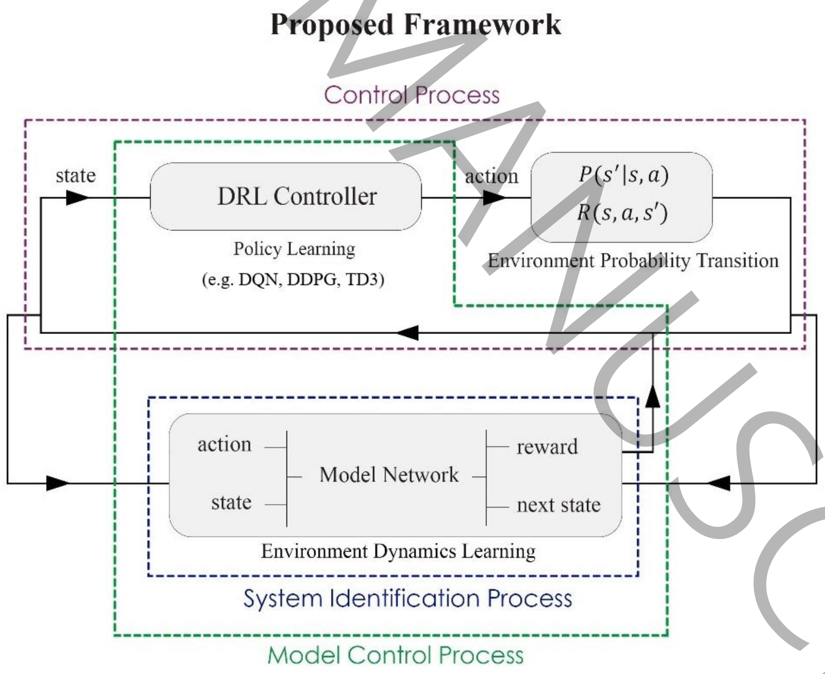

Fig. 1: Block diagram of the proposed framework
(Remark: the policy learning in DQN algorithm is done via Q-network, while the policy is learned via actor and critic networks in DDPG and TD3 algorithms)

Throughout this paper, three distinct processes are referred to: the control process, involving real-environment interactions; the model control process, in which data generated by the learned model is used; the system identification process, through which the model is continuously updated using collected transitions; and the policy learning, defined as the training of the RL agent using both real and model-generated transitions. In the following sections, each component of the framework, as represented in the block diagram, is explained in detail.

3.1 Control Process

The Control Process manages policy learning, enabling the agent to interact with the environment and optimize its decisions. As illustrated in figure 1, this process relies on neural networks, with two primary architectures: Q-Networks (used in DQN for value-based learning) and Actor-Critic Networks (used in DDPG and TD3 for continuous control tasks). These networks allow the agent to approximate the optimal policy by learning from the environment's feedback, which is obtained through probabilistic transitions represented as $P(s^{'} \mid s, a)$ and reward functions $R(s, a, s^{'})$.

This process supports both discrete and continuous action spaces and can be adapted to various RL problems by switching between value-based or actor-critic algorithms. The inclusion of algorithms such as DQN, DDPG, and TD3 ensures that the framework can be specialized for different environments, including MISO, and MIMO applications.

3.2 System Identification Process

To handle the complexity of high-dimensional continuous environments, the framework integrates a system identification process for learning the underlying dynamics of the environment. To implement the online system identification module, a feedforward neural network trained via supervised learning is used to approximate the system's transition dynamics and reward function [37]. The input to this model is a concatenated vector of the current environment state s_t and the action taken a_t, i.e. $[s_t, a_t]$, and the output consists of the predicted next state $\hat{s}_{t+1}$ and the corresponding reward $\hat{r}_t$. The neural network is trained using randomly sampled mini batches of transitions (s_t, a_t, s_{t+1}, r_t) collected online during interaction with the real environment. The loss function used for training is defined as the mean squared error (MSE) between the predicted and actual outcomes in Eq. (9) [38]. This supervised learning setup assumes the environment's dynamics are stationary and continuous over short time windows and the outputs are bounded, which allows the neural network to adaptively approximate the evolving system behavior.

$$L_{model} = \frac{1}{N}\sum_{i=1}^{N}\left(\left\|\hat{s}_i - s_i\right\|^2 + \left\|\hat{r}_i - r_i\right\|^2\right) \tag{9}$$

As depicted in Fig. 1, the aforementioned model network approximates the environment's transition dynamics, learning to map from an action state pair to a new state and reward prediction. By modeling these dynamics, the agent can simulate environment interactions and generate synthetic experiences, reducing reliance on real-world data.

This modeling process provides a key advantage in situations where data collection is costly or time-consuming. The ability to learn and predict the environment's behavior parallel to the other ongoing processes in the system allows for more efficient policy learning, enabling the agent to improve its performance without requiring extensive real-world interaction. This partnership also keeps the control process steady during disruptions and data reliability issues, taking the backup role for real interactions in the model control process as illustrated in the diagram of Fig. 1.

3.3 Model Control Process

The model control process facilitates seamless interaction between the control process and the system identification process. As shown in Fig. 1, it enables the agent to update its policy using both real and simulated experiences, thus bridging the gap between model-based and model-free learning. During the model episodes, the data used for policy learning is generated by the model that updates through system identification process. By continually refining the model of the environment, the agent can enhance the controller's performance through simulated experiences and rely on model feedback in case of disruption.

