Reinforcement Learning Based Model Predictive Controller Design

4Y04 Final Report

by

Barry Chen

Supervisor: Prashant Mhaskar
Department of Chemical Engineering, McMaster University
March, 2023

Abstract

Machine learning (ML) has long been used in chemical engineering to address the need of learning from data. Recently, there has been a growing interest in improving current industrial process control by adopting reinforcement learning (RL) technologies. It is promising to leverage RL approaches to tackle the limitations of the current model predictive controller (MPC). In this work, in addition to introducing a model-free RL-based MPC that handles nonlinear process dynamics, we propose a new way to set up the RL-based MPC through pre-training. In such a method, the RL-based MPC starts by duplicating a proven offset-free MPC via offline generated data. Then, the MPC improves itself further via online interactions. The traditional RL-based MPC requires a considerably large amount of data from real-time interactions with the environment, which could bring safety concerns. The RL-based MPC we propose can reduce online computation costs and ensure safety when implemented in a practical industrial process.

Table of Contents

Abstract . . . . . **i**

1 Introduction **1**

2 Background **1**
2.1 Offset-free MPC formulation 1
2.2 Actor-critic method in RL 2

3 Methods **2**
3.1 CSTR Model 2
3.2 Water Tank Model 3

4 Results **5**
4.1 CSTR Model 5
4.1.1 CSTR Model Actor Network Validation 5
4.1.2 Continuous Set Point Tracking Environment 6
4.1.3 CSTR Model Online Training 7
4.2 Water Tank Model 10

5 Discussion **10**

6 Conclusion **11**

References **12**

Appendices **13**
Appendix A: Additional CSTR Model Simulation Results 13
Appendix B: Additional Water Tank Model Simulation Results 16

1 Introduction

The design of an optimal control system is critical to technical systems automation in the chemical process industry. While classic approaches such as PID controllers have widespread in industrial applications, their inefficiency, caused by disturbance, dead time and constraints, leads to compromised product quality and waste of energy resources. Additionally, it is often observed in process industries that the nonlinear nature and multi-input multi-output (MIMO) systems exist, which makes the parametrization of PID controllers especially complicated. Therefore, a decent approach, MPC, which is based on recurrent real-time optimization of a mathematical process model, was developed to address the shortcomings of classic control approaches. Although MPC requires higher computational cost, it brings advantages such as implicit formulation, flexibility, and explicit model usage [1][2].

The research of adopting RL techniques to MPC has received growing attention since MPC and RL are similar in the ability to find optimal control actions. MPC generally uses models, and the performance depends heavily on the goodness of the models. However, there are many limitations of the current MPC, including the requirement of accurate models, the linear and nonadaptive nature of the conventional controllers, and the complexity of online computations [3]. In contrast, RL is generally model-free, and the efficiency relies on the effectiveness of offline/online training. The structure of MPC is fixed once put for online operations, while RL doesn't have a fixed one. Therefore, RL has the potential to accommodate unexpected changes in the actual process operations through online evolution [4]. Another advantage of RL is that it needs relatively less computational effort compared to recurrent online optimization in MPC.

Motivated by the strengths of RL approaches, we are also interested in exploring RL-based MPC design. The traditional online training process for RL is usually time-consuming and doesn't always guarantee acceptable control actions, which could lead to high system malfunctions [4]. Thus, we seek a new RL-based MPC initialized to have a decent starting point from which it could adapt to real-time process dynamics through online interactions. Our new approach has a more data-efficient training process and ensures proper control solutions at the early phase of process operations, bringing lower computational costs and safety risks.

2 Background

2.1 Offset-free MPC formulation

The nonlinear model investigated in the report is a continuous stirred tank reactor (CSTR) under closed-loop control, where an exothermic reaction of A turns into B takes place. The description of dynamics in the ODE form is as the following [5]:

$$\dot{C_A} = \frac{F}{V}(C_{A0} - C_A) - k_0 e^{-\frac{E}{RT_R}} C_A$$

$$\dot{T_R} = \frac{F}{V}(T_0 - T_R) + \frac{(-\Delta H)}{\rho C_p} k_0 e^{-\frac{E}{RT_R}} - \frac{UA}{\rho C_p V}(T_R - T_C)$$

$$\dot{T_C} = \frac{F_C}{V_C}(T_{cf} - T_C) + \frac{UA}{\rho_C C_{pC} V_C}(T_R - T_C)$$

The ODE system is then converted to a discrete-time model to enable MPC usage:

$$\begin{cases} \mathbf{x}_k = \mathbf{A} \cdot \mathbf{x}_{k-1} + \mathbf{B} \cdot \mathbf{u}_{k-1} \\ \mathbf{y}_{k-1} = \mathbf{C} \cdot \mathbf{x}_{k-1} + \mathbf{D} \cdot \mathbf{u}_{k-1} \end{cases}$$

In the model, there are four state variables, two manipulated inputs (the reactor feed flow rate F and the coolant feed flow rate F_C), and two measured outputs (the reactor temperature T_R and the coolant temperature T_C). $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are nonzero matrices, $\mathbf{D}$ is a zero matrix.

Furthermore, the offset-free approach is used to handle plant-model mismatch in MPC by introducing additional "integrating" states ($\boldsymbol{\theta}$). The new model takes the following form:

$$\begin{cases} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\theta} \end{bmatrix}_k = \begin{bmatrix} \mathbf{A} & \mathbf{G}_{\boldsymbol{\theta}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\theta} \end{bmatrix}_{k-1} + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \cdot \mathbf{u}_{k-1} \\ \mathbf{y}_{k-1} = \begin{bmatrix} \mathbf{C} & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\theta} \end{bmatrix}_{k-1} + \mathbf{D} \cdot \mathbf{u}_{k-1} \end{cases}$$

In the new model, $\mathbf{G}_{\boldsymbol{\theta}}$ is a tunable parameter and is chosen to be a zero matrix. $\mathbf{I}$ is an identity matrix. $\mathbf{L}$ is the observer gain matrix and is calculated by tuning the matrix $\mathbf{P}$. The estimation of the augmented states ($\tilde{\mathbf{x}}$) is computed by the Luenberger observer [6]:

$$\begin{cases} \hat{\mathbf{y}}_{k-1} = \begin{bmatrix} \mathbf{C} & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\boldsymbol{\theta}} \end{bmatrix}_{k-1} + \mathbf{D} \cdot \mathbf{u}_{k-1} \\ \tilde{\mathbf{x}}_k = \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\boldsymbol{\theta}} \end{bmatrix}_k = \begin{bmatrix} \mathbf{A} & \mathbf{G}_{\boldsymbol{\theta}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{x}} \\ \hat{\boldsymbol{\theta}} \end{bmatrix}_{k-1} + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \cdot \mathbf{u}_{k-1} + \mathbf{L} \cdot \left[\mathbf{y}_{k-1} - \hat{\mathbf{y}}_{k-1} \right], \\ \text{where } \mathbf{E} = \begin{bmatrix} \mathbf{A} & \mathbf{G}_{\boldsymbol{\theta}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} - \mathbf{L} \cdot \begin{bmatrix} \mathbf{C} & \mathbf{I} \end{bmatrix}, \ \mathbf{E} \cdot \vec{v}_m = \lambda_m \cdot \vec{v}_m, \\ \mathbf{P} = \begin{bmatrix} \lambda_1 & \lambda_2 & \dots & \lambda_m \end{bmatrix}, \tilde{\mathbf{x}}_k \in \mathbb{R}^{m \times 1} \end{cases}$$

2.2 Actor-critic method in RL

In brief, RL is developed to mimic human behaviours in the sequential decision-making process. The early RL approaches were generally value-based or policy-based. However, it has been discovered that value-based methods would diverge in largescale problems while policy-based approaches would have significant variance in value estimation. Thus, a hybrid technique combining the value-based and the policy-based methods, the actor-critic (AC) approach, was introduced for better performance. AC consists of two neural networks: the actor network (π_θ), which aims to find the optimal policy, and the critic network (Q_ϕ), which evaluates the decision made in certain situations. There are many popular algorithms in AC, such as deep deterministic policy gradient (DDPG), which works in continuous action spaces [7]. Due to the feature of model-free and determinism, the RL-based MPC in the report uses a DDPG framework.

3 Methods

3.1 CSTR Model

Our proposed approach is first tested on the nonlinear CSTR model described in section 2.1. The initial step involves transferring the knowledge from the offset-free MPC to the actor network constructed by a deep neural network (DNN). In this step, the offset-free MPC generates offline data in an open-loop manner, which consists of feature variables (state variables, past control actions, set point information) and target variables (current control actions). The DNN substitutes the actor network portion in the RL agent upon being fully trained via

the offline data. The pre-trained actor network is validated by comparing it against the offset-free MPC and checking its online training results using a very low actor learning rate.

Next, the objective of the RL-based MPC is set up by formulating a reward function. While the ordinary MPC tackles the minimization problems, the RL-based MPC solves maximization problems. Therefore, we choose the reward function to be the negative value of the objective function in the offset-free MPC, which is:

$$r_t = -\left(Q_y \cdot \sum_{i=1}^{n_y} |y_{i,t} - y_i^{sp}|^2 + R_u \cdot \sum_{i=1}^{n_u} |u_{i,t-1} - u_{i,t}|^2 \right)$$

Subsequently, we implement continuous set point tracking by modifying the local reset function that adjusts the model's parameters, such as state and set point values, in the Simulink RL environment. By default, the local reset function randomizes parameters at the beginning of each episode. However, it is undesirable in continuous set point tracking, where the current states inherit from the previous ones. The feature of inheriting-state is realized by accessing past data through Simulation Data Inspector.

Lastly, the model undergoes online training by tuning hyperparameters (learning rates, noise variance, noise variance decay rate, discount factor, buffer size, smoothing factor), expecting to find a sweet point where the resulting RL-based MPC outperforms the one before online training. In each online training session, the maximum number of training episodes is set to 3456 and set points change every 20 episodes. While not strictly necessary for this step, the pre-trained critic network can be acquired through online training with the actor learning rate set to 0. The pseudocode of our RL method is provided in Algorithm 1.

3.2 Water Tank Model

To investigate the feasibility of our proposed approach, a modified MATLAB water tank reinforcement learning model is also examined, which is a single-input single-output (SISO) system [8]. The testing for the water tank model undergoes similar steps. Instead of building a state space model and then replacing it with a DNN, the first step is to obtain an actor network through a conventional reinforcement learning process, which can be insufficiently trained. Next, similar reward and local reset functions are employed in the water tank model. Subsequently, the actor network is further trained by performing the continuous set point tracking task, and its resulting performance is then compared to its pre-training state. During each testing trial, the maximum number of training episodes is set to 1000 and the set point is adjusted every 10 episodes.

Algorithm 1 DDPG with pre-training

Initialize actor network $\pi(s, \theta)$ with pre-trained parameters θ^p
if pre-trained critic network is available **then**
    Initialize critic network $Q(s, a, \phi)$ with pre-trained parameters ϕ^p
else
    Initialize critic network with random parameters ϕ
end if
Initialize target network Q' and π' with the same parameters as Q and π, respectively
Initialize experience buffer $\mathcal{B}$
for episode = 1 to max-episode **do**
    Initialize stochastic noise $\mathcal{N}$ for action exploration
    Receive initial observation s_1
    for t = 1 to max-step **do**
        Select action $a_t = \pi(s_t, \theta) + \mathcal{N}_t$ according to the current policy and noise model
        Execute action a_t, observe reward r_t and new state s_{t+1}
        Store the experience $< s_t, a_t, r_t, s_{t+1} >$ in $\mathcal{B}$
        Sample a random minibatch of M experiences $< s_i, a_i, r_i, s_{i+1} >$ from $\mathcal{B}$
        Compute the value function target:

$$y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}, \theta'), \phi')$$

        Update Q by minimizing the loss function value:

$$L = \frac{1}{M} \sum_{i=1}^{M} (y_i - Q(s_i, a_i, \phi))^2$$

        Update π using the sampled policy gradient:

$$\nabla_\theta J \approx \frac{1}{M} \sum_{i=1}^{M} \nabla_{\pi(s_i,\theta)} Q(s_i, \pi(s_i, \theta), \phi) \nabla_\theta \pi(s_i, \theta)$$

        Update the target network Q' and π':

$$\phi' = \tau\phi + (1 - \tau)\phi'$$
$$\theta' = \tau\theta + (1 - \tau)\theta'$$

    end for
end for

4 Results

4.1 CSTR Model

4.1.1 CSTR Model Actor Network Validation

Figure 1 illustrates that the output response trajectories from the two MPCs are identical, indicating that in the absence of interactions with the environment, the proposed RL-based MPC can effectively replicate the performance of the offset-free MPC.