The combination of real and simulated experience, enabled by the model control process, ensures that the framework can be applied to a wide range of applications, particularly those involving high-dimensional continuous spaces such as robotic control, autonomous driving, and advanced MIMO systems.

3.4 Training Episode Scheduling Strategy

The online system identification process begins at the very start of training and continuously updates the model in parallel with the control and model control processes, using transitions collected directly from real-environment interactions. After the fixed episode threshold T_{switch}, i.e. a hyperparameter that should be tuned based on the complexity of the system, the framework begins to alternate between real control episodes and model control episodes. The selection is governed by a probability parameter $\beta \in [0,1]$, such that a real control episode is executed with probability β, and a model control episode is executed with probability $1-\beta$. This switching mechanism is designed to balance exploration in the real environment and exploitation of the learned model, and can be tuned based on environment complexity and model training performance. Hence, the episode type can be selected using a uniform random variable $z \sim u(0,1)$ as shown in Eq. (10)

$$\text{Scheduling strategy} = \begin{cases} E_R & if \;\; N_{episode} \leq T_{switch} \\ E_R & if \;\; N_{episode} \succ T_{switch} \quad , \quad z \prec \beta \\ E_M & if \;\; N_{episode} \succ T_{switch} \quad , \quad z \geq \beta \end{cases} , \tag{10}$$

where E_R and E_M respectively denote the real episodes during the control process and the model episodes during the model control process, and $N_{episode}$ refers to the number of episodes passed.

It is worth noting that while the current switching strategy between the control process and model control process is defined probabilistically for benchmarking purposes, in practical deployment, this mechanism can be adapted to respond directly to real-world conditions such as sensor failures or communication disruptions. Since the system identification module operates online and continuously from the beginning of training, the model remains up-to-date and can serve as a reliable fallback when such challenges arise.

The following section evaluates the performance of the framework across three industry-relevant environments in OpenAI Gym. The benefits and limitations of this framework will be discussed in the context of MIMO and MISO environments, considering both continuous and discrete action spaces.

4 Numerical Simulations

The results of applying the proposed framework based on three DRL algorithms are investigated in this section, focusing on how it performs in simple and complex environments.

In the following subsections, dedicated to the evaluation of DQN, DDPG, and TD3 algorithms respectively, first the general features of the simulation environments are detailed and then it's followed by the results and discussion of the Dyna-Q-based framework experiments.

To validate the defined approach, the collected reward by the agent through episodes, during both RL control and model control, and the loss of each environment's modeling process are provided. The increases and decreases in model loss curves indicate weak and strong models respectively. To evaluate the contribution of the model-generated data, performance in model control episodes is compared with that in real control episodes. Although the learning trajectories may differ, both converge to similar performance levels, indicating that the model successfully supports policy refinement. This demonstrates that the model control process can generate data of sufficient quality to guide the agent toward solving the environment, complementing real-environment interactions.

According to the statistical inherent of the data-driven algorithms, to evaluate the proposed framework in the presence of DQN, DDPG and TD3 algorithms in the following environments, the corresponding experiments have been runed via different seeds 10, 20, 30, 40 and 50. To ensure the high efficiency of the proposed novel approach, the graphs in each subsection have been plotted corresponding to the worst case, i.e. the latest convergence to the optimal response.

The role of hyperparameters in both the control signal learning and system modeling phases is also examined. Parameters such as learning rate, network architecture, batch size, and exploration noise were initially selected based on commonly reported values in the literature and subsequently refined through an iterative trial-and-error process, guided by the observed learning behavior and the complexity of the environment. It should be noted that in both simulation studies and practical applications, hyperparameter tuning must be performed individually for each plant or system, as the optimal configuration is highly dependent on the system's specific dynamics and task characteristics.

It is worth mentioning that both the policy learning and system identification modules are trained using first-order gradient-based methods, resulting in computational complexity that scales polynomially with the number of model parameters and input dimensions.

4.1 Deep Q-Network Results

The outcome of integrating the framework with DQN algorithm is evaluated on the Cart Pole environment from OpenAI Gym. The following subsection provides details about this environment.

4.1.1 Simulation Environment

Cart Pole is a part of the classic control environments in OpenAI Gym. In this setup, a pole is attached by an unactuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart, and the goal is to balance the pole by applying forces to push the cart left or right. Therefore, the action space is discrete with two values. The state space is a 4-dimensional array in continuous values of the cart's position and velocity, and the pole's angle and angular velocity.

The reward function for the mentioned experiment works relatively simply: 1 positive reward is given to each action made and the episode terminates if the pole angle is greater than ±12° or the cart position is greater than ±2.4. In each episode, the agent has 500 timesteps to interact with the environment. To solve the environment, the agent must achieve a score of around 500.

4.1.2 DQN Simulation Results

The best hyperparameters for RL control with DQN were determined through trial and error. The agent interacted with the environment for 500 timesteps, collecting rollouts, before the learning process began. The learning rate, gamma, and sample memory batch size are respectively set to 0.0003, 0.99, and 128. The exploration parameter, denoted as ϵ, begins at 1.0 and decays to 0.03 with a decay rate of 1000 episodes. The NN architecture used in the algorithm consists of three fully connected layers, each with 128 units.

As shown in Fig. 2 the agent interacts with the environment for 700 episodes, where the agent's performance converges around episode 530, reaching a score of 500. It is worth noting that the occasional drops in performance are attributed to the small exploration rate, which continues towards the end.

The next step involves incorporating the framework, which requires an online model of the environment. To construct this model, a two-layer NN is used to estimate the environment's dynamics based on experiences collected during training. The modeling process employs a learning rate of 0.001 and a memory sample batch size of 128. To assess the effectiveness of the modeling process, the model's loss is observed.

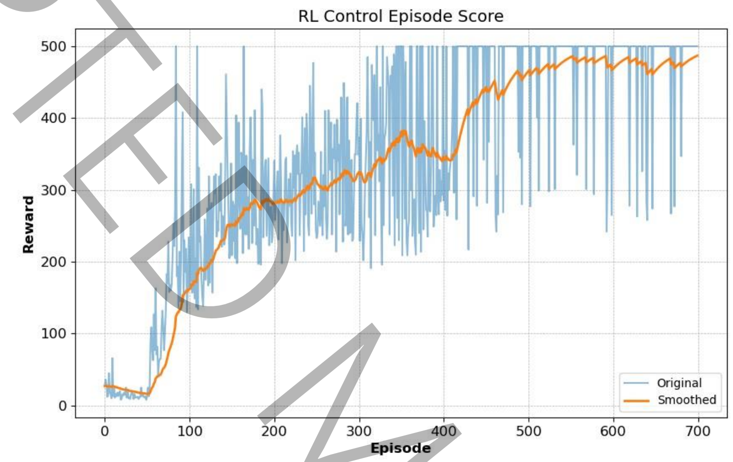

Fig. 2: The obtained reward during RL control episodes, in the presence of DQN algorithm

As shown in Fig. 3, the loss reaches the 10^{-6} range, indicating that the model has effectively learned to track the environment's dynamics. In the subsequent step, the proposed approach is utilized by incorporating the environment model into the control process. Approximately 20 percent of the control episodes are dedicated to the model control, using the model for the feedback signal to simulate interactions.

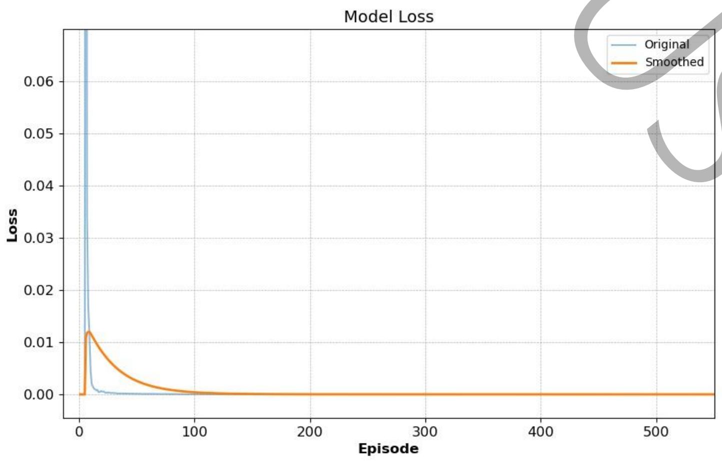

Fig. 3: The environment model loss using the suggested framework, in the presence of DQN algorithm

During this phase, shown in Fig. 4, the agent is given 150 timesteps to interact with the system's model. In the Cart Pole environment, the reward function is defined such that the agent receives a reward of +1 for every timestep it successfully balances the pole. The environment is considered "solved" or "done" when the agent reaches a total score of 500, which corresponds to the maximum of 500 timesteps without failure. Accordingly, in the model control episodes of this study, the episode length was set to 150 timesteps to evaluate whether the agent could approach a comparable level of performance using only model-generated experiences. It is anticipated that, with sufficient interaction, the agent will converge to a score of around 150 in these episodes, reflecting the effectiveness of the learned model in replicating the Cart Pole reward structure.

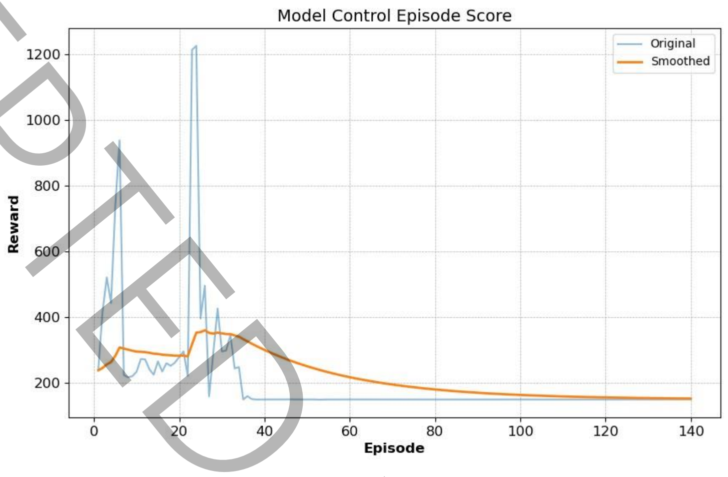

Fig. 4: The acquired reward during model control episodes, in the presence of DQN algorithm

4.2 Deep Deterministic Policy Gradient Results

In this section, the DDPG algorithm, which employs a more complex actor-critic structure, is integrated with the framework to experiment with a more challenging version of the Cart Pole environment: The Pendulum.

4.2.1 Simulation Environment

The Pendulum environment consists of a pendulum fixed at one end to a stationary point, with the other end free to move. The objective is to apply torque to the free end to swing the pendulum into an upright position. In this case, the state space is continuous with 3 dimensions, consisting of the pendulum's position and angular velocity.