Figure 2 shows that the performance of the proposed RL-based MPC is almost identical before and after doing online training, which means the proposed RL-based MPC can work safely like an offset-free MPC when learning from the environment very slowly.

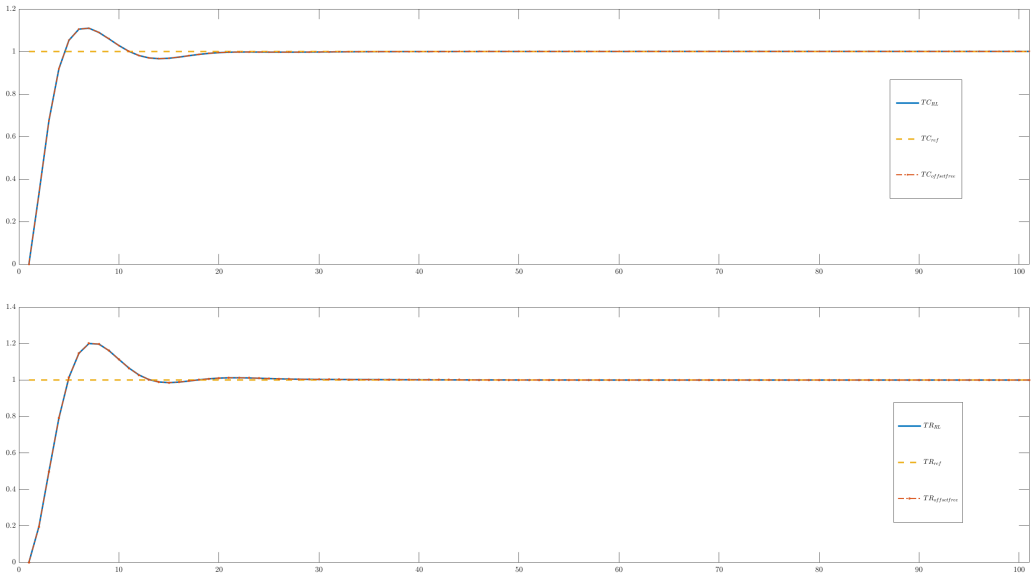

Figure 1: Performance comparison between offset-free MPC (solid blue line) and the proposed RL-based MPC without online interactions (dot-dash red line). The dashed yellow line represents set points.

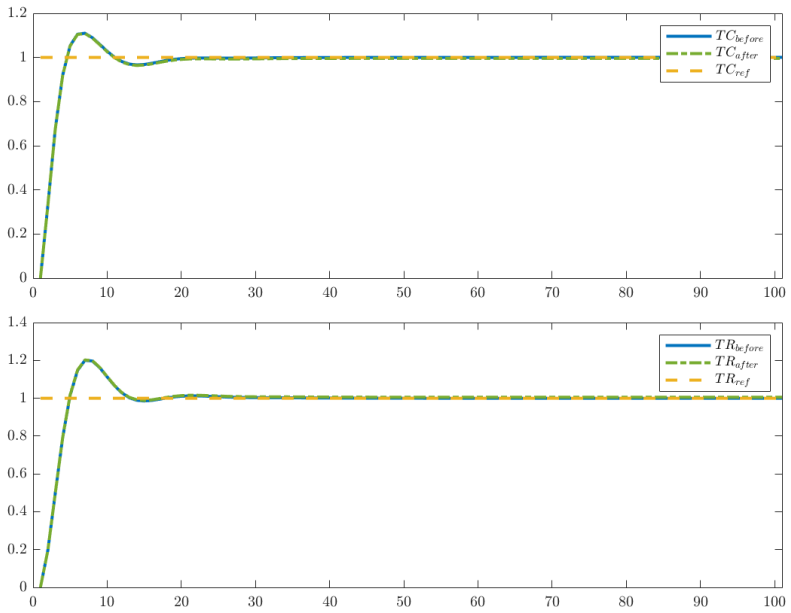

Figure 2: The performance comparison between the proposed RL-based MPC before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow line represents set points. Tuning parameters: actor learning rate = 1×10^{-10}.

4.1.2 Continuous Set Point Tracking Environment

Figure 3 illustrates the successful inheritance of current states from the states of the last episode, which implies the implementation of continuous set point tracking in the Simulink RL environment using the inheriting-state method can replicate the industrial process control environment.

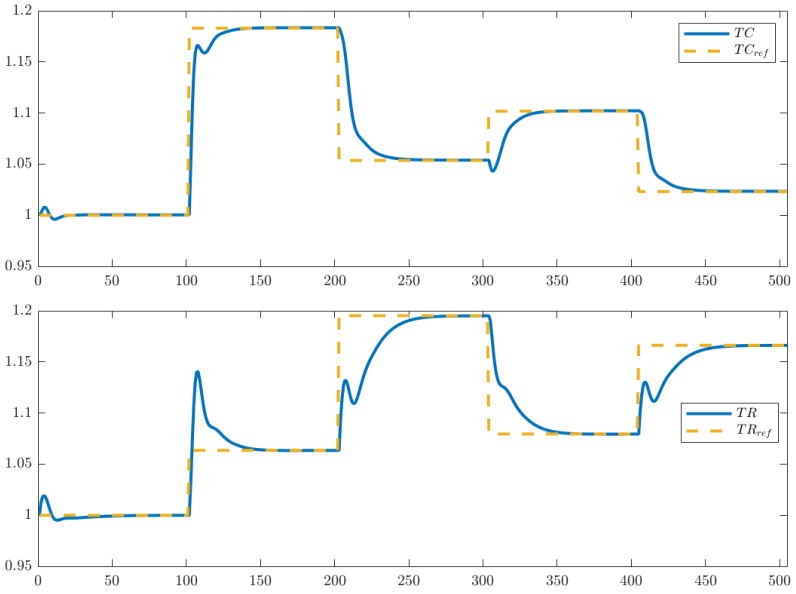

Figure 3: The output response trajectories (solid blue line) of five consecutive set point tracking using the inheriting-state method. The dashed yellow line represents set points.

4.1.3 CSTR Model Online Training

Figure 4 shows the training results without using a pre-trained critic network. It is obvious that, after doing online training, a reduction in overshoot is observed in the output response T_R. However, slightly oscillatory behaviour and more overshoot are observed in the output response T_C. Figure 5 shows that increasing the actor learning rate while keeping other hyperparameters constant leads to significantly worse behaviour of the T_C output.