Also, the action space is continuous, defined by the torque applied to the free end of the pendulum, ranging from −2 to +2. Moreover, the reward function can be found in Eq. (11).

$$r = -\left(\theta^2 + 0.1\omega^2 + 0.001\tau^2\right) \tag{11}$$

4.2.2 DDPG Simulation Results

As the complexity of the algorithm and environment increases, it becomes more challenging to obtain effective control signals and accurate models during training. Consequently, the significance of hyperparameters grows with more advanced algorithms. To illustrate the impact of hyperparameters on performance, two cases of RL control are presented in Figs. 5 and 6. Though both figures use a three-layer neural network with dimensions of 256 × 256 and a batch size of 256, the Fig. 5 shows the unstable and unresolved learning results, where the learning rate, γ, and τ are set to 0.0003, 0.99, and 0.005, respectively. Additionally, the noise standard deviation is set to 1, and the policy is updated at every timestep after the first 2000 timesteps. Whereas Fig. 6 demonstrates the outcome of proper hyperparameter tuning. Here, the learning rate, γ, and τ are adjusted to 0.0001, 0.98, and 0.02, respectively, and the policy update frequency is set to 50. Furthermore, the exploration noise standard deviation decreases gradually from 0.2 to 0.05 during the first 200 episodes of the process.

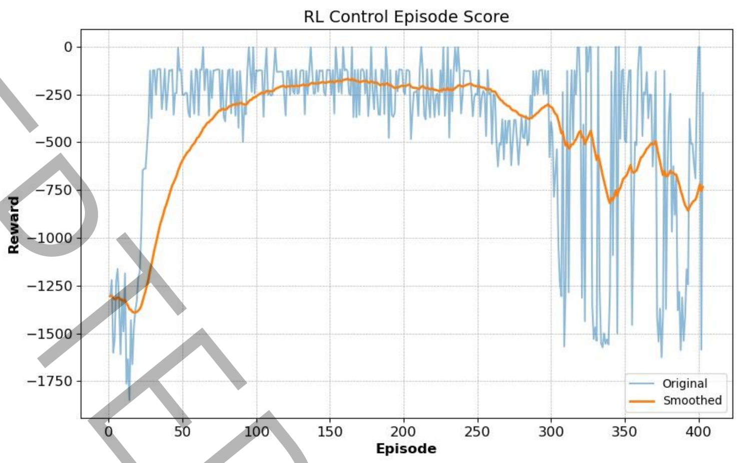

Fig. 5: The obtained reward during RL control episodes using the DDPG algorithm with untuned hyperparameters

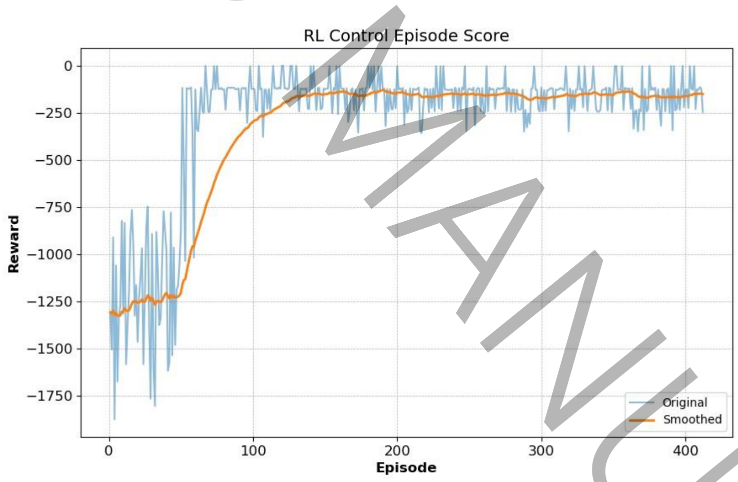

Fig. 6: The obtained reward during RL control episodes using the DDPG algorithm with tuned hyperparameters

To proceed with the first step of the provided approach, which involves model learning, it is crucial to consider the role of hyperparameters in model design. As illustrated in Figs. 7 and 8, the importance of hyperparameter tuning increases with the complexity of the environment. Fig. 7 shows that a model with a two-layer NN (128 × 128) results in suboptimal performance. In contrast, the desired model performance, achieved with a three-layer NN (256 × 256), is demonstrated in Fig. 8.

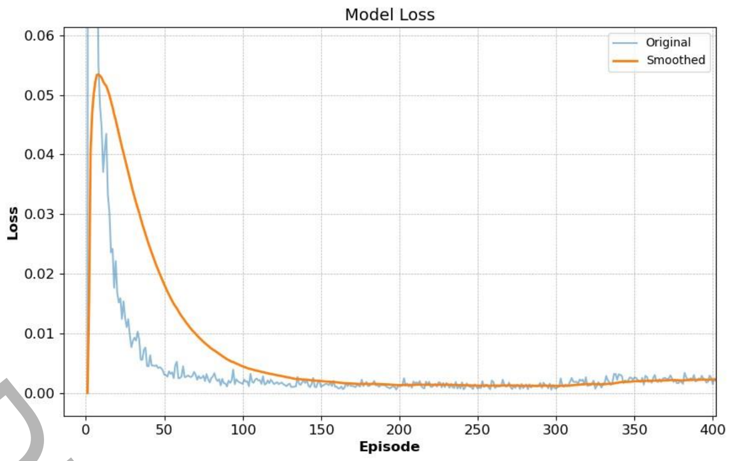

Fig. 7: The environment model loss using the suggested framework, in the presence of DDPG algorithm with untuned hyperparameters

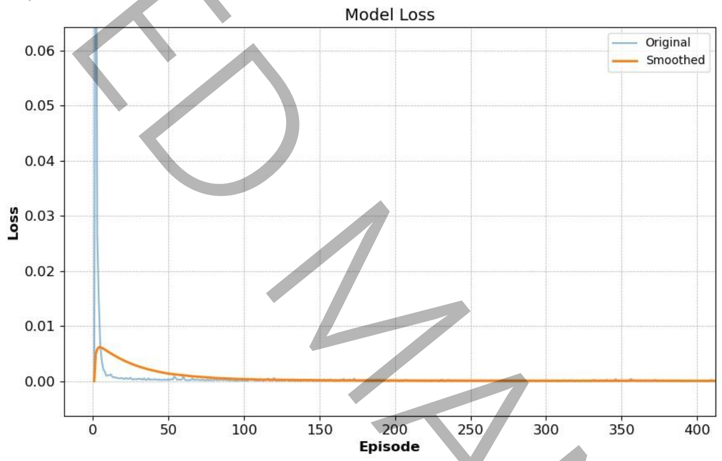

Fig. 8: The environment model loss using the suggested framework, in the presence of DDPG algorithm with tuned hyperparameters

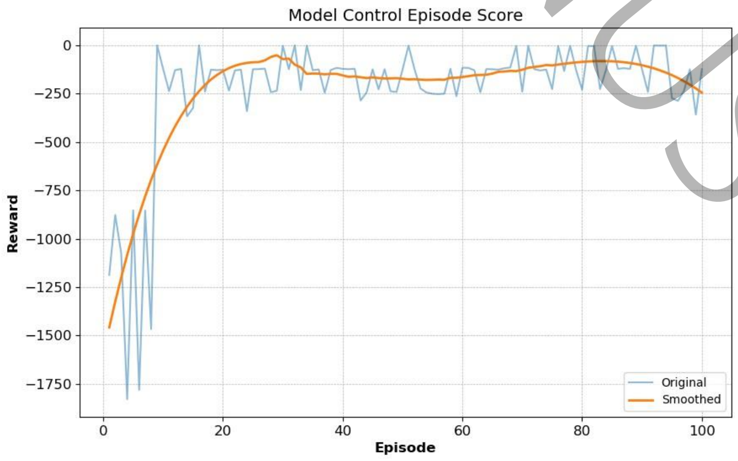

Fig. 9: The acquired reward during model control episodes, in the presence of DDPG algorithm

It is now clear that the complexity of the environment, including the action space and reward function, affects the modeling process, potentially leading to an increase in loss compared to simpler environments.

Moving on to the second step, Fig. 9 illustrates the results of model control episodes. In this environment, the agent aims to keep the pendulum upright by applying continuous torques, and performance is measured by cumulative reward, with optimal policies approaching a score close to zero (since the reward is negative and penalizes deviation from the upright position). The similarity in the trend and final reward between the control episodes in Fig. 6 and model control episodes in Fig. 9 indicates that the policy learned from model-generated data is effectively contributing to the task. Despite differences in learning paths, both curves converge to comparable performance, demonstrating that the model is capable of producing sufficiently accurate transitions to support policy refinement in parallel with real interactions.

4.3 Twin Delayed Deep Deterministic Policy Gradient Results

To validate the introduced framework in the presence of TD3 algorithm, the examination environment becomes more complex as it involves a MIMO system. The experiments using the mentioned approach with TD3 are carried out in the Bipedal Walker environment, part of the Box2D environments available in OpenAI Gym. This setup allows us to observe the impact of transitioning from MISO to MIMO high-dimensional systems on the efficiency of this framework, especially as the complexity of the learning algorithm increases.

4.3.1 Simulation Environment

The Bipedal Walker environment simulates a 4-joint robot. The significance of this examination lies in its high-dimensional, multi-output nature. The state space is a 27-element array, consisting of hull angle speed, angular velocity, horizontal and vertical speed, positions of joints, joints' angular speeds, legs' contact with the ground, and 10 lidar rangefinder measurements.

The action space is a 4-element array representing the motor speed values for each of the 4 joints (hips and knees). The reward function considers the environment solved if the agent achieves a score of +300 within 1600 timesteps. If the robot falls, it incurs a penalty of -100 points, and applying motor torque reduces the score by a small amount. A more optimal agent will attain a higher score.

4.3.2 TD3 Simulation Results

In Figs. 10 and 11, a comparison of hyperparameters for the RL control process is provided. Focusing on the differences in hyperparameters, changes such as increasing the critic learning rate from 0.0001 to 0.0003, adjusting the sample batch size from 128 to 200, modifying the NN architecture from a three-layer 256 × 256 to a three-layer 400 × 300, and altering the policy update frequency from 4 to 2 significantly improved the learning curve, preventing instability and suboptimal performance as shown in Fig. 10. With proper hyperparameter tuning, the control process demonstrated strong performance, converging to a score of 300, as illustrated in Fig. 11.