Testing using a pre-trained critic network is also performed. Figure 6 illustrates the training result of the critic network No.22. If the critic network is well-trained, episode Q0 (represented by the yellow line) is expected to converge to the true long-term discounted reward (represented by the blue line). In the figure, episode Q0 follows the general trend of the episode reward, however, there are noticeable deviations throughout the training process, which indicates the accuracy of the resulting critic network is limited. Figure 7 demonstrates that when using the same hyperparameters as Figure 4, the outcomes using the pre-trained critic network are worse (showing noticeable offsets) compared to the dynamics shown in Figure 4.

Figure 8 illustrates comparable training outcomes as those depicted in Figure 4 when using the pre-trained critic network and a different set of hyperparameters. However, when using other sets of hyperparameters, it is often observed that output response T_C fails to remain stable at the reference line despite the improvement in output response T_R.

Please refer to Appendix A for additional training results.

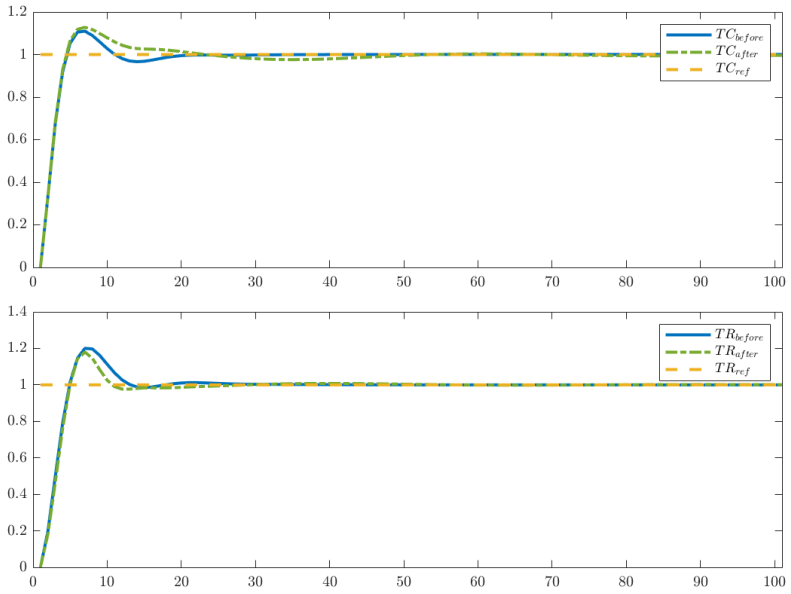

Figure 4: The performance comparison between the proposed RL-based MPC before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 1×10^{-2}, noise variance = 1.25×10^{-3}, noise variance decay rate = 5×10^{-6}, discount factor = 0.95.

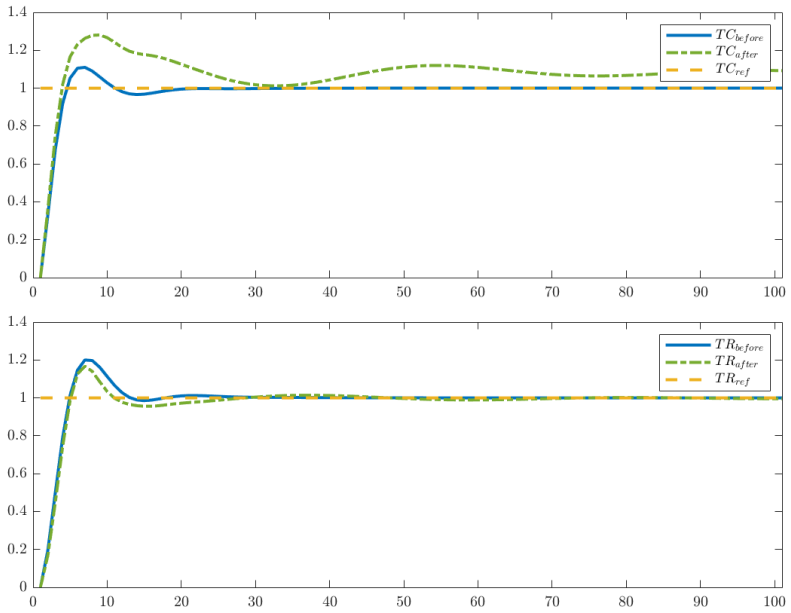

Figure 5: The performance comparison between the proposed RL-based MPC before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1.5×10^{-7}, critic learning rate = 1×10^{-2}, noise variance = 1.25×10^{-3}, noise variance decay rate = 5×10^{-6}, discount factor = 0.95.

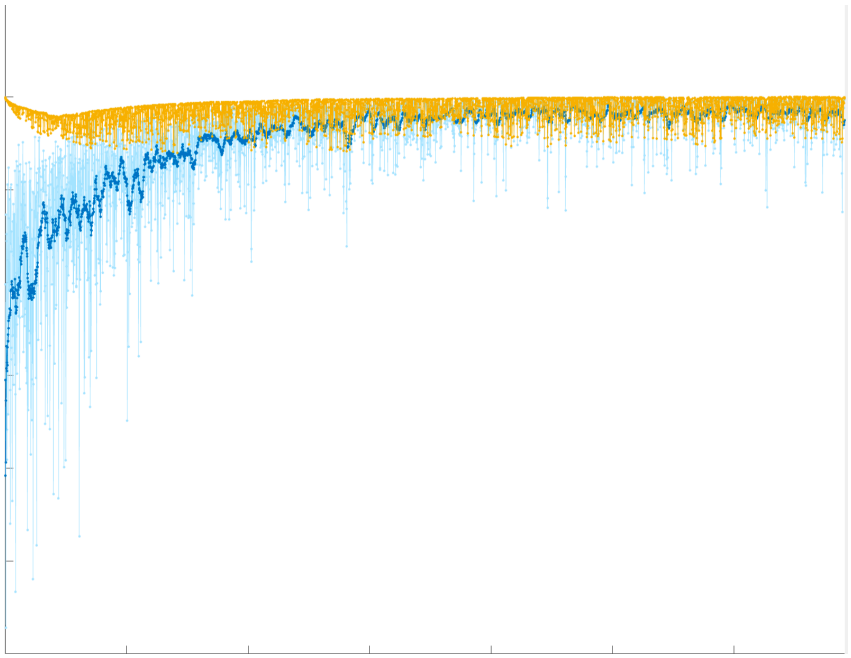

Figure 6: The training session diagram for pre-training critic network No.22. The horizontal axis denotes the episode number while the vertical axis represents the episode reward. The blue, light blue, and yellow lines represent the average reward, episode reward, and episode Q0 respectively. Tuning parameters: actor learning rate = 0, critic learning rate = 1×10^{-2}, noise variance = 2×10^{-3}, noise variance decay rate = 1×10^{-5}, discount factor = 0.95.