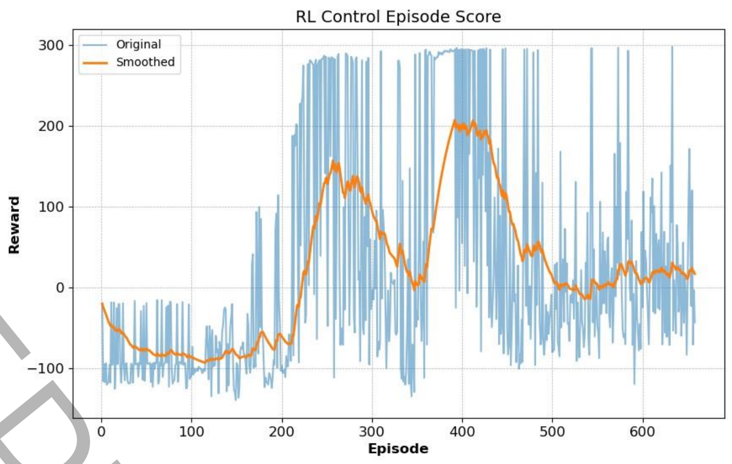

Fig. 10: The obtained reward during RL control episodes, in the presence of TD3 algorithm with untuned hyperparameters

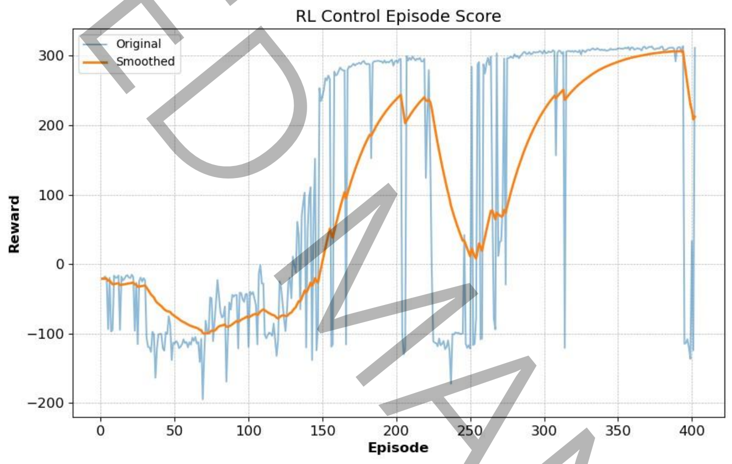

Fig. 11: The obtained reward during RL control episodes, in the presence of TD3 algorithm with tuned hyperparameters

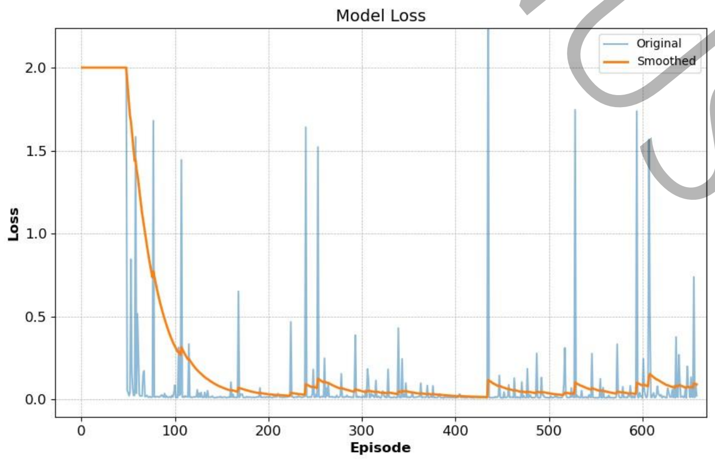

Fig. 12: The environment model loss using the suggested framework, in the presence of TD3 algorithm with untuned hyperparameters

Now, moving on to the modeling and model control phases, the results of the modeling process are shown in Figs. 12 and 13. Designing an online model in this case is much more time-consuming and computationally intensive, considering the complexity of the MIMO system compared to the earlier ones. As a result, the general accuracy of the model has decreased. However, properly adjusting the hyperparameters for the corresponding neural networks can still significantly improve the model's performance. By upgrading from a 3-layer 256 × 128 network with a batch size of 128 to a 4-layer 400 × 300 network with a batch size of 100, the model loss curve improved from Fig. 12 to Fig. 13.

For the model control episodes, 20% of the total episodes are dedicated to the simulated interactions in model, with each episode spanning 1200 timesteps. However, the model control episodes only begin after approximately 180 episodes of RL control have been completed. This decision was made because the modeling process required more time and collected rollouts to establish a solid baseline model before starting the model control phase. To ensure an effective learning process, the rewards obtained from the model episodes were constrained to the range of -300 to 400.

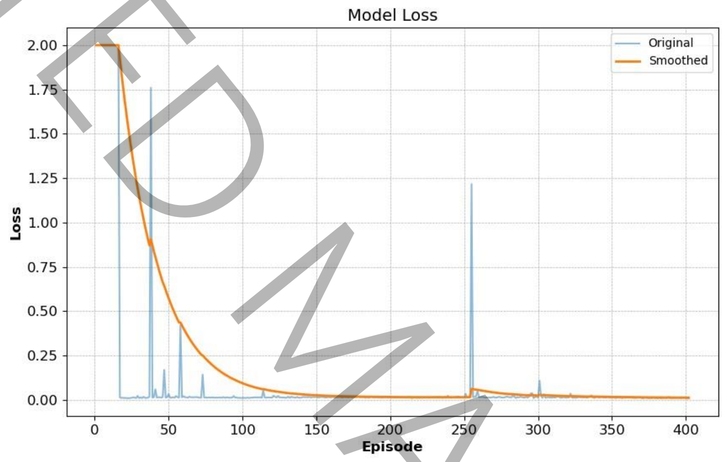

Fig. 13: The environment model loss using the suggested framework, in the presence of TD3 algorithm with tuned hyperparameters