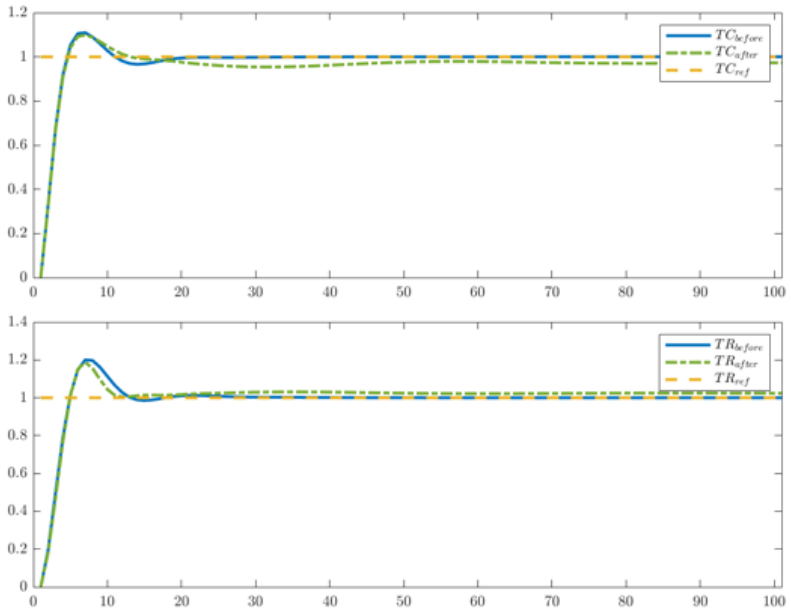

Figure 7: The performance comparison between the proposed RL-based MPC before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 1×10^{-2}, noise variance = 1.25×10^{-3}, noise variance decay rate = 5×10^{-6}, discount factor = 0.95. Pre-trained critic network No.22 is used.

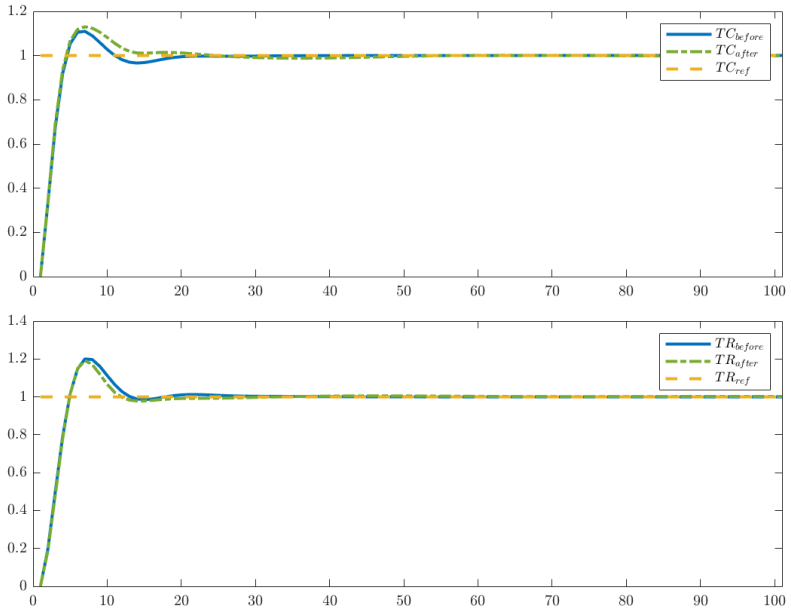

Figure 8: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 5.2×10^{-3}, noise variance = 1.5×10^{-3}, noise variance decay rate = 6×10^{-6}, discount factor = 0.95. Pre-trained critic network No.22 is used.

4.2 Water Tank Model

In Figure 9, the output response of the RL controller after online training not only successfully archives the set point but also exhibits slightly less overshoot compared to that before online training.

Please refer to Appendix B for additional training results.

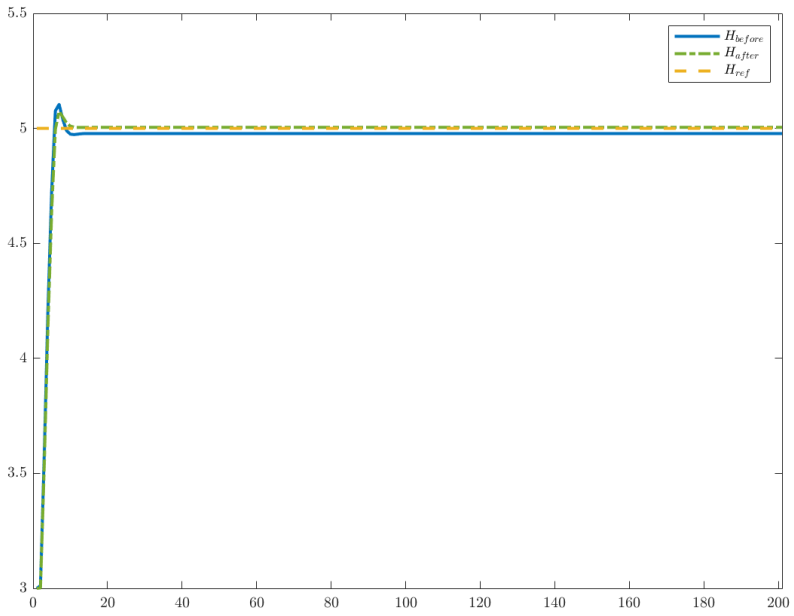

Figure 9: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow line represents the set point. Tuning parameters: actor learning rate = 2×10^{-8}, critic learning rate = 5×10^{-3}, noise variance = 0.5, noise variance decay rate = 1×10^{-5}, discount factor = 0.95.

5 Discussion

Based on the findings presented in section 4, it is evident that our proposed approach is effective in a SISO system, which validates the feasibility of our proposed method. However, the application of our approach in a MIMO system has only achieved partial success. This may be attributed to the RL agent's insufficient capability of handling complex MIMO systems. Having more state variables compared to the modified water tank model may result in increased computational complexity in the CSTR model. It's also observed that the RL agent struggles to improve both output responses, which implies that the RL agent faces difficulty coordinating multiple outputs of the MIMO system simultaneously. Thus, it is more challenging for the RL agent to determine an optimal control policy for the CSTR model.

Another potential factor that could contribute to the undesired training results of the CSTR model is the function approximation error in DDPG. In the AC method, the overestimation bias occurs in slightly stricter conditions with unnormalized gradients. Without proper handling, the overestimation could grow into a more significant bias in subsequent updates. The buildup of this error could lead to high variance in value estimation, causing poor policy updates under noisy gradients and even divergence [9].

The training results of the water tank model indicate that there may exist a set of hyperparameters (i.e., a sweet point) for the CSTR model as well. However, without having a systematic way of choosing hyperparameters, tuning a large number of hyperparameters could be very time-consuming. One research paper points out that the design of experiments (DOE) could make the process of finding the optimal hyperparameters more effective [10].

Most of the state-of-art off-policy RL algorithms fall into the category called growing batch, where data is stored in a memory replay buffer. When the hidden state-action pairs are erroneously estimated due to a mismatch between the data distribution in the batch and the policy, off-policy algorithms may fail to learn the true off-policy. According to a journal article, restricting the action space to force the agent to behave close to on-policy could mitigate the extrapolation error [11]. Memory replay buffer also plays an important role in RL because it assures the success of many RL algorithms including DDPG. It is worth noticing that past experiences are sampled uniformly in our DDPG algorithm. The disadvantage of this random sampling strategy is that the quality difference among experiences is neglected. To address the drawback, prioritized memory replay is suggested [12]. To improve our current algorithm, transitioning to Python is necessary as MATLAB offers limited customization features.

6 Conclusion

In this project, a novel RL-based approach with pre-training is proposed to develop a learning-based MPC which requires relatively low online computation costs. Our proposed approach works in a SISO system, but it's not fully successful in a MIMO system. To improve the effectiveness of our approach in the MIMO system, it is worthwhile to conduct further research, including using DOE to systematically select hyperparameters for testing and transitioning the work to Python to add customized features that can improve the effectiveness of the DDPG algorithm.

References

[1] P. K. Juneja, S. K. Sunori, A. Sharma, A. Sharma, H. Pathak, V. Joshi, and P. Bhasin, "A review on control system applications in industrial processes," IOP Conference Series: Materials Science and Engineering, vol. 1022, no. 1, p. 012010, 2021.

[2] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on Model predictive control: An engineering perspective," The International Journal of Advanced Manufacturing Technology, vol. 117, no. 5-6, pp. 1327–1349, 2021.

[3] S. Spielberg, A. Tulsyan, N. P. Lawrence, P. D. Loewen, and R. Bhushan Gopaluni, "Toward self-driving processes: A deep reinforcement learning approach to control," AIChE Journal, vol. 65, no. 10, 2019.

[4] A. Mugnini, F. Ferracuti, M. Lorenzetti, G. Comodi, and A. Arteconi, "Advanced control techniques for CHP-DH systems: A critical comparison of model predictive control and reinforcement learning," Energy Conversion and Management: X, vol. 15, p. 100264, 2022.

[5] H. Hassanpour, B. Corbett, and P. Mhaskar, "Artificial Neural Network based model predictive control: Implementing achievable set-points," AIChE Journal, vol. 68, no. 1, 2021.

[6] M. Wallace, S. S. Pon Kumar, and P. Mhaskar, "Offset-free model predictive control with explicit performance specification," Industrial & Engineering Chemistry Research, vol. 55, no. 4, pp. 995–1003, 2016.

[7] O. Dogru, K. Velswamy, and B. Huang, "Actor–critic Reinforcement Learning and application in developing computer-vision-based interface tracking," Engineering, vol. 7, no. 9, pp. 1248–1261, 2021.

[8] "Water Tank Reinforcement Learning Environment Model," MathWorks. Available: https://www.mathworks.com/help/reinforcement-learning/ug/water-tank-reinforcement-learning-environment-model.html.

[9] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," Proceedings of the 35th International Conference on Machine Learning, vol. 80, 2018.

[10] J. Freiesleben, J. Keim, and M. Grutsch, "Machine learning and design of experiments: Alternative approaches or complementary methodologies for quality improvement?," Quality and Reliability Engineering International, vol. 36, no. 6, pp. 1837–1848, 2020.

[11] S. Fujimoto, D. Meger, and D. Precup, "Off-Policy Deep Reinforcement Learning without Exploration," Proceedings of the 36th International Conference on Machine Learning, vol. 97, 2019.

[12] Y. Hou and Y. Zhang, "Improving DDPG via Prioritized Experience Replay," Research Gate, 2019.

Appendices

Appendix A: Additional CSTR Model Simulation Results

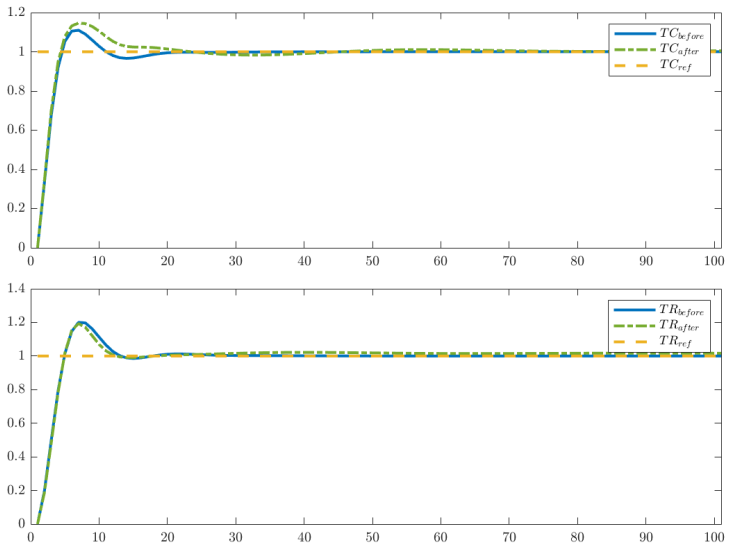

Figure A. 1: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 1×10^{-2}, noise variance = 2×10^{-3}, noise variance decay rate = 1×10^{-5}, discount factor = 0.95.