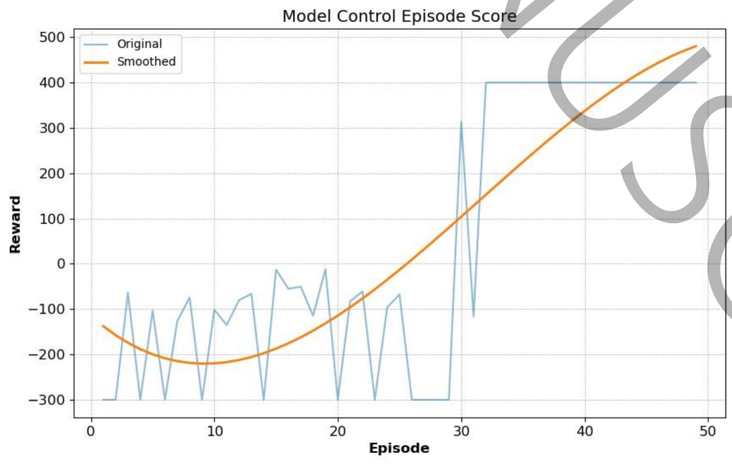

Fig. 14: The acquired reward during model control episodes, in the presence of TD3 algorithm

Fig. 14 shows the learning curve of the model control episodes, highlighting the model's performance, which is relatively close to that expected for the actual bipedal system. In the

Bipedal Walker environment, the agent must learn to coordinate continuous joint actions to walk across uneven terrain, with episode rewards typically ranging from -100 (failure) to 300 (successful walking). The convergence of the model-based learning curve toward the expected reward range indicates that the model is capable of generating realistic transitions and supporting effective policy learning, even in this more complex MIMO control setting.

It should be noted that all experiments were conducted on a Dell XPS 15 laptop equipped with a 12th Gen Intel® Core™ i7-12700H processor (20 CPUs, 2.30 GHz), 32 GB RAM, and an NVIDIA GeForce RTX 3050 Laptop GPU. Training time varied across environments and algorithms; for instance, training with TD3 on the Bipedal Walker environment required approximately 7 hours to converge, while Cart Pole with DQN required around 40 minutes. The lightweight model used in system identification enabled fast inference, supporting near real-time performance in simulation.

5 Conclusion

In this study, an improved framework for DRL was presented that integrates online system identification, based on the Dyna-Q approach. The parallel operation of system identification and model control processes with the control process in the suggested approach provides a reliable backup mechanism for industrial settings where system failures can have critical consequences. The framework's strengths and limitations were thoroughly investigated upon experiments across diverse industry-relevant environments, providing insights into its potential for addressing key challenges in industrial control systems. However, the time-consuming system identification process may limit the applicability of the suggested framework for some high-dimensional systems. Future work should focus on optimizing the system identification process for high-dimensional tasks and exploring ways to mitigate the approach's dependence on highly accurate models. In addition, while hyperparameter tuning in this study was performed manually through iterative refinement, effective for benchmarking purposes, future work may benefit from automated optimization methods such as Bayesian optimization to improve robustness and reduce tuning overhead in real-world deployments. To further improve scalability, future extensions could explore transfer learning, lightweight surrogate models, and distributed or multi-agent DRL implementations. In such settings, data-sharing constraints and communication efficiency would become critical, and techniques like log-scale quantization may help reduce bandwidth requirements while preserving learning performance.

6 Acknowledgment

Our sincere gratitude to the generous support of the Mobarakeh Steel Technology and Innovation Development (MSTID) Company, under the contract number 1402-1025222, which enabled us to carry out this study.

References

[1] K.H. Ang, G. Chong, Y. Li, PID control system analysis, design, and technology, IEEE Transactions on Control Systems Technology, 13(4) (2005) 559–576.

[2] M. Morari, J.H. Lee, Model predictive control: past, present and future, Computers & Chemical Engineering, 23(4–5) (1999) 667–682.

[3] A. Kuhnle, J.-P. Kaiser, F. Theiß, N. Stricker, G. Lanza, Designing an adaptive production control system using reinforcement learning, Journal of Intelligent Manufacturing, 32 (2021) 855–876.

[4] D. Lee, S. Koo, I. Jang, J. Kim, Comparison of deep reinforcement learning and PID controllers for automatic cold shutdown operation, Energies, 15(8) (2022) 2834.

[5] Z. Wang, T. Hong, Reinforcement learning for building controls: The opportunities and challenges, Applied Energy, 269 (2020) 115036.

[6] X. Tao, D. Zhang, W. Ma, X. Liu, D. Xu, Automatic metallic surface defect detection and recognition with convolutional neural networks, Applied Sciences, 8(9) (2018) 1575.

[7] P.J. Antsaklis, A. Rahnama, Control and machine intelligence for system autonomy, Journal of Intelligent & Robotic Systems, 91 (2018) 23–34.

[8] J.F. Arinez, Q. Chang, R.X. Gao, C. Xu, J. Zhang, Artificial intelligence in advanced manufacturing: Current status and future outlook, Journal of Manufacturing Science and Engineering, 142(11) (2020) 110804.