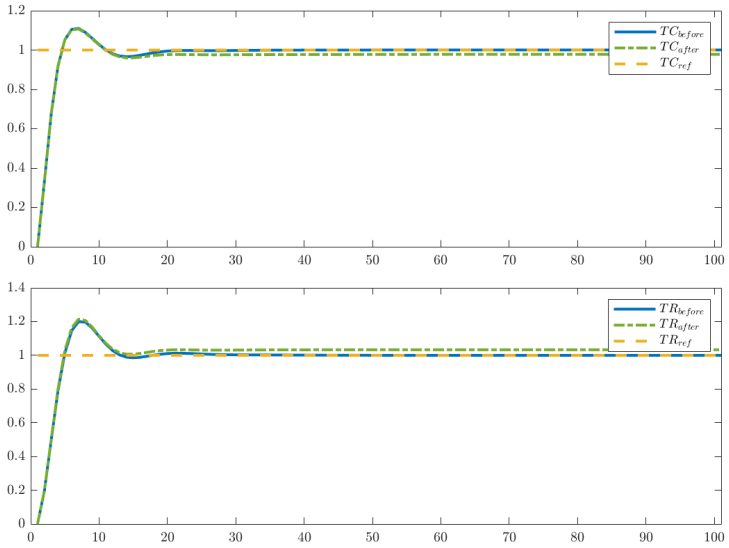

Figure A. 2: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 8×10^{-9}, critic learning rate = 8×10^{-2}, noise variance = 2×10^{-3}, noise variance decay rate = 1×10^{-5}, discount factor = 1.

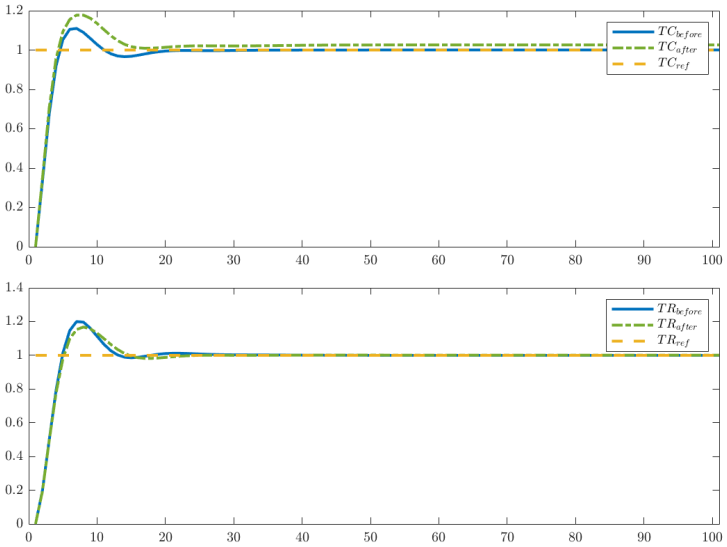

Figure A. 3: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 1×10^{-2}, noise variance = 2×10^{-3}, noise variance decay rate = 8×10^{-6}, discount factor = 0.95.

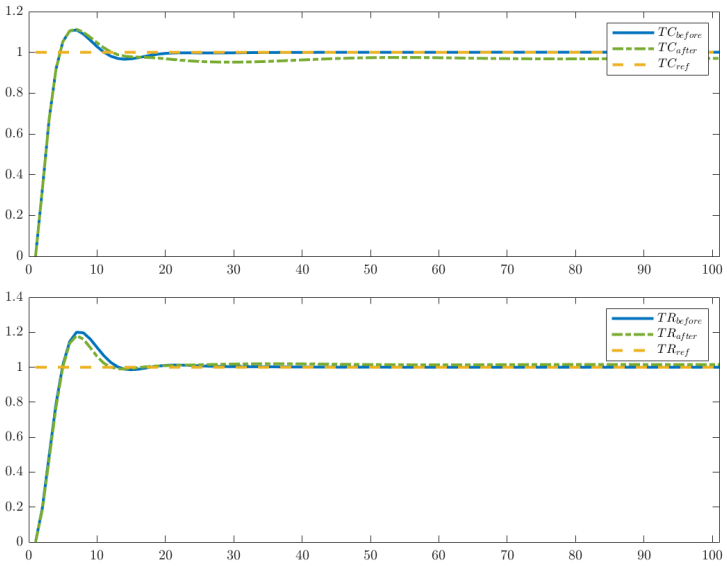

Figure A. 4: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 6.8×10^{-3}, noise variance = 2.25×10^{-3}, noise variance decay rate = 9×10^{-6}, discount factor = 0.97.

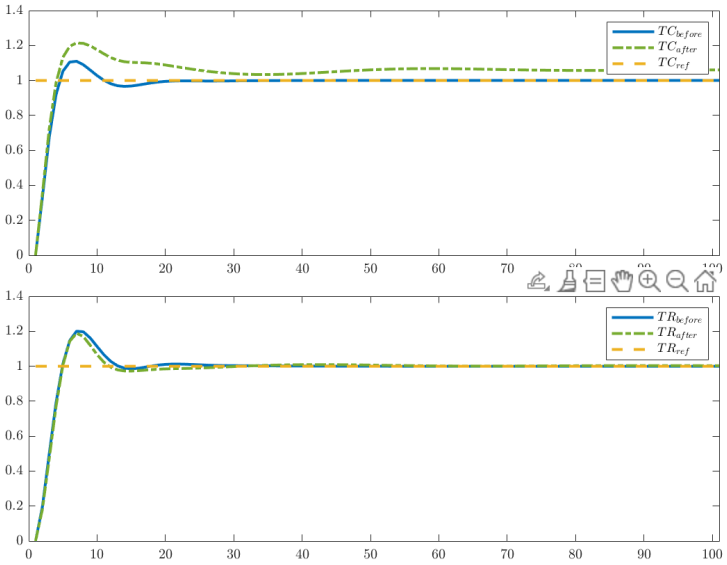

Figure A. 5: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 6.8×10^{-3}, noise variance = 2×10^{-3}, noise variance decay rate = 8×10^{-6}, discount factor = 0.96.

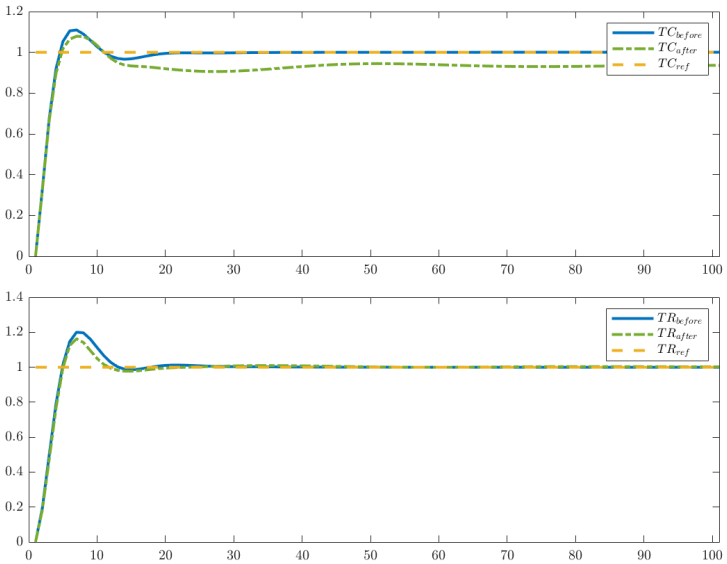

Figure A. 6: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow lines represent the set points. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 6.4×10^{-3}, noise variance = 2.5×10^{-3}, noise variance decay rate = 1×10^{-5}, discount factor = 0.92.

Appendix B: Additional Water Tank Model Simulation Results

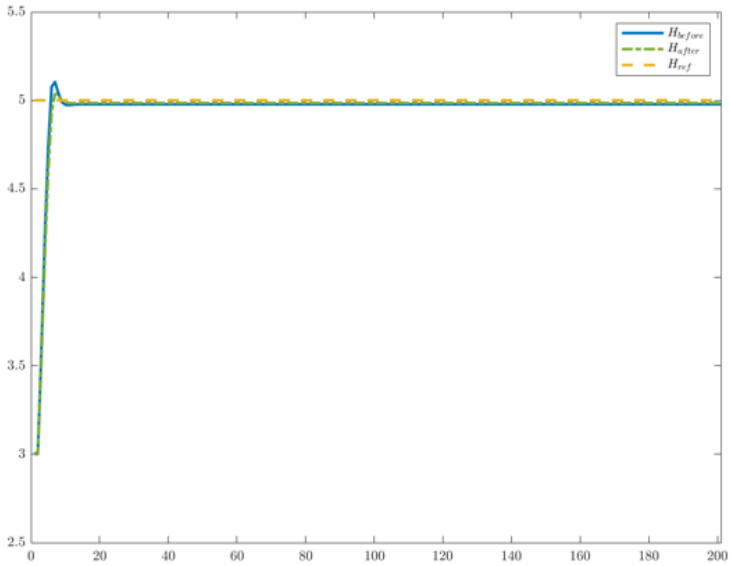

Figure B. 1: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow line represents the set point. Tuning parameters: actor learning rate = 2×10^{-8}, critic learning rate = 5×10^{-3}, noise variance = 0.5, noise variance decay rate = 1×10^{-5}, discount factor = 0.95.

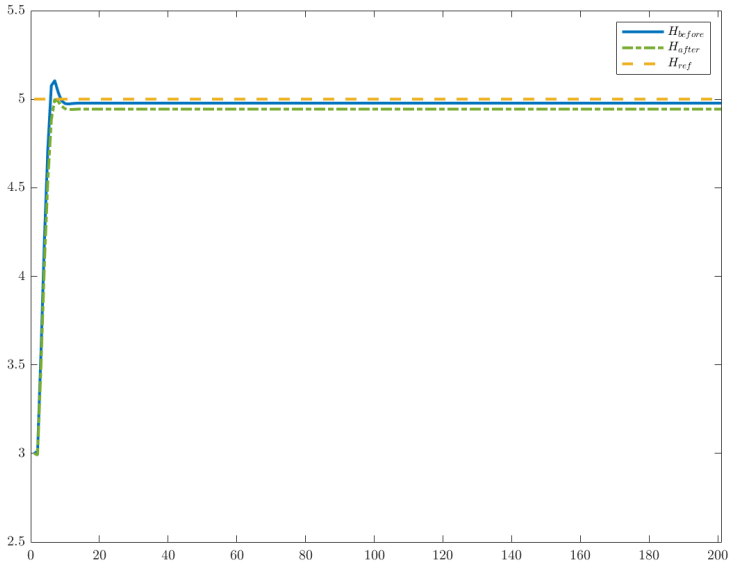

Figure B. 2: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow line represents the set point. Tuning parameters: actor learning rate = 2×10^{-8}, critic learning rate = 1×10^{-3}, noise variance = 0.5, noise variance decay rate = 1×10^{-5}, discount factor = 0.95.

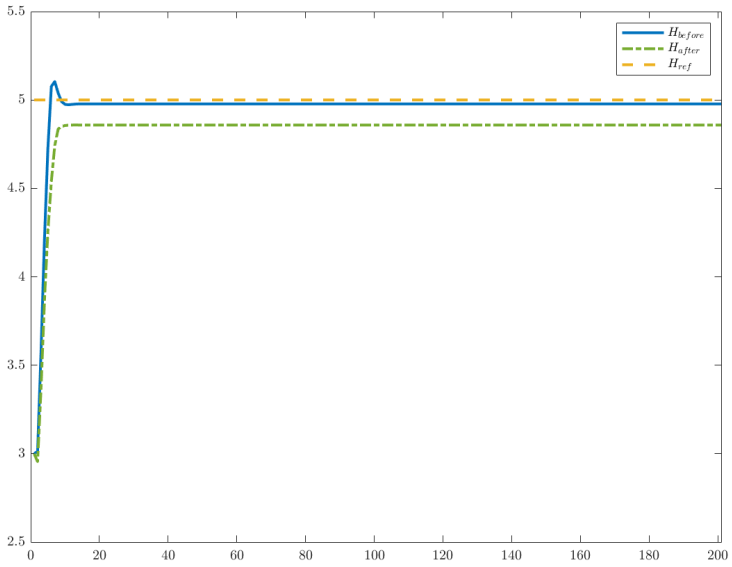

Figure B. 3: The performance comparison between the RL controller before online training (solid blue line) and that after online training (dot-dash green line). The dashed yellow line represents the set point. Tuning parameters: actor learning rate = 1×10^{-7}, critic learning rate = 1×10^{-3}, noise variance = 0.35, noise variance decay rate = 1×10^{-5}, discount factor = 0.95.