[9] S. Spielberga, A. Tulsyana, N.P. Lawrenceb, P.D. Loewenb, R.B. Gopalunia, Deep reinforcement learning for process control: A primer for beginners, Journal, 65(10) (2019).

[10] Y.-J. Park, S.-K.S. Fan, C.-Y. Hsu, A review on fault detection and process diagnostics in industrial processes, Processes, 8(9) (2020) 1123.

[11] F.-M. Luo, T. Xu, H. Lai, X.-H. Chen, W. Zhang, Y. Yu, A survey on model-based reinforcement learning, Science China Information Sciences, 67(2) (2024) 121101.

[12] R.S. Sutton, Integrated architectures for learning, planning, and reacting based on approximating dynamic programming, Machine Learning Proceedings 1990, Elsevier, 1990, pp. 216–224.

[13] R.S. Sutton, Dyna, an integrated architecture for learning, planning, and reacting, ACM Sigart Bulletin, 2(4) (1991) 160–163.

[14] E. Vitolo, A. San Miguel, J. Civera, C. Mahulea, Performance evaluation of the Dyna-Q algorithm for robot navigation, in: 2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), IEEE, 2018, pp. 322–327.

[15] S.P. Singh, Reinforcement learning with a hierarchy of abstract models, in: Proceedings of the National Conference on Artificial Intelligence, Citeseer, 1992, p. 202.

[16] B. Peng, X. Li, J. Gao, J. Liu, K.-F. Wong, Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2018, pp. 2182–2192.

[17] G.Z. Holland, The effect of planning shape on Dyna-style planning in high-dimensional state spaces, PhD diss., University of Alberta, 2018.

[18] H. Liu, Y. Yao, T. Li, M. Du, X. Wang, H. Li, M. Li, Dyna algorithm-based reinforcement learning energy management for fuel cell hybrid engineering vehicles, Journal of Energy Storage, 94 (2024) 112526.

[19] A. Budiyanto, K. Azetsu, K. Miyazaki, N. Matsunaga, On fast learning of cooperative transport by multi-robots using DeepDyna-Q, in: 2022 61st Annual Conference of the Society of Instrument and Control Engineers (SICE), IEEE, 2022, pp. 1058–1062.

[20] G. Kalweit, J. Boedecker, Uncertainty-driven imagination for continuous deep reinforcement learning, in: Conference on Robot Learning, PMLR, 2017, pp. 195–206.

[21] J. Kulhánek, E. Derner, T. de Bruin and R. Babuška, Vision-based navigation using deep reinforcement learning, in: 2019 European Conference on Mobile Robots (ECMR), Prague, Czech Republic, 2019, pp. 1–8.

[22] D. Hafner, J. Pasukonis, J. Ba, T. Lillicrap, Mastering diverse control tasks through world models, Nature, 640 (2025) 647–653.

[23] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, J. Davidson, Learning latent dynamics for planning from pixels, in: International Conference on Machine Learning, PMLR, 2019, pp. 2555–2565.

[24] M. Janner, J. Fu, M. Zhang, S. Levine, When to trust your model: Model-based policy optimization, In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, 2019, pp. 12519–12530.

[25] S. Fujimoto, H. Hoof, D. Meger, Addressing function approximation error in actor-critic methods, in: International Conference on Machine Learning, PMLR, 2018, pp. 1587–1596.

[26] P. Palanisamy, Hands-on intelligent agents with OpenAI Gym, Packt Publishing Ltd., Birmingham, UK, 2018.

[27] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction, MIT Press, Cambridge, MA, 2018.

[28] Cs. Szepesvari, Algorithms for reinforcement learning, Morgan & Claypool Publishers, Switzerland, 2010.

[29] Y.-t. Liu, J.-m. Yang, L. Chen, T. Guo, Y. Jiang, Overview of reinforcement learning based on value and policy, in: 2020 Chinese Control and Decision Conference (CCDC), IEEE, 2020, pp. 598–603.

[30] C.J. Watkins, P. Dayan, Q-learning, Machine Learning, 8 (1992) 279–292.

[31] R.S. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximately, Advances in Neural Information Processing Systems, 12 (2000) 1057–1063.

[32] D. Bennett, Y. Niv, A.J. Langdon, Value-free reinforcement learning: Policy optimization as a minimal model of operant behavior, Current Opinion in Behavioral Sciences, 41 (2021) 114–121.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature, 518(7540) (2015) 529–533.

[34] N. Gobinathan, R. Ponnusamy, Deep-Q-based reinforcement learning method to predict accuracy of Atari gaming set classification, in: 2023 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI), Chennai, India, 2023, pp. 1–4.

[35] R. Bellman, Dynamic Programming, 1st Ed., Princeton University Press, Princeton, NJ, USA, 1957.

[36] H. Tan, Reinforcement learning with deep deterministic policy gradient, in: 2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA), Xi'an, China, 2021, pp. 82–85.

[37] R.Y. Choi, A.S. Coyner, J. Kalpathy-Cramer, M.F. Chiang, J.P. Campbell, Introduction to machine learning, neural networks, and deep learning, Translational Vision Science & Technology, 9(2) (2020) 14.

[38] J. Terven, D.-M. Cordova-Esparza, J.-A. Romero-González, A. Ramírez-Pedraza, E.A. Chávez-Urbiola, A comprehensive survey of loss functions and metrics in deep learning, Artificial Intelligence Review, 58(7) (2025) 195.